Discovery of Function Forms in Three Variables

by

Ziqiang Wang

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 1998

0-612-38280-X

Abstract

To find a mathematical description of a certain class of events is the goal of mathematical modeling. Traditionally, it is the task of mathematicians and engineer-scientists. The goal of function form discovery is to develop machine intelligence systems to tackle this problem. Though the machine intelligence approach is still in its infancy, it has been demonstrated that systems based on such approach are able to give more compact and meaningful forms that describe the input data than the traditional numerical methods.

This thesis presents a function form discovery system known as FFD-II which is a significant extension of the FFD system. The adoption and extension of the data transformation mechanism of FFD allows FFD-II to discover a significantly wider variety of functional forms from numerical data than its predecessors. FFD was developed initially for finding real-valued function forms of one independent variable. It could also be used to find families of functions in an indirect way. FFD-II is able to discover function forms of two independent variables directly from numeric data for it can make use of three dimensional information that cannot be used by the indirect methods which, for example, have to rely on "cross-effects" in the discovery. Hence, FFD-II not only exhibits better performance in handling the discovery problem, but is also more flexible for future extensions. Another significant characteristics of FFD-II is its new adaptive error control. It identifies the noise patterns according to the smoothness of an observed functional image and monitors the magnitude of propagated errors according to the theoretical error analysis results. In FFD-II special treatments are also added to reduce the effects of noise. Hence, the new system has a greater tolerance to both the computational error as well as the noise of the input than FFD.

Other new contributions of FFD-II include: 1) the construction and analysis of a three dimensional based function form description language; 2) the design of special purpose numeric methods which can recognize primitive functional patterns, conduct factorization and handle partial differential transformations of three dimensional data; 3) the quantified measurements of the qualitative characteristics of a functional image and 4) the implementation of a new heuristic search process.

Acknowledgements

I wish to express here my greatest appreciation to Professor Andrew K.C. Wong for his tremendous support and encouragement over the years. He constantly impressed me with his knowledge, insight, open-mindedness and generosity as he guided me throughout my research. I am also deeply grateful for his careful review of this thesis.

In a very special way, I would like to thank Dr. Puiwing Wong, to me who has been a constant source of inspiration and expert advice. Special thanks are also extended to the other members of my Ph.D. committee, including Professor Glenn R. Heppler, Professor Mohamed Kamel, and Professor Paul Thagard. They have given me some most valuable suggestions and helped reviewing this thesis. My sincere gratitude also goes to my external examiner Professor Howard J. Hamilton. He read through the thesis with great patience and kindly helped me with a number of corrections.

I am very grateful to all the people who run the PAMI Lab (Pattern Recognition and Machine Intelligence Lab), for its unique magnificent environment throughout the years. I consider myself very fortunate to have them as my colleagues and friends.

Last, but by no means the least, my deepest gratitude goes to my wife Li and our son Jeff for their sacrifice and great patience. Indeed, their understanding, inspiration, encouragement and love were what carried me through the course of my Ph.D work. Then, of course, there is my warm-hearted sister Chaoying Wang and her loving family. They made my stay in Waterloo a most enjoyable period in my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mathematical modeling is one of the most fundamental stages of scientific theory formation. This task is a very complex intellectual activity and has traditionally been the realm of the most talented human experts. Machines have been used only as a computing device to aid human experts to process large amount of data. With the development of cognitive science and artificial intelligence, the efforts on machine synthesis of this human intelligent activity has been receiving more attention in the last decades. Various methodologies has been proposed and a number of carefully specified machine discovery systems have been created. It has been demonstrated that, to certain extent, a machine can take over not only the tedious data analysis work from human scientists but also the modeling task itself.

However, the research in this field is only a start and there is still a long way to go. Addressing the problem of function form discovery, this research is a step forward to the goal of computer automatic mathematical modeling.

1.1 Machine Learning and Machine Intelligence

Simon defined learning as "*Learning denotes the changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time*" [55]. As a science of the artificial, machine learning is a research area of machine intelligence. It seeks an algorithmic solution to the problem of modeling human learning activities.

1.1.1 Machine Intelligence

In the early years when Machine Intelligence started as a field whose goal was to mimic human intelligence in a machine, people were excited about dreaming what machine could do for them. However, they soon realized that they overlooked the difficulty of the job. Human intelligence is indeed very complex.

To replicate human intelligence we at least have to know the way to decompose the intelligent activities into appropriate parts and the interfaces that can bring the parts together. Unfortunately, we know very little about this so far. However, research of last few decades has shown the possibility of developing intelligent machines with many working systems. It has been generally acknowledged that

Firstly, the research in artificial intelligence, cognitive psychology, and scientific philosophy cofertilize each other[36, 54, 62].

Secondly, machine intelligence does not have to mimic human intelligent skills. Human intelligence represents just one point in an uncharted space of possible means of acquiring knowledge and skills[7]. It is commonly believed that the human brain can adopt new knowledge in an "optimal way", although the process can be very long. A machine's superiority, on the other hand, is its power in conducting numerical and symbolic computations. Moreover, knowledge and skills can be shared between different systems by simply "copying". These properties suggest that machine intelligence could be different from human intelligence. Theoretical analysis provides a means of exploring the possible methods, while the task-oriented approach provides a vehicle to test and improve the performance of functional intelligence systems. In this way, particular approaches to intelligence issues could be tested in a well understood problem space.

Lastly, high level intelligence is an integration of lower level intelligences. This applies to both the biological intelligence and the machine intelligence. Researchers in the field of machine intelligence have been successful in implementing intelligence in many subproblems in a variety of specific domains. Some implemented systems did even better than a human problem solver. The story of Deep Blue, a powerful supercomputer and an extraordinary chess player, defeating human world champion Garry Kasparov in 1997, is just another example. Cumulative successes not only gives us an insight into the issue of intelligence, but also provides us with a continuously expanding base for the fulfillment of new success[1].

Machine intelligence has been developed along two lines: one attempts to mimic human thinking and the other takes advantage of the computing and formal inference power of codifiable machines. However, in the foreseeable future, to replicate the full gamut of human intelligence is unrealistic. Gaining knowledge through theoretic research, applying this knowledge in working machine intelligence systems, and further developing systems that assist people in a variety of well specified tasks will remain the primary goal of machine intelligence research for the foreseeable future.

1.1.2 Machine Learning and Machine Discovery

The ability to learn is central to human intelligence and implanting learning capabilities in machines is one of the main goals of Machine Intelligence research. A system is said to learn from its environment if it improves its performance in interacting with the environment (skill acquisition) or it abstracts new knowledge from the environment (knowledge acquisition)[44, 41]. Samuel's checker playing system[50] is an example of skill acquisition learning. The system included a series of parameters each of which was able to take new numerical values. To improve the system's performance, these values were adjusted by training samples.

[1] Rodney Brooks[4, 5] demonstrated incremental machine intelligence with a mobile system called CREATURE. The system was created by decomposing the system into parts according to function and activity. All the pieces were implemented using known AI technologies and then interfaced into a complete system.

The second type of learning, known as knowledge acquisition learning, relates to the discovery of new knowledge. In the last few decades, developing machine intelligence systems with such capability is one of the most vital research area in the machine intelligence field. Generally speaking, most learning systems are also discovery systems to a certain degree. More or less, they use inference strategies to a certain level to discover new knowledge. However, in the machine intelligence literature, Machine Discovery is an unsupervised learning process seeking an accurate, concise and meaningful description of regularities or general rules to explain all or at least most observations[45, 65]. This form of learning includes conceptual clustering, constructing classifiers, fitting equations to data, discovering laws explaining a set of observations and formulating theories accounting for the behavior of a system. Since the discovery system relies solely on the observation data, it requires the greatest amount of inference.

1.1.3 Machine Intelligence in Scientific Discovery

Research in artificial intelligence and cognitive simulation has shown that the mechanisms of scientific discovery can be subsumed as special cases of the general mechanisms of problem solving [39, 63, 57]. Based on this claim, scientific discovery activities are computationally codifiable.

There are two major forms of scientific discovery, the *generation of empirical laws* and the *formation of theories* [62, 63]. The former involves descriptive generalizations that summarize observations and the latter involves postulating unobserved structures or processes. Concerning the generation of empirical laws, researchers in machine learning and machine discovery have investigated three main aspects of empirical discovery in recent years:

- *Taxonomy Formation* [64, 23]. Research on conceptual clustering [12, 42] addresses this problem by organizing a set of observations into a conceptual hierarchy, which can then be used to classify new observations.
- *Generation of Qualitative Laws* [43, 37, 19, 51, 52]. In this case, the goal is to uncover qualitative form relations that hold for a set of observations.

- *The Production of Quantitative Laws* [10, 21, 40, 69, 65, 48]. The task of this aspect is to find mathematical relations between numeric variables.

Some researchers have tried to integrate these three aspects into single systems [38, 39, 47]. However, this research addressed only the third aspect.

1.2 Function Form Discovery

In science and engineering, extracting mathematical models from numeric data is of fundamental importance. Various numerical analysis (e.g., interpolation and polynomial approximation algorithms, curve and surface fitting, etc.) and statistical methods have been applied successfully to problems in science and engineering. Traditionally, this task can only be assigned to human experts, who use machines to perform numerical calculations.

Today, the effort to endow machines with the capability of automatic modeling has become an important branch of machine learning and machine discovery. As stated in the previous section, the mechanisms of human experts' scientific behaviors are indeed the mechanisms of problem solving. The possibility of shifting the tedious task of mathematical modeling of numerical data from man to automatic machinery has been demonstrated by a number of implemented function form discovery systems, e.g., BACON by Langley in the 1980's[30, 31, 29], FFD by Wong in 1991[65], and LINUS by Phan in 1994[48].

Function form discovery is a form of empirical discovery. In the literature of machine learning, it is also classified as learning by induction, learning from examples, learning from observation and discovery, quantitative learning, or unsupervised empirical learning. The theoretical basis of this paradigm is empirical inductive generalization in which the system creates an inductive hypothesis on the basis of the given examples from the external source of information utilizing primarily domain-independent background knowledge. When the given examples are numerical data, namely a set of observed data, and the goal is to construct function form descriptions (continuous numerical functions, either explicit or implicit) that summarizes the relations of the variables involved in the given data, the paradigm is re-

$$+\frac{1}{36}x^3y^3 - \frac{1}{120}x^5y - \frac{1}{120}xy^5 + \frac{1}{720}xy^6$$
$$+\frac{1}{720}x^3y^5 + \frac{1}{2880}x^5y^4 + \frac{1}{120}x^5 - \frac{1}{6}x^3 + \frac{1}{2}xy^2$$
$$-\frac{1}{6}xy^3 + \frac{1}{6}x^3y - \frac{1}{12}x^3y^2 + \frac{1}{24}xy^4 - xy$$

The second formulation system may find, from the same data a trigonometric formula:

Representation II:

$$z = \frac{\sin(x)}{e^y}$$

According to the "Parsimony" and "Transparency" criteria, "Representation II" is a better formulation. It contains only two functional terms, $\sin(x)$ and e^y. Thus it is more compact than "Representation I", which contains thirty terms. In the meantime, "Representation II" is easier to interpret. Each term in the formula matches a geometric property, e.g., the oscillatory characteristic regards to x and the deamplification is governed by y. "Representation I" reveals very little about these underlying functional relationships. It is hard to translate the fitting parameters into meaningful related properties.

Today, the research field of function form discovery is still in its infancy. Up to the late eighties, function form discovery systems had been created in two categories. Formula construction[2] based approaches, such as BACON [26, 27, 28, 32, 34, 35] FAHRENHEIT [69, 72, 20] and ABACUS [10, 11], can only discover polynomial and rational function forms. Data analysis based approaches, such as E^* [52, 53], can only discover function forms defined in the system's protocols. The new data transformation based method launched in the early nineties, such as FFD [65] and LINUS [48], is still incomplete with many open issues. Nevertheless, it has been demonstrated that automatic function form discovery system can be used to assist in automatic knowledge acquisition, extraction of relevant knowledge from large knowledge bases, and abstraction of higher-level concepts out of data sets.

[2] We classifies the existing methodologies into three categories, namely formula construction, data analysis and data transformation. Details will be given in Section 2.1.

1.3 Motivations of This Research

Up to the present, most research on inductive leaning has been concerned with qualitative learning that creates conceptual, logic-style descriptions from the given facts. In contrast, this research, following the work of FFD and LINUS, attempts to address the quantitative learning that deals with numerical laws (more specifically, function forms) characterizing empirical data. Moreover, this research focuses on the three-variable function form discovery problems. It is motivated by the following theoretical and practical concerns.

1. From the point of view of mathematical modeling in science and engineering, this research provides an alternative to numerical analysis and the simulation of complex models. As a tool that combines traditional numerical analysis techniques with the artificial intelligence, it may serve as an intelligent assistant for human researchers in scientific studies and engineering developments to find functional regularities hidden in raw data. Research in traditional numerical analysis methods focuses only on the mathematical issues of accuracy and convergenc. They cannot be used to find a meaningful and compact numerical relationship from empirical data without the aid of human expertise. This research seeks an automation of mathematic modeling that emphasizes not only the "justification" but also the "transparency" and "parsimony".

2. From the machine discovery point of view, this research is needed to meet the growing requirements for high quality of quantitative discovery. In many fields of science, data-driven discovery is an important and powerful general theory formation method. Researchers gather empirical data as a prerequisite for building models and then search for a set of generalizations or theories to interpret physical world. For the automation of this process, function form discovery is usually the fundamental starting point for deep modeling of knowledge in machine.

   Current research in automatic empirical discovery focus mainly on qualitative rule or law discovery. Only very little effort has been put into quantitative discovery. The shortage of high quality function form discovery system is becoming an obstacle to the application of machine discovery in solving scientific and engineering problems. This research is aimed at developing an intelligence system which could engage as an intermediate processor in a scientific theory formation automation. The system processes the given empirical numeric data and provides succeeding discovery processes with high level knowledge, or more specifically mathematical formulas with high quality in terms of justification, parsimony and transparency.

3. This research demonstrates the flexibility of the data transformation approach. Prior to the introduction of the data transformation method, all function form discovery systems suffered a common limitation of being able to discover only function forms within a very limited number of function form classes. FFD introduced the data transformation technique which can discover function forms in a significantly wider range than previous approaches. However, there are considerable open theoretical and practical issues needed to be addressed with new implementations. A demonstration of how this methodology works in multi-variable function form discovery problems is surely worthwhile.

4. This research specially addresses the three-variable function form discovery problems for the following reasons. Firstly, multi-variable problems are commonly confronted in scientific research and engineering development. Secondly, solving three-variable problems is usually a starting point for addressing higher dimension problems. Lastly, the indirect approach to the multi-variable function form discovery problem taken by FFD does not allow the system to fully take advantage of data transformation technique as it is subjected to some constraints when dealing with real world problems[3].

5. As a research in the field of machine intelligence, this research also shares the general motivations with other research in the field, such as providing new philosophical perspective for investigations in cognitive science, and enriching the AI technology by making a worthwhile progress in such an important task.

[3]More details will be given in Section 4.2.2.

1.4 Organization of the Thesis

The remainder of this dissertation is organized into five chapters. Chapter 2 is a review of related work. The review is organized into three categories according to the core of the discovery systems. The current states of this research area is presented.

Chapter 3 establishes the theoretical foundations for the proposed methodology by first drawing conclusions from the review with the mechanism to be selected to build the new discovery system. A formal statement of the research problem is then given. The discussions that follow the problem statement will focus on introducing the function form description language used by the FFD-II system and the theoretical issues concerning the expressiveness and redundancy of the language.

Chapter 4 presents detailed design issues and the system implementations. Three important issues will be investigated. They are: first, why indirect methods cannot provide the advantages that the central mechanism provides; second, why error control is important in multi-variable problems and how to adaptively control the errors; and lastly, what numeric tools should be used to conduct the numeric computations involved in the discovery process. To introduce the adaptive error control method, both theoretical analyses and choices of numeric tools are presented.

Chapter 5 reports the experiments run with FFD-II. Experiments are organized into four groups. Each group emphasizes only one key issue. The first group is "Randomly Selected Functions" that verifies the fundamental discovery ability of the system. The second group is a comparison between an indirect data transformation based system and FFD-II. The third group is an extensive verification on the system's ability to model complex function forms represented by random surfaces. The last group is a test of the system's ability to handle input data with added noises.

The final chapter, Chapter 6, will conclude this thesis by highlighting the contributions and outlining the directions for future investigations.

Chapter 2

Computational Function Form Discovery

This chapter reviews previous research in the area of computational numerical law discovery, or more specifically, real value function form discovery systems. As the focus of this research is autonomous function form discovery systems, this review will concentrate on artificial intelligence systems that discover real-valued numeric relations. Although various numerical analysis methods solve the same problem of finding analytic descriptions from numeric data, we view them as mathematical tools that can be used by machine intelligence function form discovery systems. I will first classify the existing methodologies, and then review the related work accordingly.

2.1 Methodology Classification

According to the amount of inference and the techniques employed, function form synthesis methodologies can be classified into three main categories: *Numerical Analysis*, *Formula Construction* and *Data Transformation*. Some systems employ combined methodologies, but we characterize them into one of these categories according to their central approach. The two most basic techniques used in function form discovery systems are data transfor-

performing different data transformations.

Since a numeric analysis system makes its discovery based solely on predefined prototypes, and only a limited number of prototypes is defined, this method has limited capability in discovering the rich variety of function forms in scientific study and engineering practice. Only when the underlying functional relation is covered by its predefined prototypes, can this method perform a successful discovery under the criteria of justification, parsimony and transparency. However, this limitation could largely depend on the domain knowledge of human experts who create the system to solve problems in a specific application domain.

In contrast, formula construction methods do not make discoveries directly from the functional pattern matching. The discovered function is constructed under the guide heuristics for of identifying some features. As the features usually include only the simplest ones, such as monotonicity, oscillation and constancy, formula construction methods are located at the other extreme of the technique spectrum opposite to the numeric analysis methods. Minimum effort is applied in analyzing the data. The system's capability to discover relies largely on how new theoretical terms are constructed. The degree of inference used in formula construction is higher than that used in the numeric analysis method.

Traditionally, data transformation is a "pre-processing" step which serves to simplify the data before other numerical tools can be used. Application of a particular transformation may be motivated by the need to remove non-linearity, to decompose complex features into fundamental ones, to filter out noise or to capture certain global properties. It can be used in both computational mathematics and pure mathematics research. The choice of transformation is not only highly domain dependent but also guided by human cognition. In the function form discovery system that uses data transformation technique, the capability of the system largely depends on the transformation set constructed by human experts who create the system.

Let

$$\{ u_1, u_2, \cdots, u_n \}$$

be the set of variables related to a problem under study. A transformation is a one–to–one mapping $\mathcal{M}$ from $\mathcal{R}^n$ to $\mathcal{R}^m$:

$$\{ u_1, u_2, \cdots, u_n \} \overset{\mathcal{M}}{\longmapsto} \{ v_1, v_2, \cdots, v_m \} .$$

The type of the mapping $\mathcal{M}$ determines the type of the data transformation. There are three major types of data transformations related to continuous real-valued variables.

- *Algebraic Transformation.* Mapping $\mathcal{M}$ is expressed by analytic functions, usually explicitly. Most important geometrical transformations, such as rotation and scaling, are algebraic transformations.

- *Integral Transformation.* Mapping $\mathcal{M}$ contains integral operations. In general the integral transformation takes the form of

$$\begin{aligned} F(x_1, x_2, \cdots, x_n) = \int_D f(x_1, x_2, \cdots, x_n) \cdot \\ K(x_1, x_2, \cdots, x_n\ v_1, v_2, \cdots, v_m)\, dx_1 dx_2 \cdots dx_n , \end{aligned}$$

  where $D \subseteq \mathcal{R}^n$ is the integration domain, x's are the original independent variables and v's are the new independent variables. K is known as the kernel function of the transformation. Apparently, the transformed image is affected by all the points in domain D. Thus integral transformations are able to highlight certain global properties of the data.

- *Differential Transformation.* As its name indicates, differential transformations involve the description of the data in terms of their derivatives, or differences in the discretized situations. Differential transformations can reveal important analytic properties, such as slope and convexity etc., and provide elegant graphical descriptions of highly complex behaviors of nonlinear dynamic systems[14]. Many scientific terms and laws are represented in the form involving derivatives.

Some examples of transformations are listed below, where $\{x_1, x_2, \cdots, x_n, w\}$ are the relevant variables, x's are independent variables and w is dependent variable,

- A simple algebraic transformation which reducing powers into products and products into sums is the 'logarithm transformation':

$$\{x_1, x_2, \cdots, x_n, w\} \longmapsto \{x_1, x_2, \cdots, x_n, \log(w)\}.$$

For example, by the transformation $w = \log(z)$ the functional relation $z = x^y$ becomes $w = x\log(y)$, and the functional relation $z = xy$ becomes $w = \log(x) + \log(y)$.

- 'Trend removal' transformation

$$\{x_1, x_2, \cdots, x_n, w\} \longmapsto \{x_1, x_2, \cdots, x_n, w - f(x_1, x_2, \cdots, x_n)\}.$$

is an algebra transformation, where the function f describes the trend of a process.

- Fourier transformation [58] is the most famous integral transformation used in pure and applied mathematics and it plays an important role in communication theory and technology.It transforms a physical space to the frequency space. The general form of the multi-dimension Fourier Transformation is

$$\begin{aligned} \hat{f} &= F[f] = \int_{-\infty}^{\infty} e^{\,i\,x_1 y_1} \int_{-\infty}^{\infty} e^{\,i\,x_2 y_2} \cdots \int_{-\infty}^{\infty} e^{\,i\,x_n y_n} \\ &\qquad \cdot f(x_1, x_2, \cdots, x_n) dx_1\, dx_2 \cdots dx_n . \end{aligned}$$

where $f(x_1, x_2, \cdots, x_n)$ is the function to be transformed. Two other well known integral transformations include Laplace Transformation and Mellin Transformation [59, 6].

- A simple differential transformation in a three dimension Cartesian coordinate system (x, y, z) is the mapping:

$$(x, y, z) \overset{\mathcal{M}}{\longmapsto} \left(x, y, \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \right).$$

This transformation transforms a two dimensional scalar field $z(x, y)$ into its gradient vector field, if the last two terms are viewed as the coordinates of two dimensional vectors.

A function form discovery system that uses data transformation technique simplifies a given observation functional image using the simplification tools of data transformations embedded in its tool-box. The discovered function form is expressed in terms of a transformation sequence along with a simplified matching functional pattern. Hence a well organized transformation set is the key to taking advantage of data transformations and enabling the system to cover a wide range of complex function form classes. The difficulty of the function form discovery problem hinges upon the expressiveness of the description language, i.e. the way of how the system express its finding. In general, the more expressive the language, the more difficult it is to find a specific formula. A discovery system must strike a balance between the language's expressiveness and the cost of identifying one particular member from the set of all possibilities [48].

2.2 Formula Construction Approaches

2.2.1 BACON

BACON [37, 38, 39] is the most well known machine intelligence system specifically designed for automated discovery of quantitative laws from numerical data. It discovers numeric laws by analyzing the relationships between variables from data provided by examples. A number of discovery systems can be grouped with BACON since they use formula construction heuristics similar to BACON's. Table 2.1 lists some of the discovery systems in the BACON family.

According to our methodology classification, the basic discovery strategy of the BACON systems is formula construction. BACON.1 starts the discovery with a table of numerical values of relevant variables provided. Four simple heuristics (or rules) are employed by the system for driving the search to the goal.

1. If Y has the value V in a number of cases, then hypothesize that Y always has the value.

System	Year	Key Features
BACON.1	1978	Trend and constancy detectors
BACON.2	1979	Specialized method for finding constant differences
BACON.3	1981	Trend and constancy detectors Recursing to higher levels of description
BACON.4	1980	BACON.3 *plus* Intrinsic property method Common divisor method
BACON.5	1981	BACON.4 *plus* General method for finding constant differences Expectation-based methods
BACON.6	1983	BACON.5 *plus* Hill–climbing method for dealing with noise
ABACUS	1986	BACON.3 for equation formation Dimension analysis Domain splitting Logical expressions description
FAHRENHEIT	1987	BACON.4 for equation formation Scope determination
IDS	1989	Qualitative process representation Correlation analysis

Table 2.1: BACON Like Systems

2. If X and Y are linearly related with the slope S and the intercept I in a number of cases, then hypothesize that this relation always holds.

3. If X increases as Y decreases, and X and Y are not linearly related, then define a new term T as the product of X and Y.

4. If X increases as Y increases, and X and Y are not linearly related, then define a new term T as the ratio of X and Y.

In the discovery process of BACON, the system carries out a beam search, in which only a certain number of pairs of terms with the highest correlations are used to find fundamental patterns and construct new terms. Then the regularities of constancy, linearity, increasing and decreasing trends are detected for a selected pair of terms. This is accomplished by simple arithmetic operations. The detection of a regularity triggers the construction of a new term corresponding to the slope (heuristics 2, for linear relationships), the ratio (heuristics 4, for increasing trends) or the product (heuristics 3, for decreasing trends). A law is attained when the data can be related together as one final constant or linear relation (heuristics 1 and 2). Thus inductive inference is performed by the production rules which generate the terms.

In its later versions, some new features were added to enhance the system's capability (refers to Table 2.1). For example, the difference technique allows the system to discover polynomial relations and the recursive technique enables the system to deal with multi–variable tasks.

BACON is an important system because:

- It is the first machine intelligence systems that employs formula construction approach to the function form discovery problems;
- BACON itself has several successive versions concerned with slightly different aspects and components [Langley, 1978, 1979, 1981, Langley et al. 1981, 1982, 1983b, 1984, Bradshaw et al. 1980, 1983a];
- Many different quantitative law discovery systems adapt BACON's discovery strategy. For example, FAHRENHEIT [20, 69], which adapts BACON.4 as its formula discovery machine, is an extension of BACON that determines the scope of the discovered formulas, and ABACUS [10, 11, 16] and IDS [46, 47] adapts BACON's strategy as their equation formation components;
- It can be easily integrated with other qualitative discovery strategies to create a discovery system that performs both qualitative and quantitative empirical discovery, e.g. GLAUBER, STAHL, and DALTON [39, 70], ABACUS, and IDS;
- BACON is a very clear and thoroughly tested algorithm and may be used as a standard by which subsequent systems are evaluated.

BACON was evaluated by a number of scientific law rediscovery tasks. However, due to the very small number of production rules used to recognize features for triggering transformations, it can only find rational functions. Though it could be argued that any continuous function can be approximated to any order by a polynomial, this restriction to the rational function class is a great drawback from the point of view of parsimony and transparency. This limitation is also shared by systems that use BACON's discovery strategy. Nevertheless, considering its small set of production rules and plausible application in the field of elementary chemistry and physics, BACON is one of the most important systems in the history of machine intelligence approach to the function form discovery problems.

An example of BACON's formula construction is the rediscovery of Kepler's third law. Table 2.2 illustrates the terms that were constructed for the discovery of Kepler's third law.

Plant	Observation Data		Term 1	Term 2	Term 3
	Distance(D)	Period(P)	D/P	D^2/P	D^3/P^2
A	1.0	1.0	1.0	1.0	1.0
B	4.0	8.0	0.5	2.0	1.0
C	9.0	27.0	0.333	3.0	1.0

Table 2.2: An Example of BACON's Formula Construction

The discovery system was first given a set of observation data related to a pair of original variables Distance(D) and Period(P). The detected trend is a increasing trend, i.e. P increases as D increases. Thus a new term D/P is constructed according to Heuristic 4 (page 17). Since no linearity and constancy is detected in the new generated term, the construction process will go on. Based on the observed decreasing trend of term D/P as D increases, the second new term D^2/P is constructed, and then the third, D^3/P^2 is constructed from D/P and D^2/P. The last term is found to be constant, thus it leads to the discovery of the function form $D^3/P^2 = 1$.

2.2.2 ABACUS, FAHRENHEIT and IDS

BACON searches for a function form description based on trend analysis. Clearly, if there is no trend detected, or in other words, the underlying function is non-monotonic, the system will not find the solution. For example, if the observation range is $x \in [-1, 2]$ and the underlying function relation is $y = x^2$, no new term will be constructed by BACON since the observation data set is not monotonic, in spite of that the functional relation could be discovered by a two-step term construction: Term1 $= y/x$ and Term2 $=$ Term1$/x = y/x^2 =$ 1 (Constant). ABACUS addressed this problem by integrating qualitative discovery with quantitative discovery.

In ABACUS' equation learning, it searches for the best equations to describe the observed data with a set of inductive rules similar to those used by BACON.3 (a released and enhanced version):

1. If X and Y are qualitatively proportional to a user-specifiable degree, generate new terms x/y and $x - y$.
2. If X and Y are inversely qualitatively proportional to a user specifiable degree, generate new terms xy and $x + y$.
3. If a term X is found constant for all events, the learning task is completed.
4. If a term X is found constant for a subset of the events, the subset is removed from the list of events and associated with the equation describing it.

Rule 4 allows the system to find multiple equations to describe piecewise functions. To speed up the searching, ABACUS also employs rules based on three domain-independent constraints. They are unit compatibility rule (prohibiting the generation of the terms with incompatible unit), redundancy detection (prohibiting the generation of the terms which are mathematically equivalent yet syntactically different) and tautology detection (prohibiting the generation of the terms which are simplifiable by mathematical cancelation).

6. Periodic with increasing (or decreasing) amplitude,
7. Periodic without trend,
8. Periodic with trend.

When the first pattern is observed, the system will halt. If the second pattern is observed, backtracking will be invoked. When one of the remaining six patterns is observed, the system constructs new hypotheses according to the rules in its knowledge base, utilizing the given protocols and arithmetic operators. For example, if the pattern "periodic with increasing amplitude" is observed, the new hypothesis will be comprised of "the current hypothesis"$\pm$"periodic"$\times$"monotonic", and if a non-monotonic and non-periodic pattern is observed, it may infer that the residual contains a term of "monotonic"$-$"monotonic" or "monotonic"/"monotonic". Each new hypothesis is examined one by one. If the new hypothesis has a lower error than the current one, new residual image is calculated and the process is repeated. Otherwise, it discards the new hypothesis and checks the remaining hypotheses.

Gerwin reported fifteen tests. Twelve tests had three component functions, one had two components, one had a single component and one consisted of randomly generated data. The test came out with 40% accuracy, comparing with 41% accuracy rate of doing by hand.

Gerwin's algorithm is an artificial intelligence approach. The significance of this algorithm is that it has error tolerance ability and it can carry out function form discovery based upon only a very small observation data set (in Gerwin's test, only ten observation data points were used for a single discovery task). Thus, the algorithm is quite efficient. However, the drawback of this approach is obvious. The function forms that could be handled are very limited. Since the system constructs hypothesis solely based upon analytic pattern analysis and more than one protocols may have the same analytic property, it is not guaranteed that the best expression is obtained. In Gerwin's test, the system modeled $x^{3/2}\log(x)$ as $x^2 - x$ and $\sin(x) + x^{1/2}/x$ as $\sin(x) + e^{x/2}/x^{5/2}$. Moreover, since the protocols must have good significant analytic properties for the analysis step in this approach, it is hard to extend the function form coverage.

2.3 Numerical Analysis Approaches

2.3.1 E^* Algorithm

Unlike previous function form discovery systems designed to find functional relationships in numerical data independent of deep domain knowledge, Schaffer [52, 53] attacked function-finding problems by treating function form discovery as a classification task. His E^* algorithm decides, among a fixed finite set of parameterized formulas, which formula is most applicable to a given numeric observation data set. This differs from its previous work in two aspects. First, it concentrates on reliable identification of a few function forms rather than on heuristic search of an infinite space of potential relations. Second, it introduces the use of different concepts, such as "distinction", "significance" and "lack of fit", for evaluating apparent functional relationships. The algorithm can be considered as a numerical analysis function form discovery approach.

Observing that a large portion of bivariate functions that were published in the journal *Physical Review* in the early 1900's fall only within a small range of function forms, Schaffer proposed his E^* algorithm that emphasizes reliability rather than flexibility. Only eight possible choices are included in E^* algorithm. They are listed in Table 2.3. To decide among these choices, regression analysis of data is conducted, and three notions: *significance*, *distinction* and *systematic lack of fit* are used to measure the goodness of the fitting results.

Significance, a statistical measure, is used to measure the strength of a functional pattern in terms of how unlikely it is to have arisen by chance from purely random data. *Distinction* indicates how well a functional relationship approximates the observation data. In other words, it measures how different a candidate function is from other function forms with which it might be easily confused. The last notion "*Systematic Lack of Fit*" measures the possibility of describing the fitting residuals by another formula. If this is the case, then we could say that there is strong evidence that the relationship between given variables is not what the system has discovered.

Schaffer tested his algorithm with 352 sets of data of bivariate functions that were

No.	Expression
1	$y = k_1 \cdot x + k_2$
2	$y = k \cdot x^{-2}$
3	$y = k \cdot x^{-1}$
4	$y = k \cdot x^{-1/2}$
5	$y = k \cdot x^{1/2}$
6	$y = k \cdot x^{1}$
7	$y = k \cdot x^{2}$
8	NULL

where k_1, k_2 and k are parameters

and NULL *means "No Relationship Identified".*

Table 2.3: Prototypes of E^* Algorithm

published in the journal Physical Reviews in the early 1900's. The results were interesting. Compared with BACON, it performs equally well in identifying the correct formula in 30% of testing cases. However, E^* was much less likely to select an incorrect formula as the solution. BACON gave 30% incorrect answers while E^* only gave 10%.

Relying only on statistical analysis, E^* has a relatively large tolerance to noise. From the philosophy of scientific discovery point of view, E^* brought some new terms into its discovery process that were not considered by its previous systems. First, deep domain knowledge is normally brought to bear in scientific analytic work and can usually reduce efforts. E^* only selects 7 possible fitting function patterns but performs quite well in its special problem domain. Second, to evaluate the findings, we need combined criteria for identifying a potential solution during the discovery process (such as the "significant" and "distinction" notions in E^*). Lastly, to reduce the chance of incorrect result, we need to give a discovery result along with a confidence measure (e.g. the notion of "systematic lack of fit" used by E^*).

The drawback of this approach is that, because the set of possible solutions is limited, it is effective only in those cases where the predefined formulas include the unknown function. For applications in a wide range of discovery tasks, the set must be expanded significantly and the three criteria must be modified accordingly to take into account the new formulas. Nevertheless, this kind of *ac hoc* modifications cannot improve the potential of this approach too much. Thus the major limitation of the E^* algorithm and is its inflexibility in dealing with a wide range of function forms.

2.3.2 KEDS

KEDS [49] is a function form discovery system which deals with piecewise functions. After failing to integrate CLUSTER2 [61], a cluster algorithm, and ABACUS into a piecewise function form discovery system of useful for engineering, Rao and Lu observed that "in order to discover models for engineering domains, the task of partitioning the domain space should be closely linked to the relationships that are to be discovered". This observation led to the development of KEDS, a two–phase discovery system. The partitioning is model driven and is based upon the relationships that are discovered from the data, while the discovery process is restricted within the boundaries of the regions created by the partitioning.

KEDS requires generalized knowledge about the kinds of relationships that are expected to be obtainable. This knowledge is expressed in the form of parameterized equation templates. KEDS first tries to fit the observation to one of the template functions. If it fails to obtain an acceptable fitting, it tries to partition the domain by sample clustering. After partitioning, equation fitting is carried out within each region again. This process is repeated until an acceptable piecewise function form is obtained.

Since KEDS is destined to solve real–world engineering problems, it employs only polynomials as its template functions. In terms of accuracy and efficient and meaningful partitioning, it achieves its goal within a limited set of function forms. However, its drawback is obvious. Since the solutions can only be piecewise polynomial functions, in terms of parsimony and transparency, the system cannot generate quality solutions.

2.4 Data Transformation Approaches

2.4.1 FFD

FFD uses the data transformation approach, introduced by Wong[65] in 1991. Since this research uses the same method and falls in the same category, the next chapter will be devoted to the fundamental issues of this approach while only a brief review will be given in this part.

FFD approaches function form discovery problems with a two-phase model, *feature simplification* and *function form abstraction*. The former is implemented by successively applying data transformations selected from a set of transformation classes which are predefined in the system's applicable *transformation set*, and the latter is done by numeric fitting to one of the function prototypes predefined in the system's *primitive function set*.

During its discovery process, the system searches for the transformation sequence that transforms the initial given functional image into a recognizable simple image. Heuristics, based on the measurement of the simplicities of the transformed functional images and the complexities of the total transformation sequences that have been applied, are employed in a best-first search. Once the system identifies a transformed image as a primitive image, a function form is declared to be discovered. The system reports the transformation sequence, possibly along with a set of descriptive parameters, and the final matching primitive function as the discovered solution.

Five general purpose data transformations were included in the FFD implementation. They are *logarithm*, *function inverse*, *reciprocal*, *factorization* and *differential*. The primitive function set is composed of three classes of polynomial functions.

$$\begin{aligned} c_1y^2 + c_2t^2 + c_3t + c_4 &= 0, \quad \text{for } c_1 \neq 0 \\ c_1ty + c_2t^2 + c_3t + c_4 &= 0, \quad \text{for } c_1 \neq 0 \\ c_1t^2 + c_2y + c_3t + c_4 &= 0, \quad \text{for } c_1 \neq 0 \end{aligned}$$

FFD was tested with twenty randomly generated binary combination functions, such as $y = t + \arctan(t) + 1$, $y = e^{-t} + 1/(2t)$, $1/(1 - 2\log(t))$, and so on. Among them, fifteen matching solutions and two approximations were found. Besides the randomly generated binary combination functions, FFD was also tested with five nonlinear ordinary differential equations with closed form solutions. It found three accurate functional forms and one approximation.

Two extensions also enable FFD to deal with oscillatory functions of the form $y = A(x) \cdot s(\cos(w(x)))$, where A and s are two functions, and families of functions with one extra variable control parameter. The latter function form is a special form of three-variable function form. This research focuses on the discovery of three-variable function forms. Thus the details of the extension will be discussed later.

Considering that previous discovery systems can only discover function forms in a very limited number of functional classes, FFD did open a new era in its area. As a novel system that uses the data transformation approach, FFD introduced quite a number of new ideas. Some of them have not been incorporated, while others were implemented with the simplest method possible [65]. It is therefore too early to summarize the limitations of the data transformation approach. However, from the implemented systems in this category, we can draw the following conclusions from the general point of view:

1. Data transformation approach has a great potential in dealing with complex function forms.

2. Data transformation approach is a very flexible methodology which could be either used as a general purpose function form discovery methodology or tailored to meet the needs of special applications.

3. The flexibility of this approach is also indicated by the capability of adopting other function form discovery methodologies in a cumulative way. In other words, we can easily use data transformation technique at the top level of the architecture of an discovery system which assigns specified subtasks to some other embedded low level systems.

4. The complexity of the data transformation approach is usually high in both the numeric computation aspect and the computer memory requirement aspect. The system must be able to perform data transformation with acceptable accuracy level. That requires relatively large observation data set.

5. Since some selected data transformations can only be numerically implemented with certain accuracy, for example the differential transformation, to prevent the computational error to explode is very important for the successful discovery of a function form.

Our current understanding on the relationship between the system's capability and the choice of transformation set and primitive set is still superficial. Any activities that help us to gain theoretical knowledge, any experiments that enable us to gain practical understandings and any new implementations that achieve new capabilities would be proven beneficial to the progress in the research area of function form discovery.

2.4.2 Linus

Linus [48] is the second discovery system that takes data transformation approach. Its transformation set contains the five data transformations defined in FFD and its primitive function set includes three different classes of polynomials.

$$
\begin{aligned}
&y + c = 0\,, \\
&y^2 + c_1xy + c_2y + c3x^2 + c_4x + c_5 = 0\,, \\
&(c_1x^3 + c_2x^2 + c_3x + c_4)y + c_5x^3 + c_6x^2 + c_7x + c_8 = 0
\end{aligned}
$$

New features introduced by Linus include:

- Interactive experiment query according to the error level and minimum sample required to make a discovery.

- Automatic range splitting and subtasks formation based on the applicability of specific transformations. This allows the system to deal with non-monotonic function forms and to a certain level to deal with piecewise function forms.

- Multi–solution output.
- Solution refinement through parameter calibration.
- Subtask solutions merging by checking obtained solutions with different ranges.

LINUS was reported to have successfully discovered the function forms of four selected 2-variable functions (three of them are not monotonic), and the solutions for twenty linear and nonlinear ordinary differential equations. In one experiment, LINUS output nineteen different forms for the same observation data set. Thus the input numeric data can be interpreted by different ways.

Unlike KEDS' range splitting which is based on cluster analysis, LINUS' range splitting is based on the monotonicity of the observation data. The result is that LINUS cannot handle piecewise function forms whose dividing points is not the local maximum or minimum of the function. For example, LINUS could not find the following piecewise function forms

$$y = \begin{cases} 2x & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases}$$

and

$$y = \begin{cases} -x^2 & \text{if } x \leq 0 \\ x^2 & \text{otherwise ,} \end{cases}$$

though it can handle the piecewise form

$$y = \begin{cases} -x & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases}$$

The other shortcoming of LINUS is the incomplete use of observation data. When range splitting is needed, the program may drop some observation points around the splitting points to avoid infinite sample value, and the program does not make any effort later to get those points back. Furthermore, the program cannot provide us the necessary information about the splitting points which is important for piecewise function form description. LINUS's sub-solution merging strategies need to be improved significantly for handling real world function form discovery problems.

2.5 Other Approaches

COPPER

COPPER combines qualitative reasoning with quantitative reasoning. Its qualitative reasoning is based on dimensional analysis and the quantitative reasoning to address possible missing arguments, verify constructed terms and decide on polynomial formula in describing input numeric data. Π–theorem [2] is the basis of dimensional analysis technique. Its footprints can be found in many engineering domains. Three key steps of this technique are: 1) identifying all of the relevant physical terms and their units, 2) selecting base arguments from the identified terms, and 3) constructing dimensionless combinations (different products of exponents) of the terms.

The first few steps of COPPER system are purely dimensional analysis. All the primitives of the description space with units and rules for generating derived descriptors must be provided by the user. When the system finds out that there is no missing argument, it constructs physically meaningful terms and iterates through each phase to look for a simple functional formula which is a low-degree polynomial.

Since COPPER's emphasis is to discover physically meaningful formulas through dimensional analyses, it only works with formulas that are linear combination of products and ratios of the unknown arguments. Many physical laws are in this function class, especially in a good number of engineering applications. From this perspective and the Weierstrass approximation theorem [17], COPPER's performance surpass the performance of BACON and ABACUS. COPPER has demonstrated an important way to use deep domain knowledge in a function form discovery system.

2.6 Summary

All the discovery systems we have reviewed aim to find quantitative relationships between numerical terms. Though many techniques were used to enhance the ability of the discovery systems, including 1) preprocessing data by data transformations, 2) introducing domain knowledge to speed up the search, and 3) utilizing statistic tools to deal with noisy data, all of the early systems share a common fatal shortcoming — highly limited scope of discoverable function forms. As indicated in the preceding review, the systems in BACON family make their discovery within polynomial and rational functions, and data analysis approaches make their discovery with a set of arbitrarily selected prototypes.

The data transformation approach synthesizes a wide range of function forms by combining two fundamental techniques, data transformations and functional pattern recognitions, into one system. The data transformation approach method can be viewed as a general quantitative law discovery model. It can adopt other methodologies in an "cumulative way" to enhance the performance of a new system. However, the FFD system is only a first attempt of this approach and leaves still many unsolved problems and opportunities for improvements. LINUS, as the first successor of FFD, has contributed with two major improvements — releasing the monotonicity constraint by range splitting, and formula refining through parameter calibration. This research aims to make progress in a different direction, i.e. discovering three-variable function forms using the data transformation approach.

Chapter 3

Function Form Discovery by Data Transformation

Generally speaking, the function form discovery problem is the following: design a procedure that can select a formula $f(y, \mathbf{x}) = 0$, $f \in \mathcal{F}$, $\mathcal{F}$ is a set of formulas called 'available formula set', and the selection should optimally match with a set of given observation instances $O = \{(y_i, \mathbf{x}_i)\}$, called 'observation data set', in terms of justification, parsimony and transparency. The available formula set $\mathcal{F}$ could be a limited set of function form protocols (such as the functions listed in Table 2.3 for the E^* algorithm), a class of analytic functions (such as rational functions), or an implicit set defined in a recursive way (like in FFD). It is also called the function form coverage of a discovery system. Typically, $\mathrm{x} \in R \subseteq \Re^n$ and $y \in D \subseteq \Re$, where $\Re$ is the set of real numbers, n is an integer, R is called the domain of the function form, D the range of the function, and $R \times D$ the observation domain. This problem is also called the *real-valued function form discovery problem*, *numeric function form discovery problem* or the *quantitative reasoning problem*.

If $\mathcal{F}$ contains only mathematic formulas related to continuous functions, it is then called continuous function form discovery problem. Throughout this thesis, we are concerned only the problem of continuous function form discovery. Therefore the term "function" will be used to denote real-valued continuous function forms, unless explicitly noted otherwise.

Furthermore, we restrict our attention to three-variable problems, i.e. $\mathbf{x} \in R \subseteq \Re^2$.

In this chapter, I will discuss the foundations of the machine discovery system FFD-II. The discussion is divided into three parts. In the first part, I will describe the basics of the proposed methodology by showing that data transformation model is a general function form discovery model. A formal statement of the research problem will then be presented in the second part. In the last part I will first introduce the function form description language used by FFD-II, and then focus on theoretical issues concerning the description language, such as expressiveness, necessity, sufficiency, and redundancy.

3.1 General Model of Function Form Discovery

3.1.1 Central Mechanism of Data Transformation

As stated in the review, function form discovery systems fall in three categories: numeric analysis, data transformation and formula construction. Numeric analysis methods emphasize the recognition of functional patterns directly from the observations. Formula construction methods, in contrast, try to simplify the original observations into a very simple functional form, such as a constant or a linear function. The data transformation method is a combination of these two methods.

The data transformation model employs both a rich set of tools for data simplification, known as the data transformation set, and a set of numeric tools that can recognize functional patterns in a set of functions known as the primitive set. Thus, it can be viewed as a general model of function form discovery methods.

However, the data transformation approach is not simply one that combines the two different approaches together. For handling the task of simplifying a wide variety of functional patterns, the data transformation set must be carefully composed. The primitive set must be able to represent, in general, as many as possible of the simplified functional patterns efficiently. Furthermore, the system must be able to use appropriate functional pattern simplification tools to simplify a given observation data set efficiently into a recognizable primitive. Figure 3.1 depicts the central mechanism of function form discovery by data transformation. The key idea of this approach is recursively simplifying the functional image through data transformations until a simple recognizable functional pattern is reached[1].

3.1.2 Numeric Analysis in General

The data transformation model is a general function form discovery model. The numeric analysis method could be viewed as one extreme while the formula construction method is its opposite. For a better understanding of the data transformation method, let us first examine one extreme, the numeric analysis method, from the general point of view. The other extreme will be discussed in the next section.

As an extreme, numeric analysis approaches are composed of an empty set of data transformations and a relatively large set of primitive functional patterns, namely the function prototypes. The system does not search for an operation that can simplify the functional pattern. Instead, the discovery is solely the identification of one matching prototype function which best describes the observation data. From this perspective, traditional numeric methods seem to qualify as functional form discovery methods. To avoid this confusion, the following may be considered as the criteria which distinguish the machine discovery methods from pure numeric methods.

- Measured by the system's ability to discover, a discovery system should cover a larger variety of different function forms. A method which can only find function form representations within a very few possibilities is disqualified as a discovery method. However, the qualifiers of "many" and "few" are only relative. Nevertheless, any system that simply performs numeric approximation or interpolation using a few selected formulas is not a machine discovery system.

[1] A functional image is a set of observation data that numerically represents a function form. A functional image is said to be "simple" if through a few application of data transformations it could be transformed into a functional image that could be fitted to one of the selected simple functions, primitives, defined in the system. Formal definitions will be given later on in this thesis.

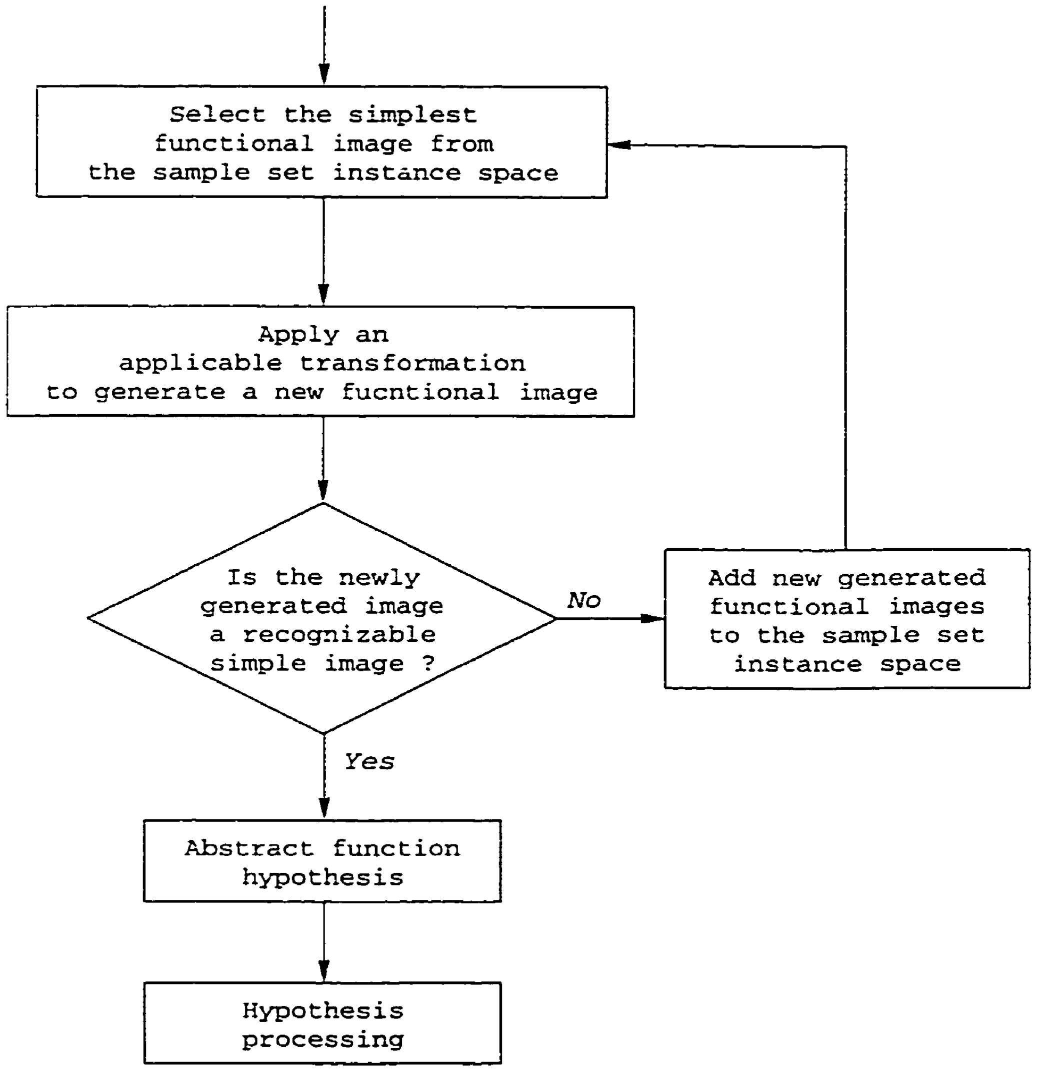

Select the simplest functional image from the sample set instance space
Apply an applicable transformation to generate a new fucntional image
Is the newly generated image a recognizable simple image ?
No
Add new generated functional images to the sample set instance space
Yes
Abstract function hypothesis
Hypothesis processing

Figure 3.1: The Central Mechanism of Data Transformation Approach

- Measured by the capability of the methodology itself, a discovery system usually utilizes not only traditional numeric tools but also some other techniques to enhance its discovery ability. For example, cluster analysis was employed by the KEDS system and dimensional analysis were used by some other systems.

- Measured by the quality of the solution, a discovery system should give a simpler and more interpretable solution instead of solely an accurate solution. Traditional numeric analysis techniques solve function form discovery problems considering only the justification criteria. The possible parameters of the functional form templates are decided by minimizing a numerical measurement of error. The simplicity and meaningfulness of the solution rely on the human expert who tries to solve the problem using certain numeric tools. In contrast, a discovery system that use numeric analysis approach should have the ability to take over the task of human experts to a certain level in choosing the simpler and meaningful expression to describe the given observation data set. However, due to the limited set of prototype function forms a system can handle, the ability of any system, that takes the numeric analysis approach to find a simple and meaningful function form is also limited.

- An intelligent approach should be able to generate a solution along with a set of meaningful measurements. An example is E^* algorithm, which measures the quality of its discovery by the measurements of *significance*, *distinction* and *systematic lack of fit*.

Since the system does not need any inference ability to choose operations from rule space, numeric analysis approaches require the least amount of inference. This is one of the reasons why this approach cannot go too far from traditional numeric methods in performing data modeling.

3.1.3 Formula Construction in General

The formula construction model can also be viewed as a data transformation methodology. Any formula construction approach must be able to decide when a discovery is accomplished.

This activity involves the detection of some very simple functional patterns. These patterns correspond to the primitives in the data transformation model. The system must also be able to incrementally construct a formula, (called a theoretical term as in the BACON system), utilizing inference rules and elementary formulas. These elementary formulas along with the way through which a formula is constructed are equivalent to the data transformation set and sequence in a data transformation approach. Function form discovery systems that take the formula construction approach are composed of two fundamental parts, as with the data transformation approach. Therefore, the two types of systems have the same fundamental structure.

Consider the BACON system. It is a formula construction based system which is one of the most well-known function form discovery systems. Its formula construction rules (Heuristics 3 and 4 on page 17) can be rewritten as two algebraic transformations:

$$\begin{aligned} T1: (X, Y) &\longmapsto (X, X/Y) \\ T2: (X, Y) &\longmapsto (X, X \cdot Y), \end{aligned}$$

and its termination conditions (Heuristics 1 and 2 on page 16) correspond to the following two primitives:

$$\begin{aligned} P1: Y &= a \cdot X + b \\ P2: Y &= C. \end{aligned}$$

We have seen the rediscovery of Kepler's third law by BACON in Section 2.2.1. To illustrate the concept of function form discovery by data transformation, let us examine the same problem from the general point of view. The discovery can be made by first transforming (D, P) into $(D, D/P)$ by applying $T1$; then applying $T2$ to $(D, D/P)$ to generate $(D, D^2/P)$; and finally applying $T2$ to $(D, D^2/P)$ to generated $(D, D^3/P)$. After this transformation, the data match with the primitive function $P2$ because $D^3/P = 1$. The discovered function form can be represented by a transformation sequence along with the matching primitive $P2$ as

$$(T2 \circ T2 \circ T1, Y = 1)$$

where Y is a function of the original terms D and P decided by the transformations that have been applied. In this example, $Y = D^3/P$.

It is easy to distinguish a numeric analysis function form discovery system, which has an empty transformation set, from a data transformation function form discovery system. The differences between a formula construction approach and a data transformation approach are a little subtle. However, we classify them into different categories because they have different constructions, emphases and performances. The differences between the constructions of these two different approaches are:

Operations The construction rule set of a formula construction system is and domain dependent. BACON includes only two rules to deal with rationals and polynomials, and the Gerwin's algorithm includes six binary rules for combining six different analytic functions.In contrast, the transformation set employed by a data transformation system is usually more general and may include any one-to-one continuous mapping. In both cases, the transformations in the set should be well coordinated to enhance the performance of the system.

Heuristics The heuristics related to each operation in a formula construction system are usually more "precise". Each operation can only be applied when a specific pattern is detected between current related terms. This requires a thorough understanding on the effects of each operation applied to a certain data instance. Examples are BACON's heuristics on page 16 and the elementary patterns that the Gerwin's algorithm looks for to trigger a certain construction step on page 22. In contrast, in a data transformation system, the heuristics are usually more coarse and general since it is not practical to construct a large, general-purpose and well understood transformation set at the moment.

Primitives Unlike data transformation systems which in principle can include any functional pattern in its primitive functional pattern set, formula construction systems limit their primitive functional pattern set only to constant or linear functions for the reasons of the structure of their heuristics and operation set.

With different constructions, these two methodologies have different emphases and performances.

- Formula construction methods emphasize reliability. The reliability refers to both the tolerance to error and the consistency of the theoretical and practical coverage. Since

  1. the operation set is compact and only those transformations that are less sensitive to noise are chosen,
  2. the fundamental patterns the system choosing to handle are usually the simplest,
  3. statistical techniques can be easily included in the system,

  the system usually has good tolerance to both the input noise and the computational errors. Furthermore, considering also the fact that the properties of the employed transformations and primitives in a formula construction approach are usually simple and easy to analyze, we know exactly what function form could or could not be discovered by the system. For example, BACON can and can only find polynomial and rational function forms.

- Data transformation methods, in contrast, emphasize the coverage of a large variety of complex function forms. That is achieved by a well organized transformation set along with a primitive set. However, as a tradeoff of employing some powerful but noise sensitive transformations, the system is relatively vulnerable to noise. The employment of some advanced transformations, such as differential transformations, makes it a difficult task to describe the exact function form coverage of a discovery system that takes data transformation approach. Moreover, due to the great number of selectable function forms and the limited numeric computation accuracy with digital computing machine using selected numerical methods, systems that perform data transformation can not guarantee the discovery of all function forms that are claimed discoverable by theoretical analyses of the system.

Therefore, when we seek an application in simple domain, for example elementary physics and chemistry, formula construction method is a good choice for its reliability and efficiency.

3.1.4 Cumulative Enhancement

Besides the ability to cover a wide variety of function forms, the most important benefit for taking data transformation approach is the possibility and flexibility of constructing new high performance function form discovery systems in a cumulative way. The 'cumulative methodology' is the methodology to construct new systems with some other augmented simpler systems (old systems). The new system should perform better and carries out more tasks. Meanwhile, the new system not only includes the old ones but also assigns them new roles.

The data transformation approach is a general and flexible discovery method. All the existing methodologies can find their new roles in this approach. It has been mentioned that the discovery algorithms in the other two categories share the same limitation of small discoverable function form classes. This limitation prevents any previous discovery algorithm from being a general-purpose function form machine discovery system. However, due to their robustness and efficiency in dealing with certain simple function form discovery tasks, they can be used in a data transformation based discovery system to perform some simple functional pattern recognitions.

There are two major ways in which we can build a data transformation based function form discovery system using cumulative methodology. *First*, the primitive functional pattern set could be organized in a better way by employing simpler function form discovery systems. It means that the recognition of the primitives does not have to rely solely on traditional numerical tools. The recognition tasks can also be carried out by selected function form discovery systems. Since a function form discovery system usually provides us with a more compact and meaningful fit, the performance of the system could be largely enhanced in the way of organizing the primitive set with some well selected simple function form discovery systems.

Second, some transformations we may choose may include descriptive expressions. A system must be able to find these expressions for carrying out the discovery process or completing a discovery. For example, if a system includes partial derivatives as one transformation, a lower dimension discovery system is required to find the possible boundary conditions for inverting those data transformations so that a complete function form description can be obtained. The other example is dimension reduction transformation. When solving multi-variable function form discovery problems, reducing the dimension of the problem is an important way to simplify the problem. However, if we want to use this strategy, it is necessary to have an associated function form discovery system that can take over the discovery tasks with fewer dimensions. Such a system is usually simpler than the new built system of higher dimension. Some transformations may be observation instance related and must be constructed based on the recognition of some special functional pattern from the corresponding observation instance. This will require the discovery of function forms within a specific function form class. A simple function form discovery algorithm should be available to do this job. Factorization and dimension reduction transformations[2] are two examples of this type of transformations.

3.2 A Formal Statement of the Research Problem

We have just discussed the data transformation function form discovery model as a general function form discovery model. The numeric analysis and formula construction models are two of its special cases. A system based on the data transformation model must have both the ability to simplify functional patterns and the ability to recognize functions in certain primary function classes. When the system is give an observation data set, it recursively selects simplification tools (defined as the data transformations in the system's tool-box) to simplify the observation data set, and tries to match the simplified observation data set with one of the functions in a selected set of function templates. In the discovery system, the

[2]The factorization is a transformation employed by FFD-II, while the dimension reduction is carried out as a special type of primitive recognition — compositional primitive recognition in the FFD-II system. More details will be given when they are introduced.

simplification tools are known as the data transformation set, while the function templates are known as the primitive functional pattern set.

In this section, the research problem will be formally stated in the form of function form discovery by data transformation. Before the statement, a number of definitions will be given. I will start with the definitions of function and observation, then carry on with the two major components, the data transformation set and the primitive set, and finally give the definition of function form description language. Since the symbols introduced in the definitions will be used as a convention in the remaining part of this thesis, they will be summarized at the end of the definition part.

3.2.1 Definitions

Definition 1 *Let $f : D_1 \subset \Re^2 \longmapsto D_2 \subset \Re$ be an unknown real-value function governing the system under our study, where D_1 is called the* function domain, *D_2 is called the* function range *and $D_1 \times D_2$ is called the* observation domain. *An* observation *is a real-valued three-tuple (a, b, c) such that:*

$$c = f(a,\ b) \text{ and } (a,\ b,\ c) \in D_1 \times D_2 \tag{3.1}$$

An observation data set *is a set of observations*

$$S_O \overset{\text{def}}{=} \left\{ (u_i,\ v_i,\ w_i) \;\middle|\; \begin{array}{l} w_i = f(u_i,\ v_i); \\ (u_i,\ v_i,\ w_i) \in D_1 \times D_2; \\ i = 1, \cdots, m \end{array} \right\} \tag{3.2}$$

An observation data set is also called a functional image *or an* image *in short.*

An observation data set is a numeric representation (instantiation) of a function form.

Definition 2 *A* transformation *is a "one-to-one onto mapping" T defined on any non-empty subset $D_f \subset \Re^3$, where $\Re^3$ denotes three-dimensional Cartesian Space, such that:*

$$T: \ D_f \longmapsto D_t, \tag{3.3}$$

where $D_f \subseteq \Re^3$ is the transformation domain and $D_t \subseteq \Re^3$ is the transformation range. The inverse transformation *of a transformation T is thus the transformation*

$$T^{-1}: \ D_t \longmapsto D_f \tag{3.4}$$

such that $\forall(u, v, w) \in D_f, \ T^{-1}(T(u, v, w)) \equiv (u, v, w)$. A transformation class *$\boldsymbol{T}$ is a set of transformations that includes either a single transformation or a number of transformations described by a parameterized transformation. A transformation in a transformation class is an instantiation of the class. As such, we can express a transformation as an instance of the corresponding transformation as:*

$$\boldsymbol{T}\Big|_{parametric\ expression\ descriptions} \tag{3.5}$$

Without ambiguity, an instantiated transformation class can also be written as

$$T\Big|_{parametric\ expression\ descriptions}. \tag{3.6}$$

For example, the parameterized transformation

$$\boldsymbol{T}_F \stackrel{\text{def}}{=} \left\{ T_F \;\middle|\; \begin{array}{c} T_F : (u, v, w) \longmapsto (u, v, \frac{w}{a \cdot u + b \cdot v + c}) \\ \forall a, b, c \in \Re, \ \ a \cdot b \cdot c \neq 0 \end{array} \right\}$$

is a transformation class. One of its instantiations

$$\boldsymbol{T}_F\Big|_{u+v} : (u, v, w) \longmapsto \left(u, v, \frac{w}{u+v}\right)$$

is a transformation where $u + v$ is the parametric expression description of this particular instantiation.

Definition 3 *Let $T_1 : D_{f1} \mapsto D_{t1}$ and $T_2 : D_{f2} \mapsto D_{t2}$ be two transformations, and $D_i = D_{t1} \cap D_{f2}$ is not empty. The transformation* composition operator *$\circ$ defines a binary* *operation of two transformations, written as* $T_2 \circ T_1$ *, that yields a new transformation* T

$$T_2 \circ T_1 \overset{\text{def}}{=} T : D_f \longmapsto D_t \left|\ \begin{array}{l} \forall(x, y, z) \in D_f,\ T(x, y, z) \in D_t, \\ \text{and } T(x, y, z) = T_2(T_1(x, y, z)) \end{array}\right. \tag{3.7}$$

where $D_f = T_1^{-1}(D_i) \subseteq D_{f1}$ *and* $D_t = T_2(D_i) \subseteq D_{t2}$ *.*

Note that a transformation is a one-to-one onto mapping. As such, the inverse of a composed transformation $T = T_2 \circ T_1$ is the transformation

$$T^{-1} = (T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1} \tag{3.8}$$

An example of transformation composition is $T_F|_{u+v} \circ T_F|_{u-v}$:

$$T_F|_{u+v} \circ T_F|_{u-v} \ :\ (u, v, w) \longmapsto \left(u, v, \frac{w}{u^2 - v^2}\right)$$

Definition 4 *A* <u>transformation class set</u> *is a set of transformation classes*

$$\mathcal{S}_T \overset{\text{def}}{=} \left\{ \boldsymbol{T}_i \ \middle|\ \begin{array}{l} \boldsymbol{T}_i \text{ is a transformation class} \\ i = 1, \cdots, K \end{array} \right\} \tag{3.9}$$

where K *is an integer denoting the size of the transformation class set. The transformation set defined by Equation (3.10) is called the* <u>base transformation set</u> *corresponding to transformation class set* $\mathcal{S}_T$ *:*

$$\boldsymbol{S}_T \overset{\text{def}}{=} \boldsymbol{I} \cup \left(\bigcup_{i=1}^{K} \boldsymbol{T}_i \right) \tag{3.10}$$

where $\boldsymbol{I} = \{ I \}$ *and* I *is the* identity transformation

$$I \overset{\text{def}}{=} (u, v, w) \longmapsto (u, v, w), \quad \forall (u, v, w) \in \Re^3 \tag{3.11}$$

Note that each transformation class $\boldsymbol{T}_i$ is also a set of transformations. The bold calligraphic letter "$\mathcal{S}$" denotes a transformation class set and a simple bold capital letter "$\boldsymbol{S}$" denotes the union of transformation classes in a transformation class set $\mathcal{S}$.

Symbol	Name	Expression
$\boldsymbol{T}_F$	**Linear Factorization**	$\left\{ T_F \,\middle\vert\, \begin{array}{c} T_F : (u, v, w) \longmapsto (u, v, \frac{w}{a \cdot u + b \cdot v + c}) \\ \forall a, b, c \in \Re, \quad a \cdot b \cdot c \neq 0 \end{array} \right\}$
$\boldsymbol{T}_L$	Logarithm	$\{ T_L \mid T_L : (u,\ v,\ w) \longmapsto (\,u,\ v,\ \log(w)\,)\,\}$
$\boldsymbol{T}_D$	Differentiation	$\{ T_D \mid T_D : (u,\ v,\ w) \longmapsto (\,u,\ v,\ \partial w/\partial u\,)\,\}$

Table 3.1: An Example Transformation Class Set

As an example, Table 3.1 defines transformation classes for a transformation classes set. Among the transformation classes, the transformation class $\boldsymbol{T}_L$ contains a single non-parameterized transformation, and $\boldsymbol{T}_F$ contains many transformations expressed by a parameterized transformation with parameters a, b and c. The transformation class $\boldsymbol{T}_D$ is a little tricky. It does not include any parameter in its expression. However, it is a parameterized transformation class. Recalling the definition of transformation, any transformation must be a one-to-one mapping. For this reason, any differential transformation has hidden parameters, or more precisely, parametric expressions that contribute as the deterministic conditions required for inverting the transformation by an integral. Let $(u, v, w = h(u, v))$ be the underlying functional relation corresponding to the variable triple (u, v, w). If

$$\left(u, v, \hat{w} = \hat{h}(u, v)\right) = T_D\,(u, v, w = h(u, v)),$$

then

$$\hat{h}(u, v) = \partial w/\partial u = h'_u(u, v)$$

and

$$f(u, v) = \int \hat{h}(u, v)\mathrm{d}u + C(v)$$

where $C(v)$ is the integral constant. When we are given a function form corresponding to the triple $(u, v, \hat{w})$ as $\hat{w} = \hat{h}(u, v)$, and we know that $(u, v, \hat{w}) = T_D(u, v, w)$, it is necessary to specify the integral constant $C(v)$ for extracting the functional relationship between u, v and w, i.e. $w = h(u, v)$. However, the specification of the parametric expression can take many forms. For example, a simple way to specify the parametric expression is to express it with a pair of equations that determines the integral of the partial differential:

$$\begin{cases} u = f(v) \\ w = g(v) \end{cases} \tag{3.12}$$

where f and g are functions in class C^∞. It means, on the smooth curve $u = f(v)$ on a functional image with associated variables u, v and w, the value of w is related to the value of v as $w = g(v)$. Thus, the functional relationship between u, v and w will be:

$$w = \int_{f(v)}^{u} \hat{h}(u, v)\mathrm{d}u + g(v)$$

An instantiation of a parameterized transformation is a transformation. The specified parametric values or expressions are associated with the transformation as its subscript. For example, $T_F|_{u+v}$ stands for the factorization transformation

$$(u, v, w) \longmapsto (u, v, \frac{w}{u+v})$$

and $T_D|_{[u=0,\, w=v^2]}$ stands for the differential transformation with the indicated parametric expressions, i.e.

$$\begin{cases} T_D \;:\; (u, v, w) \longmapsto (u, v, \partial w/\partial u) \\ \quad \text{where } \; w = v^2, \; \text{ when } u = 0. \end{cases}$$

Definition 5 *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two transformation sets, the transformation set generated by $\boldsymbol{A}$ and $\boldsymbol{B}$ with respect to the composition operator $\circ$, or the* generated transformation set *in brief, is denoted by $\boldsymbol{A} \circ \boldsymbol{B}$:*

$$\boldsymbol{A} \circ \boldsymbol{B} \overset{\text{def}}{=\!=} \left\{ T \;\middle|\; \begin{array}{l} T \text{ is transformation, and} \\ \forall \bar{T}, \; \underline{\text{if}} \; \bar{T} \in \boldsymbol{A} \text{ or } \bar{T} \in \boldsymbol{B}, \\ \quad \underline{\text{then}} \; \bar{T} \in \boldsymbol{A} \circ \boldsymbol{B} \\ \forall T_1, T_2 \in \boldsymbol{A} \circ \boldsymbol{B}, \; \underline{\text{if}} \; T_1 \circ T_2 \text{ exists,} \\ \quad \underline{\text{then}} \; T_1 \circ T_2 \in \boldsymbol{A} \circ \boldsymbol{B} \end{array} \right\} \tag{3.13}$$

The generated transformation set of a single transformation set $\boldsymbol{A}$ *is denoted by* $\boldsymbol{A}^*$:

$$\boldsymbol{A}^* \overset{\text{def}}{=\!=} \boldsymbol{A} \circ \boldsymbol{A} \tag{3.14}$$

Similarly, the transformation set generated by a transformation class set $\boldsymbol{\mathcal{S}}_{\mathrm{T}} = \{\boldsymbol{T}_1, \cdots, \boldsymbol{T}_K\}$ *is defined as*[3]

$$\widehat{\boldsymbol{S}_{\mathrm{T}}} \overset{\text{def}}{=\!=} \boldsymbol{S}_{\mathrm{T}}^* = \boldsymbol{I} \circ \boldsymbol{T}_1 \circ \cdots \circ \boldsymbol{T}_K \tag{3.15}$$

Definition 6 *Let* $\boldsymbol{\mathcal{S}}_{\mathrm{T}} = \{\boldsymbol{T}_1, \cdots, \boldsymbol{T}_K\}$ *be a transformation class set,* $\boldsymbol{S}_{\mathrm{T}}$ *be the corresponding base transformation set, and* $\widehat{\boldsymbol{S}_{\mathrm{T}}} = \boldsymbol{S}_{\mathrm{T}}^*$ *be the corresponding generated transformation set. Any element of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$*, which defines a transformation or more precisely a composed transformation, is called a* <u>transformation sequence</u> *generated by* $\boldsymbol{\mathcal{S}}_{\mathrm{T}}$*, or in brief, a transformation sequence.*

Definition 7 *Let* $\boldsymbol{T}$ *be a transformation class, the* <u>rank</u> *of* $\boldsymbol{T}$ *is an arbitrarily selected non-negtive integer associated with* $\boldsymbol{T}$

$$\overline{Rank}\,(\boldsymbol{T}) = K \in \boldsymbol{N} \tag{3.16}$$

where $\boldsymbol{N}$ *is the set of natural numbers (i.e. non-negative integers). The rank of a transformation in a transformation class* $\boldsymbol{T}$ *equals to the rank of the transformation class*

$$\overline{Rank}\,(T) = \overline{Rank}\,(\boldsymbol{T}), \;\; \forall T \in \boldsymbol{T} \tag{3.17}$$

Let $\boldsymbol{\mathcal{S}}_{\mathrm{T}} = \{\boldsymbol{T}_1, \cdots, \boldsymbol{T}_K\}$ *be a transformation class set,* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ *be the corresponding generated transformation set, and* $I_i = \overline{Rank}\,(\boldsymbol{T}_i)$, $i = 1, \cdots, K$ *be the corresponding*

[3] Note that the composition operation "$\circ$" is associative but not commutative when it is applied to transformations (Definition 3). "$\circ$" is both associative and commutative when it is applied to transformation classes (Definition 5). Thus in Equation 3.15 which transformation class appears first does not affect the result.

A "Hatted Capital Letter" "$\widehat{\boldsymbol{S}_{\mathrm{T}}}$" is used to emphasize that the transformation set is generated by the transformation class set $\boldsymbol{\mathcal{S}}_{\mathrm{T}}$. A generated transformation set $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ is also denoted by $\boldsymbol{S}_{\mathrm{T}}^*$ or $\boldsymbol{\mathcal{S}}_{\mathrm{T}}^*$ throughout this thesis.

transformation classes rank values. Let $\bar{T} \in \widehat{\boldsymbol{S}_T}$, and $\bar{T} = \bar{T}_1 \circ \bar{T}_2 \circ \cdots \circ \bar{T}_O$, where each $\bar{T}_i$, $(i = 1, \cdots, O)$ is a transformation belong to one of the transformation classes in set $\boldsymbol{S}_T$, i.e. $\bar{T}_i \in \boldsymbol{T}_{K_i}$ and $K_i \in \{1, \cdots, K\}$. The rank of transformation sequence $\bar{T}$ is defined to be the sum of rank values of each individual transformation

$$\overline{Rank}\ (\bar{T}) \overset{\text{def}}{=\!=} \sum_{i=1}^{O} I_{K_i} \tag{3.18}$$

The order *of transformation sequence $\bar{T}$ is the number of transformations in the sequence:*

$$\overline{Order}\ (\bar{T}) \overset{\text{def}}{=\!=} O \tag{3.19}$$

Definition 8 *A* functional primitive class *is a set of functions (either a single function or parameterized functions) $\{z = f(x, y)\}$ in the class C^{∞} which could be either explicit or implicit. A functional primitive class is denoted by $\boldsymbol{F}$.*

Definition 9 *A* compositional primitive class *is a two-dimensional expression set $\{g(x, y)\}$ (either a single expression or a number of parameterized expressions), and each $g(x, y)$ is in the class C^{∞}. A compositional primitive class is denoted by $\boldsymbol{E}$.*

A functional primitive class is a function template which stands for a set of functions distinguished by different parameter settings (if any). As such, we will use the bold capital letter "$\boldsymbol{F}$" to denote a functional primitive class and use the italicized capital letter "F" to denote an instant function in the set "$\boldsymbol{F}$". Similarly, a compositional primitive class is an expression template which stands for a set of expressions distinguished by different parameter settings (if any). So that "$\boldsymbol{E}$" represents a compositional primitive class and "E" represents an instant expression in the set $\boldsymbol{E}$.

Definition 10 *An element F in a functional primitive class $\boldsymbol{F}$ is called a* functional primitive, *written as $F \in \boldsymbol{F}$. An element E in a composition primitive class $\boldsymbol{E}$ is called a* compositional primitive, *written as $E \in \boldsymbol{E}$.*

Definition 11 *A functional primitive class or a compositional primitive class is called a* primitive class, *and denoted by* $\boldsymbol{P}$. *A* primitive class set *is composed of primitive classes.*

$$\boldsymbol{S}_{\mathrm{P}} \overset{\mathrm{def}}{=\!=} \left\{ \boldsymbol{F}_i, \boldsymbol{E}_j \;\middle|\; \begin{array}{c} \boldsymbol{F}_i \text{ is functional primitive class,} \\ \boldsymbol{E}_j \text{ is compositional primitive class,} \\ i = 1, \cdots, m;\ j = 1, \cdots, n \end{array} \right\} \tag{3.20}$$

where the integer 'm' is the number of functional primitives and the integer 'n' is the number of compositional primitives. Corresponding to the primitive class set $\boldsymbol{S}_P$, a primitive set *is defined as*

$$S_{\mathrm{P}} \overset{\mathrm{def}}{=\!=} \left(\bigcup_{i=1}^{m} F_i \right) \cup \left(\bigcup_{i=1}^{n} E_i \right) \tag{3.21}$$

An element P in the set S_{P} is either a functional primitive F or a compositional primitive E. It is called a primitive.

Table 3.2 lists two examples of primitive classes. The compositional primitive class $\boldsymbol{E}_L$ contains a parameterized expression with parameter θ and the functional primitive class $\boldsymbol{F}_L$ contains a parameterized function with parameters a, b and c. An instantiation of $\boldsymbol{E}_L$, such as the expression $\frac{3}{5}u + \frac{4}{5}v$, is a compositional primitive expression. An instantiation of $\boldsymbol{F}_L$, such as $w = u + v$, is a functional primitive expression.

Symbol	Name	Expression
$\boldsymbol{E}_L$	Linear Compositional Primitive	$\left\{ E_L \;\middle|\; \begin{array}{c} u\cos(\theta) + v\sin(\theta), \\ \forall \theta \in [0, \pi) \end{array} \right\}$
$\boldsymbol{F}_L$	Linear Functional Primitive	$\left\{ F_L \;\middle|\; \begin{array}{r} w - (a \cdot u + b \cdot v + c) = 0, \\ \forall a, b, c \in \Re \end{array} \right\}$

Table 3.2: Examples of Primitives

Definition 12 *A* <u>function form description language</u> $\mathcal{L}$ *has two components — a transformation class set and a primitive class set, written as*

$$\mathcal{L} \stackrel{\text{def}}{=} (\mathcal{S}_{\mathrm{T}}, \mathcal{S}_{\mathrm{P}}) \tag{3.22}$$

where

$$\mathcal{S}_{\mathrm{T}} = \{\boldsymbol{T}_1, \cdots, \boldsymbol{T}_K\} \tag{3.23}$$

is the transformation class set, and

$$\mathcal{S}_{\mathrm{P}} = \{\boldsymbol{F}_1, \cdots, \boldsymbol{F}_m\} \cup \{\boldsymbol{E}_1, \cdots, \boldsymbol{E}_n\} \tag{3.24}$$

is the primitive class set. A function form description language is also called a <u>language</u> *in brief.*

Definition 13 *Let* $\mathcal{L}$ *be a function form description language described by equations* (3.22), *(3.23), and* (3.24). *Let* $\widehat{S_{\mathrm{T}}} = \mathcal{S}_{\mathrm{T}}^{*}$ *be the transformation set generated*[4] *by* $\mathcal{S}_{\mathrm{T}}$, *and* S_{P} *be the primitive set corresponding to* $\mathcal{S}_{\mathrm{P}}$. *A* <u>function form description</u> *in language* $\mathcal{L}$ *is defined as*

$$\begin{gathered}\mathcal{D}_{\mathcal{L}} \stackrel{\text{def}}{=} (D_T, D_P) \\ \text{for any } D_T \in \widehat{S_{\mathrm{T}}}, \text{ and } D_P \in S_{\mathrm{P}}\end{gathered} \tag{3.25}$$

where, D_T *and* D_P *are called the* <u>description transformation sequence</u> *and the* <u>description primitive</u> *respectively.* D_T *can also be written as*

$$D_T = T_k \circ T_{k-1} \circ \cdots \circ T_1 \tag{3.26}$$

where each $T_i \in S_T$, $(i = 1, \cdots, k)$, *is a base transformation. The* <u>rank</u> *of the description equals to the rank of the description transformation sequence:*

$$\begin{aligned}\overline{Rank}\,(\mathcal{D}_{\mathcal{L}}) &= \overline{Rank}\,(D_T, D_P) \\ &\stackrel{\text{def}}{=} \overline{Rank}\,(D_T)\end{aligned} \tag{3.27}$$

[4] Refer to Definition 5.

Definition 14 *If a three-variable function form $z = f(x, y)$ (or $f(x, y, z) = 0$) can be expressed by a function form description in a language $\mathcal{L}$, it is* describable *in $\mathcal{L}$.*

A function form description language $\mathcal{L}$ could also be viewed as a set of functions that are describable by the language. As such, if a function form $f(x, y, z) = 0$ or $z = f(x, y)$ is describable in $\mathcal{L}$, it is denoted by

$$f \in \mathcal{L}.$$

Definition 15 *Let $\mathcal{L}1$ and $\mathcal{L}2$ be two function form description languages. If*

$$\forall f \in \mathcal{L}_1 \Longrightarrow f \in \mathcal{L}_2$$

it is said that $\mathcal{L}_2$ is a super-language *to $\mathcal{L}_1$, or $\mathcal{L}_1$ is a* sub-language *to $\mathcal{L}_2$. It is denoted by*

$$\mathcal{L}1 \sqsubseteq \mathcal{L}2 \quad \text{or} \tag{3.28}$$

$$\mathcal{L}2 \sqsupseteq \mathcal{L}1 \tag{3.29}$$

If $\mathcal{L}1 \sqsubseteq \mathcal{L}2$ and $\mathcal{L}2 \sqsubseteq \mathcal{L}1$, it is said that $\mathcal{L}1$ is equivalent *to $\mathcal{L}2$, and denoted by $\mathcal{L}1 \equiv \mathcal{L}2$.*

Definition 16 *Let $f(x, y, z) = 0$ or $z = f(x, y)$ be a function form describable in a function form description language $\mathcal{L} = (\mathcal{S}_T, \mathcal{S}_P)$. The* rank *of function form f in language $\mathcal{L}$ is defined as*

$$\overline{Rank}\ (f, \mathcal{L}) = \min \left\{ \overline{Rank}\ (\mathcal{D}_{\mathcal{L}}) \;\middle|\; \begin{array}{l} \forall \mathcal{D}_{\mathcal{L}} \in \mathcal{L}, \text{ and} \\ \mathcal{D}_{\mathcal{L}} \text{ is a descrip-} \\ \text{tion of } f \end{array} \right\} \tag{3.30}$$

The *Complexity* of a function form f in language $\mathcal{L}$ is measured by its rank in the language and the complexity of the corresponding description primitive[5]. When we construct the

[5] To the need of this research, the complexity of a primitive is the number of non-zero control parameters. In a more general measurement we should consider how difficult it is to find the expression of the primitive and hence the function form.

search heuristics, we take into account both of the rank of the transformation sequence that has been applied, and how likely the instance functional image can be transformed into a primitive with the application of a new transformation.

An Example Function Form Description Language:

To understand the definitions, let us examine an example function form description language and how function forms are expressed in the language. However, the rank values will not be shown in this example. They will be more meaningful to be presented in the part where the proposed function form description language is introduced.

The example function form description language is based on the transformation classes listed in Table 3.1 and the primitive classes listed in Table 3.2. Let us first list the components and the relevant sets as the following,

The Transformation Class Set

$$\mathcal{S}_{\mathrm{T}} = \{\, \boldsymbol{T_F} \,,\, \boldsymbol{T_L} \,,\, \boldsymbol{T_D} \,\} \tag{3.31}$$

The Base Transformation Set

$$\boldsymbol{S}_{\mathrm{T}} = \boldsymbol{I} \bigcup \boldsymbol{T_F} \bigcup \boldsymbol{T_L} \bigcup \boldsymbol{T_D} \tag{3.32}$$

The Generated Transformation Set

$$\widehat{\boldsymbol{S}_{\mathrm{T}}} = \boldsymbol{S}_{\mathrm{T}}^{\star} = \boldsymbol{I} \circ \boldsymbol{T_F} \circ \boldsymbol{T_L} \circ \boldsymbol{T_D} \tag{3.33}$$

The Primitive Class Set

$$\mathcal{S}_{\mathrm{P}} = \{\, \boldsymbol{E_L} \,,\, \boldsymbol{F_L} \,\} \tag{3.34}$$

The Primitive Set

$$\boldsymbol{S}_{\mathrm{P}} = \boldsymbol{E_L} \bigcup \boldsymbol{F_L} \tag{3.35}$$

The example function form description language can then be defined as:

The Description Language

$$\mathcal{L} = (\mathcal{S}_T, \mathcal{S}_P) \tag{3.36}$$

$$= (\{\boldsymbol{T_F}, \boldsymbol{T_L}, \boldsymbol{T_D}\}, \{\boldsymbol{E_L}, \boldsymbol{F_L}\}) \tag{3.37}$$

Using this example function form description language, the function

$$z = e^{(x^4 - y^4)} \tag{3.38}$$

has the following two possible descriptions[6]:

Function Form Description-1

$$D_{\mathcal{L}}^{(1)} = (T_D|_{[u=0,\, w=v^2]} \circ T_F|_{u+v} \circ T_F|_{u-v} \circ T_L\,,\; w = 2u)$$

Function Form Description-2

$$D_{\mathcal{L}}^{(2)} = (T_D|_{[u=0,\, w=v^2]} \circ T_F|_{u+v} \circ T_F|_{u-v} \circ T_L\,,\; u)$$

Thus, the function of Equation 3.38 is an instance of the language $\mathcal{L}$, written as

$$\underline{z = e^{(x^4 - y^4)}} \in \mathcal{L}$$

To transfer a function form description into a simple function, we need to invert the transformation sequence in the inverse order. Let us see the process of inverting "*Description-1*" first.

Starting from the primitive function D_P:

$$w = 2u,$$

where u, v and w are the variables that are generated by applying the transformation sequence D_T:

$$T_D|_{[u=0,\, w=v^2]} \circ T_F|_{u+v} \circ T_F|_{u-v} \circ T_L$$

[6] Note that the primitives of the two descriptions are different. More detail will be given soon.

to the original variable set (x, y, z), we invert the last transformation being applied, which is the differential transformation $T_D|_{[u=0,\, w=v^2]}$, through integration:

$$(u, v, w) \longmapsto (u, v, \int_0^u 2udu + v^2) = (u, v, u^2 + v^2),$$

where the integral function is the primitive function, and the initial condition of $w = v^2$ when $u = 0$ (specified by the subscript of T_D) determines the integral bounds and constant, then invert $T_F|_{(u+v)}$:

$$(u, v, u^2 + v^2) \longmapsto (u, v, (u + v)(u^2 + v^2)),$$

invert $T_F|_{(u-v)}$:

$$(u, v, (u + v)(u^2 + v^2)) \longmapsto (u, v, (u + v)(u - v)(u^2 + v^2)) = (u, v, u^4 - v^4),$$

invert T_L:

$$(u, v, u^4 - v^4) \longmapsto (u, v, e^{u^4 - v^4}).$$

Having inverted the transformation sequence, the variable triple (u, v, w) corresponding to the primitive has been turned into $(u, v, e^{u^4-v^4})$. Since (u, v, w) is a transformed variable set that is obtained by applying the transformation sequence D_T to a variable set (x, y, z), we should substituting the variable set (x, y, z) into the final obtained expression. That yields the function:

$$z = e^{x^4 - y^4}.$$

This completes the inverting process with a function identical to Equation 3.38

To rewrite function form "*Description-2*" to a simple function, we must first figure out a one dimension function $f(t)$ that relates the compositional primitive $\underline{u}$ to the independent variable w. In general, a compositional primitive expression $C(u, v)$ implies that the primitive function form can be expressed by a function f:

$$w = f(C(u, v)),$$

which is indeed a parametric expression. Thus, to complete a function form description that includes a compositional primitive, a one-dimensional function, namely the "descriptive expression",

$$w = f(t) \tag{3.39}$$

is required to couple with the compositional primitive. Otherwise, the function form description is not an unique description, which means it could stands for many function forms. In a function form discovery task, $f(t)$ is obtained by carrying out two-variable function form discovery upon a set of two-dimensional observation data. In this example, since the function to be expressed by the example language is known, we know the descriptive expression is $f(t) = 2t$. Hence we can start from the variable set $(u, v, w = 2u)$ and invert the transformation sequence, which is exactly the same as it appears in "*Description-1*", in the same way as we did in the transfer of "*Description-1*" into an explicit function.

Equation set (3.12), which is required to couple a differential transformation, and equation (3.39) are two types of parametric expressions of the function form description language presented here. They are all single-variable functions. Since the major concern of this research is the function form discovery problems in three-dimensional space, we treat them only as hidden parametric expressions that could be passed to an available two-variable function form discovery system that handles the tasks of discovering the necessary expressions[7].

Definition 17 *A* descriptive expression *of a function form description is a two-variable continuous function which is required for inverting a certain transformation or completing a function form description that contains compositional primitive as a component. A* descriptive image *is a set of real number pairs which is the numeric representation of a two-variable function in the class C^∞.*

We have seen how the transformation set and the primitive set in a function form description language work together to represent a function. A data transformation based function form discovery system is constructed on the bases of a defined language. If a system is constructed based on the description language $\mathcal{L}$ (Equation 3.37 and 3.37), it

[7] In the implementation, the FFD system and least-squares polynomial fitting method are considered as two choices

will discover the function form description "*Description-1*" or "*Description-2*" from a set of give observation data, such as

$$\left\{ (x_{ij}, y_{ij}, z_{ij}) \;\middle|\; \begin{array}{l} x_{ij} = -1 + 0.1i\,,\ y_{ij} = -1 + 0.1j\,,\ z_{ij} = e^{x_{ij}^4 - y_{ij}^4} \\ \quad \text{for} \quad i = 0, \cdots, 20;\ j = 0, \cdots, 20 \end{array} \right\}.$$

The example we have examined is an explicit function. Both of the functional descriptions we have given are invertible and the inversion results in an explicit function, which is a composition of elementary analytic functions. We shall mention here that things may not turn out to be that nice. To extract the functional expression regarding to the initial variables by inverting the discovered function form description, may result in an implicit function, an expression with integral operation or a set of equations. In other words, there are some functional descriptions that can only be inverted numerically. When the transformation set includes the differential operation or the functional inverse[8], this phenomenon is not avoidable. For example, if we add one more transformation class

$$\boldsymbol{T}_R : \ (u, v, w) \longmapsto (u, v, 1/w)$$

to the transformation class set $\boldsymbol{\mathcal{S}}_T$(Equation (3.33)) and one more functional primitive class

$$\boldsymbol{F}_S : \ w^2 = (1 - u^2)(1 - u^2 v^2)$$

to the primitive class set $\boldsymbol{\mathcal{S}}_P$(equation (3.35)), a new function form description language $\boldsymbol{\mathcal{L}}_n$ is defined:

$$\boldsymbol{\mathcal{L}}_n \ = \ (\{\boldsymbol{T}_F, \boldsymbol{T}_L, \boldsymbol{T}_D, \boldsymbol{T}_R\}, \{\boldsymbol{E}_L, \boldsymbol{F}_L, \boldsymbol{F}_S\}) \qquad (3.40)$$

Using this new description language, we can express the elliptic integral

$$z = f(x, y) = \int_0^x \frac{1}{(1 - t^2)(1 - y^2 t^2)} dt$$

by

$$(T_R \circ T_D|_{[u=0,\, w=0]}\,,\ F_S).$$

[8] Functional inverse is one of the transformation classes employed by FFD-II.

Elliptic integrals are special functions that can only be expressed in integral format. This example shows not only an example of the description of a complex function form but also the powerfulness of the data transformation method.

Different function form representations have different syntactic simplicities and semantic meaningfulness. The data transformation based model has the capability to handle complex function forms in various formats. Moreover, from a given observation data set, such a system can provide us with multiple solutions.

3.2.2 The Statement

The problem addressed in this thesis can now be stated as the follows.

Three-variable Function Form Discovery Problem:

Given :

1. an *Observation Data Set* $\mathcal{O}_f$ which is governed by an unknown explicit or implicit three-variable underlying function form $z = f(x, y)$.
2. a *Function Form Description Language* $\mathcal{L} = (\mathcal{S}_T, \mathcal{S}_P)$, where
   - the *Primitive Class Set* $\mathcal{S}_P$ contains :
   $$\mathcal{S}_P \overset{\text{def}}{=\!=} \left\{ \; \boldsymbol{F}_i \,,\; \boldsymbol{E}_j \;\middle|\; \begin{array}{l} i = 1, \cdots, m \\ j = 1, \cdots, n \end{array} \right\}$$
   - the *Ranked Transformation Class Set* $\mathcal{S}_T$ is:
   $$\left\{ \begin{array}{ll} \mathcal{S}_T = \{ \boldsymbol{T}_1, \boldsymbol{T}_2, \cdots, \boldsymbol{T}_k \} & \\ \overline{Rank}\,(\boldsymbol{T}_i) = \Gamma_i, & i = 1, \cdots, k \end{array} \right.$$
3. a *Maximum Rank* $\mathcal{R}_{\max}$.
4. a *Matching Error Tolerance Level* $\delta_{\max}$.

Construct : a *Function Form Description* $\mathcal{D}_{\mathcal{L}} = (D_T, D_P) \in \mathcal{L}$, such that:

1. $\overline{Rank}\ (D_T) \leq \mathcal{R}_{\max}$.
2. the averaged deviation between the function form $z = \hat{f}(x, y)$ represented by $D_{\mathcal{L}}$ and the underlying function form $z = f(x, y)$ represented by the observation data set $\mathcal{O}_f$ is less than $\delta_{\max}$.

Simplification Assumptions

At this stage, we restrict our scope with the following assumptions. Without explicit mention, these assumptions will exist throughout this thesis.

Continuity The unknown function form and all the underlying function forms of transformed images belong to the class C^{∞} in the observation domain.

Sufficient Observation We can acquire sufficient fine step observation data.

Known Expected Error The process that generates the observation data set are well determined, i.e. the expected error level is known.

Acquirable Descriptive Expression Any descriptive expression[9] required for completing a function form description could be obtained from an existing two-dimensional function form discovery system via passing an descriptive image to that system.

The first assumption implies that we need only to consider continuous transformations in the construction of $\mathcal{S}_T$ and $\mathcal{S}_P$, and the applicability of basic differential transformations. The second assumption is about the availability of the observation data. The third assumption ensures that the error propagation during the search can be estimated.

The last assumption allows this research to focus on the three-variable function form discovery problems. The discovery of two-variable function forms relies on an available

[9] Refer to Definition 17, page 56.

low dimension discovery system. Recall that we have been confronted with two types of descriptive expressions in an example (Refer to Equation 3.12 and 3.39). Since the FFD-II system contains the same type of transformations and compositional primitive, these two types of descriptive expressions are required to represent a complete functional format relationship[10] between variables. It is assumed that there is a supporting system that can find those expressions[11]. However, without the descriptive expressions, the constructed transformation sequence, if viewed only as a sequence of transformation classes along with the matching compositional primitive still reveals the underlying regularities of the observed functional image, and can be interpreted as a numeric law concerning the corresponding investigated real world problem. That means, the system can serve as a special mathematic modeling tool.

3.3 FFD-II Function Form Description Language

As has been pointed out, the performance of a data transformation based function form discovery system highly relies on the description language itself. The ability to simplify functional patterns is determined by the transformation class set, while the ability to recognize primitive patterns is determined by the primitive class set. The combination of these two abilities enables the system to discover function forms from numeric observation data. Since no specific application domain is specified, the focus is only placed on those general-purpose language components and related issues.

3.3.1 The Transformation Class

There is no doubt that a well tailored transformation class set ensures a wide function form coverage and better computational efficiency. When we are confronted with the task of

[10] A functional format relationship is represented with an analytic function, either implicit or explicit, while a numeric law can be generally expressed by any mathematic formula, such as an analytic function or a differential equation.

[11] The FFD system or simply a polynomial interpolation algorithm can be used as the supporting system.

constructing a transformation class set, we may ask at the very beginning: "What are the criteria for choosing transformation classes to meet our needs?". Unfortunately, we know very little to the answer of this question up to now. This issue is still open and calling for more attention in this research field.

Generally speaking, it is relatively easy to tell what transformations are necessary for covering a certain function form class. But it is difficult to tell what is a sufficient transformation class set to solve the function form discovery problems drawn from a specific population. In other words, it is relatively easy to characterize a single transformation, but hard to know how the different transformations in a defined transformation class set affect and enhance the performance of each other. Especially, when differential transformations are included, the analysis becomes very complicated.

Our purpose is to design a system that can deal with "common" function forms in scientific fields. These common function forms are generated by the combinations of fundamental analytic functions through fundamental function construction operations (listed in Table 3.3 and Table 3.4). Generally speaking, among these fundamental functions, the constant function class is the simplest (from the perspective of being easy to identify, compute and manipulate), the power function class is the second simplest, and the rest are about the same. Among the operators, the linear operations " $+$ " and " $-$ " are the simplest, " $\times$ " is next, and " $\div$ " may be more difficult to handle than " $\times$ ". Functional composition provides the most function form variations and is usually the hardest to handle.

Most of the analytic functions we can find in a first year calculus text book fall into the class we have just described. As a simple example, all polynomials are derived by repeatedly combining a constant function and a power function with the binary operators addition and multiplication. The function form $f(x)^{g(x)}$ can be rewritten as $e^{\log(f(x))\cdot g(x)}$. As such, it is a functional composition of the exponential function and the function generated by the product of two functions: the function g and the functional composition of log function and the function f.

With this guideline of what function forms we are going to deal with, we will consider those general transformations that simplify either the fundamental functions or the combi-

Expressions	**Function Name**
c	Constant functions
$(\cdots)^n$	Power functions
$(\cdots)^{1/n}$	Root functions
$\sin(\cdots)$, $\cos(\cdots)$, $\tan(\cdots)$, $\cot(\cdots)$, $\cdots$	Trigonometric functions
$\exp(\cdots)$	Exponential functions
$\log(\cdots)$	Logarithm functions

Table 3.3: Fundamental Analytic Function Forms

Symbol	Meaning
$+$	Addition operator
$-$	Subtraction operator
$\times$	Multiplication operator
$\div$	Division operator
$\odot$	Functional composition operator

Table 3.4: Fundamental Function Construction Operations

nation operators under certain circumstances. In the rest of this chapter, I use the following symbols to define the transformations. The variable set includes:

- u_1, u_2 and u_d denote the variable set of the current state (i.e. before transformation). Among them, u_1, u_2 are independent while u_d is dependent.
- v_1, v_2 and v_d are used to denote the variable set of a transformed state. Among them, v_1, v_2 are independent while v_d is dependent.

Let T be a transformation applicable to the variable set (u_1, u_2, u_d), and (v_1, v_2, v_d)

be the variable set that (u_1, u_2, u_d) is transformed into, i.e.

$$(u_1, u_2, u_d) \xmapsto{T} (v_1, v_2, v_d),$$

a transformation will be formulated as:

$$T : \begin{cases} v_1 = f(u_1, u_2, u_d) \\ v_2 = g(u_1, u_2, u_d) \\ v_d = h(u_1, u_2, u_d) \end{cases} \tag{3.41}$$

where f, g, and h stand for the expressions that specify the relationship between the original variable set and the new variable set. Before we define the transformations it should be mentioned that the expressions f, g, and h may contain certain parameters, thus the above formula will also pertain to a transformation class $\boldsymbol{T} = \{T\}$.

Now let us define the basic data transformations for FFD-II. When a transformation is defined, a brief description of the usefulness of the transformation in dealing with the general function forms will be given. I will also give the applicability conditions and the inverse transformation of each data transformation.

Logarithm

$$T_{\text{LOG}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = \log|u_d| \end{cases} \tag{3.42}$$

The importance of this transformation is that

1. it is a fundamental function we intend to deal with (Table 3.3);
2. it transfers power and root functions into linear combinations;
3. it transfers multiplication into addition;

4. it removes the functional composition (operation "$\odot$") from the exponential expression $\exp(f(u, v))$.

To preserve the continuity, logarithm can only be applied to a functional image with constant sign. The inversion of this transformation is

$$T_{\mathrm{LOG}}^{-1} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = \pm\exp(u_d) \end{cases} \tag{3.43}$$

where the sign "$\pm$"is decided by functional image to which the logarithm transformation was applied to obtain the image $\mathcal{O}_{(u_1, u_2, u_d)}$.

Reciprocal

$$T_{\mathrm{REC}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = 1/u_d \end{cases} \tag{3.44}$$

Reciprocal is an inverse of multiplication. It is a simple and common algebraic operation that exchanges the the numerator with the denominator of a proportional expression resulted by the "$\div$" operator. When combined with differential transformations, it can sometimes dramatically change the functional pattern. For preserving the continuity, the reciprocal transformation must not be applied to a functional image with different signs[12]. The inverse transformation of T_{REC} is equivalent to T_{REC}

$$T_{\mathrm{REC}}^{-1} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = 1/u_d \end{cases} \tag{3.45}$$

[12] It has been assumed that the underlying function is continuous function. If different signs are observed in an image, it implies that there must be some points in the observation domain whose function value are zero. As such, the application of T_{REC} will yield infinity.

Factorization

$$T_{\text{FAC}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = \dfrac{u_d}{f(u_1, u_2)} \end{cases} \tag{3.46}$$

This is an important transformation that can best decompose the functional patterns derived by the operator " $\times$ ". However, it is important that the factor function $f(u_1, u_2)$ must be simple and can be easily observed by some other means. The extraction of the factor function is a task of functional pattern discovery — a function form discovery related problem. Besides simple numeric tools, function form discovery algorithms can also be employed to carry out this task. The inversion of this transformation is

$$T^{-1}_{\text{FAC}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = u_d \cdot f(u_1, u_2) \end{cases} \tag{3.47}$$

The simplest factorization transformation is the one with a linear factor

$$T_{\text{FAC}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = u_d/(u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C) \end{cases} \tag{3.48}$$

where θ and C are control parameters. The corresponding inverse transformation is

$$T^{-1}_{\text{FAC}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = (u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C) \cdot u_d \end{cases} \tag{3.49}$$

Functional Inverse

$$T_{\mathrm{INV}} : \begin{cases} v_1 = u_d \\ v_2 = u_2 \\ v_d = u_1 \end{cases} \tag{3.50}$$

This is the algebraic transformation that changes the pre-assumed dependent variable. Generally speaking, a function form

$$f(u, v, w) = 0 \tag{3.51}$$

that relates three variables together is only a functional relationship among the variables. The only criterion for a variable to be the dependent variable is the "*monodrome requirement*", which means that the variable w can be viewed as the dependent variable if and only if: $\forall u, v \in \Re$, $f(u, v, w_1) = 0$ and $f(u, v, w_2) = 0$ only when $w1 = w2$.

Functional inverse is a simple and important transformation. Firstly, combined with differential transformations, it helps to express many fundamental function forms. We will see this when we introduce the differential transformation later. Secondly, the complexity of an explicit expression of a functional relationship usually depends largely on the choice of which variable is viewed as the dependent variable and put on the left side of the equation. In other words, $w = g(u, v)$, $u = h(w, v)$ and equation (3.51) may be different representations of the same functional relationship, but the expression $g(u, v)$ could be much more complicated and harder to handle than expression $h(w, v)$.

This transformation can be applied to a function which is monotonic to the variable u_1. The inverse transformation is equivalent to the transformation itself

$$T_{\mathrm{INV}}^{-1} : \begin{cases} v_1 = u_d \\ v_2 = u_2 \\ v_d = u_1 \end{cases} \tag{3.52}$$

Independent Variable Exchange

$$T_{\text{VEX}} : \begin{cases} v_1 = u_2 \\ v_2 = u_1 \\ v_d = u_d \end{cases} \tag{3.53}$$

This transformation allows the system to manipulate the given image equally in both directions corresponding to the variables u_1 and u_2. Combining independent variable exchange with functional inverse, we can rotate variables or exchange the position of any two variables in the triple (x, y, z), provided the functional inverse transformation is applicable when necessary. Thus a more compact transformation class set could be constructed with the employment of T_{VEX}. There is no constraint on the application of this transformation. The inverse transformation of T_{VEX} is equivalent to itself

$$T^{-1}_{\text{VEX}} : \begin{cases} v_1 = u_2 \\ v_2 = u_1 \\ v_d = u_d \end{cases} \tag{3.54}$$

Differential Transformation

Differentiation is one of the most important methods in conducting mathematic analysis. In addition to the fact that many scientific laws are expressed in terms of differential equations, many geometric properties, such as slope and curvature, are expressed with differential terms. The key idea of the data transformation model is recursively simplifying a functional image until a simple image that the system can recognize is attained. Differential transformations claim their key roles in simplifying functional patterns with their superior ability in manipulating functional patterns when coordinated by other transformations.

To see the capability of differential transformations in simplifying functional patterns, let us give some examples. The fundamental function forms listed in Table 3.3 can be rewritten as differential equations with corresponding deterministic conditions. Results are listed in Table 3.5. Clearly, the original functions were all transformed into one of the simplest

Function Form	Differential Equation	Initial Condition
$f(x) = c$	$f' = 0$	$f(0) = c$
$f(x) = x^n$	$x \cdot f' - n \cdot f = 0$	$f(0) = 0$
$f(x) = x^{1/n}$	$n \cdot f' - x \cdot f = 0$	$f(0) = 0$
$f(x) = \sin(x)$	$(f')^2 + f^2 = 1$	$f(0) = 0$
$f(x) = \cos(x)$	$(f')^2 + f^2 = 1$	$f(0) = 1$
$f(x) = \tan(x)$	$f' - f^2 = 1$	$f(0) = 0$
$f(x) = \cot(x)$	$f' + f^2 = -1$	$f(\pi/2) = 0$
$f(x) = \exp(x)$	$f' - f = 0$	$f(0) = 0$
$f(x) = \log(x)$	$x \cdot f' = 1$	$f(1) = 0$

Table 3.5: Differential Representations of Fundamental Function Forms

function classes —— second order polynomials $P_2(x, f, f')$. Thus the differential transformation is the most important transformation that should be included in the transformation classes set.

The simplest format of three-variable differential transformation is the following transformation with a single partial differential.

$$T_{\text{DIF}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = \dfrac{\partial u_d}{\partial u_1} \end{cases} \tag{3.55}$$

Instead of summarizing a number of fundamental function combinations that can be simplified by T_{DIF} (as what have been done in Table 3.5) let me present an example of how T_{DIF} decomposes a function form into simple components. Differential transformation T_{DIF} is a linear operation. It means that when it is applied to a linear combination of two functions,

$$h(x, y) = f(x, y) \pm g(x, y),$$

the result is the linear combination of the differentials of the two component functions. As such, if the function $g(x, y)$ is irrelevant to x, that means g could be treated as a constant when taking differential respected to x, we obtain from applying T_{DIF} the result:

$$T_{\text{DIF}}(x, y,\ f(x, y) \pm g(x, y)\,) = (x, y,\ f'_x(x, y)\,).$$

In a function form discovery problem to find the underlying function of the form $z = f(x, y) \pm g(y)$, the application of T_{DIF} provides a possible way to reduces the complexity of the problem by decomposing the initial task into two tasks: to discover $f'_x(x, y)$ and to discover a single variable function $g(y)$. The latter could be handled by a discovery system of lower dimension.

Since it is assumed that all underlying functions are in the class C^∞, the differential transformation is always applicable. The inverse transformation of T_{DIF} is an integral.

$$T^{-1}_{\text{DIF}} : \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = \displaystyle\int_{c_0(u_2)}^{u_1} u_d \, du_1 + c(u_2) \end{cases} \tag{3.56}$$

where c_0 and c are two single-variable continuous functions, known as descriptive expressions[13]. In FFD-II, the inverse is computed by numeric integration.

3.3.2 The Primitive Classes

The choice of primitives must strike a balance among four criteria, namely generality, non-redundancy, effectiveness and simplicity.

- *Generality* means that the primitive set must be able to represent most of the final simplified features in a particular problem domain. In other words,

[13] Finding these expressions are just a task of discovering two-variable function forms from numeric data. They can be done by passing the corresponding two-dimensional sample data to FFD or a polynomial fitting algorithm.

when combined with the transformation class set, it must allow a system to discover a wide range of function forms of our interest. It emphasizes the overall discovery capability of a system using a certain description language.

- *Non-redundancy* means that the primitive set must not include those functional forms that are simplifiable by the defined transformations. This criterion emphasizes the reduction of redundancy.
- *Effectiveness* means that the primitive set must be able to represent as many functional features as possible. This criterion emphasizes the variety of primitives that a system can recognize and the "speeding up" of finding a solution.
- *Simplicity* means that the primitive must be simple to be expressed and easy to be recognized. Measured by the number of the total control parameters, a well tailored primitive should use less parameters. Furthermore, matching a functional image with the primitive should not be too costly. This criterion concerns the computational complexity of the system.

Functional Primitives

There are two elementary types of functional primitives, *normal primitive function* and *extended primitive function*. A transformation sequence transforms the original functional image related to the initial variable triple into a new functional image related to a new variable triple. If we can find a formula that describes the transformed function image, the original underlying function form can be expressed by that formula and the corresponding transformation sequence. Any single formula in the system's tool-box that can be used to describe a functional image is a normal primitive function. Expressed by an equation, a normal primitive function has the form

$$F_P(v_1, v_2, v_d) = 0 \tag{3.57}$$

where $F_P(v_1, v_2, v_d)$ is an analytic function and (v_1, v_2, v_d) is the variable triple to which a transformed functional image is related. This primitive function type is easier to handle than extended primitive function type. FFD and LINUS both employ only this type of primitive functions in their tool-box for solving function pattern recognition problems.

In contrast, BACON does not bind generated variables into pairs. Instead, it views all new generated variables as theoretical terms. Knowing how a theoretical term has been constructed from the original variables, the system finds the underlying function form related to the original variables by finding certain types of regularities[14] between selected compatible theoretical term pair. This is an example of the second type primitive functions — extended primitive functions.

An extended primitive function is a functional relationship among a set of selected theoretical terms, or variables, associated with different generated functional images currently exist in the data space[15]

$$F_P(t_1, t_2, \cdots, t_m) = 0 \tag{3.58}$$

where t_i's are variables (theoretical terms) associated with their corresponding transformed variable triples or functional images. Since a theoretical term (or its corresponding data) is obtained from applying data transformations to the original variable triple (or functional image), there exists a functional dependency between any generated theoretical term and the original variables determined by the corresponding transformation sequence. If it is found that a certain set of nodes existing in the current data space satisfies Equation 3.58, a function form concerning the original variables is discovered. This is the foundation of an extended primitive function.

Including extended primitive function in a discovery system's tool-kit makes it possible for the system to recognize a wider varieties of complex functional patterns. As a result, it either speeds up the discovery or extends the function form coverage of the system.

[14] Regularity used here is just the other word for primitive function.

[15] Recall that a function form discovery system works with two spaces — rule space and data space. Data transformations can be viewed as the rule space and the data space is composed of a set of generated functional images.

Although we consider only the normal primitive in FFD-II, it is worthy to be mentioned that the extended primitive type is an important primitive type. It allows the discovery system to use the information in the data space in a more flexible way, and provides the designer with more chances to encode desirable domain knowledge into the system. For example, extended primitive functions could be selected on the base of certain mathematically interesting and meaningful clauses. In this way, short cuts to a compact and meaningful solution are assigned to the discovery system. Dimensional analysis techniques may be used to compose an extended primitive function that is meaningful in terms of physical meaningfulness in the problems in special application domains. This issue its own is a rich research area and should be viewed as a worthwhile direction of future research.

As we discussed in Section 3.1.4, primitive recognition could be done by a simpler discovery systems[16]. In general, the more the primitive functions that the discovery system can recognize, the wider range of function forms the system can cover. However, the focus of this research is the fundamentals of function form discovery by data transformation. The compactness is emphasized, instead of powerfulness, in the construction of the primitive function set for FFD-II. Hence, only a small set of analytic function classes will be considered.

Among the analytic function classes, polynomial function class are the simplest. It is a good function class to be considered as primitive functions for the following reasons.

- Polynomial functions can approximate any function in class C^∞ to any arbitrary order. This is supported by Taylor expansion and various interpolation theorems.
- The dependency to the control parameters (coefficients of a polynomial) is linear. Thus the fitting could be easily implemented with simple and well established numerical tools.

[16] Simpler system refers to a function form discovery system that recognizes fewer function forms more efficiently than the host system. Examples of such systems that can be considered by a data transformation based discovery system include E^*, KEPLER[67] and KEDS

- The second order polynomials can quantitatively represent all the important first and second order curvatures and qualitatively represent important functional behaviors such as local minima and maxima, elliptic, parabolic, flat, hyperbolic points, etc. on a surface.
- Simple algebraic analysis methods are available, e.g. factorization, for analyzing the properties of polynomial expressions.
- Polynomials have been demonstrated to be a good choice by the existing data transformation based function form discovery systems such as FFD and LINUS. This research further supports this claim in three-variable function form discovery problems.

The following two primitive function forms are designed for FFD-II.

$$\begin{aligned} F_q : w = & \; C_5 \cdot u^2 + C_4 \cdot u \cdot v + C_3 \cdot v^2 \\ & + C_2 \cdot u + C_1 \cdot v + C_0 . \end{aligned} \tag{3.59}$$

$$\begin{aligned} F_r : w^2 = & \; C_5 \cdot u^2 + C_4 \cdot u \cdot v + C_3 \cdot v^2 \\ & + C_2 \cdot u + C_1 \cdot v + C_0 . \end{aligned} \tag{3.60}$$

The differences between F_q and F_r should not be neglected. F_q could be easily transformed into a constance function by a transformation sequence consisted of a few applications of the transformations previously defined. Considering the task of fitting six parameters as a trade off of search more nodes, it is worthwhile. However, it is usually very hard to transform an expression of the form F_r into a simpler function form using the transformations defined in this chapter. Considering that the square root function is a very common function format, it would be better to be included in the primitive set directly.

Compositional Primitives

Table 3.4 lists five operators that can be used to generate complex function forms from elementary functions. It has been pointed out that functional composition operator $\odot$ is the most important one that combines simple function forms into hard to handle complex function forms. With a single operator $\odot$, two simple functions $f(t)$ and $g(x, y)$ can be combined into a single composed function:

$$h(x, y) = (f \odot g)(x, y) = f(g(x, y)) \tag{3.61}$$

In section 3.3.1, five general-purpose transformation classes were identified. Unfortunately, none of them directly simplifies function forms given by Equation 3.61. In practice, it may happen that through a sequence of data transformations, the original functional image is transformed into a new image that satisfies Equation 3.61. In solving real world problems in a particular domain, it might be desirable that a certain set of variables (with dimensions) are combined into a dimensionless atom before a meaningful formula could be obtained (this requirement specifies an additional function expression $g(x, y)$). In the former case, a compositional primitive helps to enhance the capability of the system, and in the latter case, a compositional primitive provides a way to encoding domain knowledge.

In Equation 3.61, function $h(x, y)$ can be decomposed into two functions — the core expression $g(x, y)$ determines the "fine-grain" behavior while the function $f(t)$ determines the "global feature". The compositional primitive set specifies the functional patterns (core expressions $g(x, y)$) recognizable to the system. Through certain segmentation scheme, we can assign the discovery system with the ability to identify the functional components $g(x, y)$ and $f(t)$.

Recall that we have discussed the "cumulative methodology" in Section 3.1.4. Applying it in both of the tasks of finding a hypothetic core expression $g(x, y)$ and the one-dimension function $f(t)$ are straightforward. They both are function form discovery problems that can be solved using specific discovery methodologies or existing function form discovery systems.

Generally, the composition of a compositional primitive is closely related to the application domain. To demonstrate the idea of using the compositional primitive in a discovery system, linear functions was chosen as the only compositional primitive class recognizable to FFD-II, i.e.

$$g(x, y) = x \cdot \cos(\theta) + y \cdot \sin(\theta) \tag{3.62}$$

where θ is a control parameter. The corresponding compositional primitive is defined as

$$E_l \;:\; u \cdot \cos(\theta) + v \cdot \sin(\theta) \tag{3.63}$$

This compositional primitive addresses the recognition of cylindric functional patterns in an image. This class of images is related to a one-dimension function with a planar coordinate system rotation. The detection and fitting of this core function will be discussed later.

3.3.3 The Function Form Description Language

The function form description language used by FFD-II can now be stated. The choice of the rank value of each transformation class will be discussed in the rest parts of this chapter and in Chapter 4.

Transformation Class Set

$$\widetilde{\boldsymbol{S}_T} = \{\, \boldsymbol{T}_{\text{LOG}} \,,\; \boldsymbol{T}_{\text{REC}} \,,\; \boldsymbol{T}_{\text{FAC}} \,,\; \boldsymbol{T}_{\text{INV}} \,,\; \boldsymbol{T}_{\text{VEX}} \,,\; \boldsymbol{T}_{\text{DIF}} \,\} \tag{3.64}$$

Table 3.6 lists the definitions of each transformation classes appeared in the definition. Notice that in the table $\boldsymbol{T}_{\text{LOG}}$, $\boldsymbol{T}_{\text{REC}}$, $\boldsymbol{T}_{\text{INV}}$, and $\boldsymbol{T}_{\text{VEX}}$ are single element transformation set.

Primitive Class Set

$$\widetilde{\boldsymbol{S}_P} = \{\, \boldsymbol{F}_q \,,\; \boldsymbol{F}_r \,,\; \boldsymbol{E}_l \,\} \tag{3.65}$$

where $\boldsymbol{F}_q = \{F_q\}$, $\boldsymbol{F}_r = \{F_r\}$ and $\boldsymbol{E}_l = \{E_l\}$ (see *Eq.*(3.59), *Eq.*(3.60) and *Eq.*(3.63) for the corresponding formulas).

The Language

$$\tilde{\boldsymbol{\mathcal{L}}} = (\, \widetilde{\boldsymbol{S}_T} \,,\; \widetilde{\boldsymbol{S}_P} \,) \tag{3.66}$$

Symbol	Definition	Trans. Description	Inverse Description	Rank
$\boldsymbol{T}_{\text{LOG}}$	$\boldsymbol{T}_{\text{LOG}} = \{ T_{\text{LOG}} \}$	(3.42)	(3.43)	1
$\boldsymbol{T}_{\text{REC}}$	$\boldsymbol{T}_{\text{REC}} = \{ T_{\text{REC}} \}$	(3.44)	(3.45)	1
$\boldsymbol{T}_{\text{FAC}}$	$\boldsymbol{T}_{\text{FAC}} = \{ T_{\text{FAC}} \}$	(3.48)	(3.49)	1
$\boldsymbol{T}_{\text{INV}}$	$\boldsymbol{T}_{\text{INV}} = \{ T_{\text{INV}} \}$	(3.50)	(3.52)	0
$\boldsymbol{T}_{\text{VEX}}$	$\boldsymbol{T}_{\text{VEX}} = \{ T_{\text{VEX}} \}$	(3.53)	(3.54)	0
$\boldsymbol{T}_{\text{DIF}}$	$\boldsymbol{T}_{\text{DIF}} = \{ T_{\text{DIF}} \}$	(3.55)	(3.56)	2

Table 3.6: Transformation Classes of FFD-II

Some Relevant Sets :

- *The Base Transformation Set*

$$\widetilde{S_T} = \boldsymbol{I} \cup \boldsymbol{T}_{\text{LOG}} \cup \boldsymbol{T}_{\text{REC}} \cup \boldsymbol{T}_{\text{FAC}} \cup \boldsymbol{T}_{\text{INV}} \cup \boldsymbol{T}_{\text{VEX}} \cup \boldsymbol{T}_{\text{DIF}} \tag{3.67}$$

- *The Generated Transformation Set*

$$\widetilde{S_T}^{*} = \{ \boldsymbol{I}, \boldsymbol{T}_{\text{LOG}}, \boldsymbol{T}_{\text{REC}}, \boldsymbol{T}_{\text{FAC}}, \boldsymbol{T}_{\text{INV}}, \boldsymbol{T}_{\text{VEX}}, \boldsymbol{T}_{\text{DIF}} \}^{*} \tag{3.68}$$

- *The Primitive Set*

$$\widetilde{S_P} = \boldsymbol{F}_q \cup \boldsymbol{F}_r \cup \boldsymbol{E}_l \tag{3.69}$$

3.3.4 Transformation Macro, Redundancy and Expressiveness

The "*Transformation Macro*" technique is introduced to speed up the search for the goal. A transformation macro is formed by a combination of a sequence of the original transformations, and the search algorithm treats the result as a single transformation. It is a big step in the search space. As such, it reduces the search depth required to move from the start state to a goal state. However, on the other hand, it increases the branching factor at each state.

Since FFD-II system carries out "best first" search, the transformation macro technique is embedded in the settings of transformation ranks which is one of the major factors of the searching heuristics. Strictly speaking, this treatment is not to speed up the searching, which is commonly the motivation for introducing operator macros, but to make the transformation class set more compact without defining unnecessary transformation classes. In this section I will first examine transformation macros from the aspects of their ranks. Then I will discuss redundant transformation sequences. As a close and important issue, I will discuss the expressiveness of the function form description languages in the last section of this part.

Transformation Macros and Their Ranks

Proposition 1 *Let u, v and w be the only three variables in a problem. The five* variable shuffle *transformations*[17]

$$\textit{Shuffle 1}: \quad (u,\ v,\ w) \xrightarrow{T_{S1}} (u,\ w,\ v) \tag{3.70}$$

$$\textit{Shuffle 2}: \quad (u,\ v,\ w) \xrightarrow{T_{S2}} (v,\ u,\ w) \tag{3.71}$$

$$\textit{Shuffle 3}: \quad (u,\ v,\ w) \xrightarrow{T_{S3}} (v,\ w,\ u) \tag{3.72}$$

$$\textit{Shuffle 4}: \quad (u,\ v,\ w) \xrightarrow{T_{S4}} (w,\ u,\ v) \tag{3.73}$$

$$\textit{Shuffle 5}: \quad (u,\ v,\ w) \xrightarrow{T_{S5}} (w,\ v,\ u) \tag{3.74}$$

are all combinations of T_{VEX} and T_{INV}, and the resultant ranks are all zero, provided T_{VEX} and T_{INV} are applicable to each corresponding states.

[17] In this part, we use $(r(u,v,w),\ s(u,v,w),\ t(u,v,w))$ to denote a new variable triple $(u_n,\ v_n,\ w_n)$ that is related to the original variable triple $(u,\ v,\ w)$ by

$$\begin{cases} u_n &= r(u,\ v,\ w) \\ v_n &= s(u,\ v,\ w) \\ w_n &= t(u,\ v,\ w) \end{cases}$$

where r, s and t denote three functional expressions. However, it is required that the above mapping is "one-to-one onto" mapping.

Transformation	Composition Sequence	Resultant Rank
T_{S1}	$T_{\text{VEX}} \circ T_{\text{INV}} \circ T_{\text{VEX}}$	0
T_{S2}	T_{VEX}	0
T_{S3}	$T_{\text{VEX}} \circ T_{\text{INV}}$	0
T_{S4}	$T_{\text{INV}} \circ T_{\text{VEX}}$	0
T_{S5}	T_{INV}	0

Table 3.7: Transformation Combinations of Variable Shuffling

[*Proof*] The first part of the proposition can be proved by Table 3.7. Based on the result of the first part and considering that (1) the rank value of a transformation sequence is the sum of the rank values of each transformation in the sequence; and (2) T_{VEX} and T_{INV} are ranked zero, the second part of the proposition is obvious. $\square$

Proposition 2 *The maximum number of irrelevant variable shuffle transformation is two. In other words, we can arbitrarily choose two and only two different variable shuffle transformations defined by Equations (3.70) through (3.74) and all the rest can be derived by combinations of the two selected.*

This proposition can be easily proved by enumerating all the possible combinations of any arbitrary choice of two different variable shuffle transformations. It implies that we can replace $\boldsymbol{T}_{\text{VEX}}$ or $\boldsymbol{T}_{\text{INV}}$ in language $\tilde{\mathcal{L}}$ by a different variable shuffle transformation (or replace both) and construct a new language that is equivalent to the language $\tilde{\mathcal{L}}$ used by FFD-II.

Proposition 3 *The differential transformation*

$$T_{\text{DIF2}} \ : \ (u, v, w) \xrightarrow{T_{\text{DIF2}}} \left(u, v, \frac{\partial w}{\partial v}\right) \tag{3.75}$$

is a transformation in set $\widetilde{S_T}^*$ *which has the rank value equal to* T_{DIF}.

[*Proof*] Clearly,

$$T_{\text{DIF2}} = T_{\text{VEX}} \circ T_{\text{DIF}} \circ T_{\text{VEX}} \in \widetilde{S_T}^{*}$$

Thus,

$$\begin{aligned}
\overline{Rank}\,(T_{\text{DIF2}}) &= \overline{Rank}\,(T_{\text{VEX}} \circ T_{\text{DIF}} \circ T_{\text{VEX}}) \\
&= \overline{Rank}\,(T_{\text{VEX}}) + \overline{Rank}\,(T_{\text{DIF}}) + \overline{Rank}\,(T_{\text{VEX}}) \\
&= 0 + 2 + 0 = 2 \\
&= \overline{Rank}\,(T_{\text{DIF}})
\end{aligned}$$

The proof is completed. □

Based on Proposition 1 we can prove the following propositions in the same way as we did in the proof of Proposition 3.

Proposition 4 *Let*

$$T_{\text{LOG1}} : (u, v, w) \longmapsto (\log(u), v, w) \tag{3.76}$$

$$T_{\text{LOG2}} : (u, v, w) \longmapsto (u, \log(v), w) \tag{3.77}$$

$$T_{\text{REC1}} : (u, v, w) \longmapsto (1/u, v, w) \tag{3.78}$$

$$T_{\text{REC2}} : (u, v, w) \longmapsto (u, 1/v, w) \tag{3.79}$$

$$T_{\text{FAC1}} : (u, v, w) \longmapsto (u/(v\cos(\theta) + w\sin(\theta) + c), v, w) \tag{3.80}$$

$$T_{\text{FAC2}} : (u, v, w) \longmapsto (u, v/(w\cos(\theta) + u\sin(\theta) + c), w) \tag{3.81}$$

Then

(1) T_{LOG1} *and* T_{LOG2} *are combinations of variable shuffles and* T_{LOG}, *and have the same rank value as* T_{LOG}.

(2) T_{REC1} *and* T_{REC2} *are combinations of variable shuffle and* T_{REC}, *and have the same rank value as* T_{REC},

(3) T_{FAC1} *and* T_{FAC2} *are combinations of variable shuffles and* T_{FAC}, *and have the same rank value as* T_{FAC}.

Group No.	Group Name	Trans. Classes in the Group	Rank
1	Variable Shuffle	$\{T_{S1}\}$, $\{T_{S2}\}$, $\{T_{S3}\}$, $\{T_{S4}\}$, $\{T_{S5}\}$	0
2	Logarithm	$\boldsymbol{T}_{\text{LOG}}$, $\{T_{\text{LOG1}}\}$, $\{T_{\text{LOG2}}\}$	1
3	Reciprocal	$\boldsymbol{T}_{\text{REC}}$, $\{T_{\text{REC1}}\}$, $\{T_{\text{REC2}}\}$	1
4	Factorization	$\boldsymbol{T}_{\text{FAC}}$, $\{T_{\text{FAC1}}\}$, $\{T_{\text{FAC2}}\}$	1
5	Differential	$\boldsymbol{T}_{\text{DIF}}$, $\{T_{\text{DIF2}}\}$	2

Table 3.8: Grouping of Fundamental Transformation Classes

provided each transformation can be applied as required.

Clearly, the transformations introduced in Proposition 1, 3 and 4 are all fundamental but not all of them are necessary to be included in the definition of description language $\tilde{\mathcal{L}}$ at the same time. On the other hand, equivalent languages can be constructed by replacing some transformation classes with selected different transformation classes.

If we group all the new transformation classes into five groups (Table 3.8), we can construct equivalent languages by taking two different classes from group #1 and one from each of the remaining groups. As an example, the function form description language

$$\mathcal{L}_1 = \left(\left\{ \begin{array}{l} \{T_{\text{LOG1}}\},\ \{T_{\text{REC1}}\},\ \{T_{\text{FAC1}}\}, \\ \{T_{S1}\},\ \{T_{S3}\},\ \{T_{\text{DIF2}}\} \end{array} \right\},\ \widetilde{\mathcal{S}_P} \right) \tag{3.82}$$

is equivalent to the language $\tilde{\mathcal{L}}$. The propositions (1 through 4) show us the reasons why only $\boldsymbol{T}_{\text{VEX}}$ and $\boldsymbol{T}_{\text{INV}}$ are included in the transformation class set $\widetilde{\mathcal{S}_T}$ and their rank values are set to zero.

More Composition Properties

We have seen some basic transformation compositions in Section 3.3.4. Intuitively, two transformation sequences may sometimes reach the same state. In search for the transformation sequence which simplifies the original image into a primitive, repeating searches in two equivalent branches is what should be avoided. This is the issue of transformation redundancy analysis. Let us first formally define redundancy.

Definition 18 *Let T_1 and T_2 be two transformations (sequences) in a generated transformation set $\widehat{S_T}$ corresponding to S_T. T_1 and T_2 are said to be* equivalent *iff* (1) *they are defined on the same domain $D \subset \Re^3$, and* (2) $\forall (x,y,z) \in D$, $T_1(x,y,z) = T_2(x,y,z)$. *An equivalent transformation pair is denoted by $T_1 \equiv T_2$. An* equivalent transformation class *in $\widehat{S_T}$ is a non-empty transformation set*

$$\Psi \overset{\text{def}}{=\!=} \left\{ T \;\middle|\; \begin{array}{l} \Psi \subset \widehat{S_T}, \text{ and } \Psi \neq \Lambda; \\ \exists T_r \in \Psi, \quad \forall \bar{T} \in \widehat{S_T}, \; \bar{T} \in \Psi \; \textit{iff} \; \bar{T} \equiv T_r \end{array} \right\}. \tag{3.83}$$

where Λ denotes the empty set and T_r is a selected transformation sequence in class Ψ known as the representative *of Ψ .*

Apparently, " $\equiv$ " relation is a transitive binary relation and any two different equivalent transformation classes are disjoint. Thus, a generated transformation set can be divided into a set of disjointed equivalent classes. In other words, the collection of all equivalent transformation classes of a generated transformation set $\widehat{S_T}$, that is $\{\Psi_i\}$, is a *partition* of $\widehat{S_T}$.

Definition 19 *Let $\widehat{S_T}$ be the generated transformation set of S_T and*

$$\widehat{S_T} = \bigcup_{i=1}^{K} \Psi_i \tag{3.84}$$

where (1) K is an integer which is allowed to be infinity, (2) each Ψ_i , $i = 1, \cdots, K$, is an equivalent transformation class, and (3) $\forall i \neq j$, $\Psi_i \bigcap \Psi_j$ is an empty set. Then,

$$\{\Psi_1, \; \Psi_2, \; \cdots, \; \Psi_K\} \tag{3.85}$$

is called a ground partitioning *of $\widehat{S_T}$, and K is called the* cardinality *of $\widehat{S_T}$. It is denoted by $\overline{Card}\,(\widehat{S_T})$ or $\overline{Card}\,(S_T)$, (or simply $\left|\widehat{S_T}\right|$ or $|S_T|$).*

Definition 20 *Let* $\boldsymbol{\Psi}_i$ *be an equivalent class of a generated transformation set* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$. *A* ground transformation *of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ *in* $\boldsymbol{\Psi}_i$, *is an arbitrarily selected transformation*[18] $\Psi_i \in \boldsymbol{\Psi}_i$ *such that* $\forall T \in \boldsymbol{\Psi}_i$

$$\begin{cases} \overline{Order}\,(T) > \overline{Order}\,(\Psi_i)\,, \quad \text{or} \\ \overline{Order}\,(T) = \overline{Order}\,(\Psi_i)\,,\ \overline{Rank}\,(T) \geq \overline{Rank}\,(\Psi_i)\,. \end{cases} \tag{3.86}$$

Let $\{\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2, \cdots, \boldsymbol{\Psi}_K\}$ *be a ground partitioning of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$, *and* $\Psi_i \in \boldsymbol{\Psi}_i$ *is a ground transformation of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ *in* $\boldsymbol{\Psi}_i$, $(i = 1, 2, \cdots, K)$. *Then, the transformation set*

$$\{\Psi_1, \Psi_2, \cdots, \Psi_K\} \tag{3.87}$$

is called a ground kernel *of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$. *A ground kernel is denoted by* $\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}}$ *or* $\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}$.

Definition 21 *Let* $\boldsymbol{S}_{\mathrm{T}}$ *be a transformation class set,* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ *be the generated transformation set of* $\boldsymbol{S}_{\mathrm{T}}$, *a transformation* $T \in \widehat{\boldsymbol{S}_{\mathrm{T}}}$ *is* redundant *with respect to a ground kernel* $\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}$ *iff* $T \notin \boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}$.

Since $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ has only $|\,\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}\,|$ equivalent transformation classes, it has only $|\,\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}\,|$ ground transformations in a ground transformation set. Thus, there are only $|\,\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}\,|$ transformation combinations that are not redundant. All the rest are redundant transformations.

Definition 22 *Let* $\boldsymbol{S}_{\mathrm{T}}$ *be a base transformation set,* $\widehat{\boldsymbol{S}_{\mathrm{T}}} = \boldsymbol{S}_{\mathrm{T}}^{\star}$ *be the generated transformation set, and* $\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}} = \{\Psi_1, \Psi_2, \cdots, \Psi_K\}$ *be a ground kernel of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}$. *Then,* $\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}$ *is called a* regular ground kernel *if* $\boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}$ *satisfies*

$$\begin{cases} \forall \Psi_i \in \boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}} \\ \quad \text{if} \quad \Psi_i = T_{ik_i} \circ T_{ik_{i-1}} \circ \cdots \circ T_{i1} \\ \quad \text{then } T_{ik_{i-1}} \circ T_{ik_{i-2}} \circ \cdots \circ T_{i1} \in \boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}} \ \text{ and } T_{ik_i} \in \boldsymbol{\Theta}_{\boldsymbol{S}_{\mathrm{T}}^{\star}} \end{cases} \tag{3.88}$$

A regular ground kernel is denoted by $\overline{\boldsymbol{\Theta}}_{\boldsymbol{S}_{\mathrm{T}}}$ *or* $\overline{\boldsymbol{\Theta}}_{\boldsymbol{S}_{\mathrm{T}}^{\star}}$.

[18] Note that there could be more than one transformations in $\boldsymbol{\Psi}$ that satisfy Equation (3.86).

Proposition 5 *Let $\widehat{\boldsymbol{S}_{\mathrm{T}}}$ be any generated transformation set based on a base transformation set $\boldsymbol{S}_{\mathrm{T}}$. Then, there is at least one ground kernel that is a regular ground kernel.*

[*Proof*]

(1) Constructing a subset $\boldsymbol{\Phi}$ of $\widehat{\boldsymbol{S}_{\mathrm{T}}}$.

Let $\boldsymbol{\Phi} = \bigcup_{k=1}^{\infty} \boldsymbol{\Phi}_k$, where $\boldsymbol{\Phi}_i$ are recursively defined as

$$\begin{cases} \boldsymbol{\Phi}_{\mathrm{B}} & = \left\{ T \left| \begin{array}{l} T \in \boldsymbol{S}_{\mathrm{T}}; \\ \forall T_1, T_2 \in \boldsymbol{\Phi}_{\mathrm{B}}, T_1 \not\equiv T_2 \end{array} \right. \right\} \\ \boldsymbol{\Phi}_1 & = \boldsymbol{\Phi}_{\mathrm{B}} \\ \boldsymbol{\Phi}_i & = \left\{ T_2 \circ T_1 \left| \begin{array}{l} \forall T_1 \in \boldsymbol{\Phi}_{i-1}, T_2 \in \boldsymbol{\Phi}_{\mathrm{B}}, \quad \text{s.t.} \\ \qquad \forall T \in \bigcup_{j=1}^{i-1} \boldsymbol{\Phi}_j, T_2 \circ T_1 \not\equiv T; \\ \forall T' \text{ and } T'' \in \boldsymbol{\Phi}_i, T' \not\equiv T'' \end{array} \right. \right\} \\ & (i = 2, 3, \cdots, \infty) \end{cases} \tag{3.89}$$

(2) From the construction of $\boldsymbol{\Phi}$, it is obvious that $\forall T_1 \neq T_2 \in \boldsymbol{\Phi}, T_1 \not\equiv T_2$.

(3) $\forall T = T_1 \circ T_2 \circ \cdots \circ T_m \in \widehat{\boldsymbol{S}_{\mathrm{T}}}$, $\exists T' \in \boldsymbol{\Phi}$, such that $T \equiv T'$,

where, $T_1, T_2, \cdots, T_m \in \boldsymbol{S}_{\mathrm{T}}$.

This can be proved with mathematic induction. First, from the construction rule of $\boldsymbol{\Phi}_{\mathrm{B}}$, $\forall T \in \boldsymbol{S}_{\mathrm{T}}$, $\exists T' \in \boldsymbol{\Phi}_{\mathrm{B}} \subset \boldsymbol{\Phi}$, $T \equiv T'$. Second, assume that $\exists n$, for any transformation sequence T that is composed of less than n base transformations, there exists a transformation $T' \in \boldsymbol{\Phi}$, such that $T \equiv T'$. Now let $T^{(n+1)}$ be a transformation sequence composed of $n+1$ base transformations. We can write $T^{(n+1)} = T \circ T^{(n)}$, where T is a base transformation and $T^{(n)}$ is a transformation sequence composed of n base transformations. From the inductive assumption and the construction rule of $\boldsymbol{\Phi}$, there exists a transformation $T' \in \bigcup_{k=1}^{n+1} \boldsymbol{\Phi}_k \subset \boldsymbol{\Phi}$, such that $T^{(n+1)} \equiv T'$. Thus, by mathematic induction, the proof is completed.

Furthermore, since it is obvious that Equation 3.86 is satisfied for each element in $\boldsymbol{\Phi}$, $\boldsymbol{\Phi}$ is a ground kernel of $\widehat{\boldsymbol{S}_{\mathrm{T}}}$.

The proposition is proved by summarizing (1), (2) and (3). □

Proposition 5 provides an important theoretical result, i.e. "To a function form discovery system, if we describe the redundancy properly, the system will find any existing solution by searching only the space of non-redundant transformation sequences".

Proposition 6 *Let* $\widehat{\boldsymbol{S}_{\mathrm{T}}}'$ *and* $\widehat{\boldsymbol{S}_{\mathrm{T}}}''$ *be two generated transformation sets based on the base transformation sets* $\boldsymbol{S}_{\mathrm{T}}'$ *and* $\boldsymbol{S}_{\mathrm{T}}''$ *respectively, and assume that* $\boldsymbol{S}_{\mathrm{T}}' \in \boldsymbol{S}_{\mathrm{T}}''$. *Then, if* $\boldsymbol{\Theta}_{S_{\mathrm{T}}'}$ *is a regular ground kernel of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}'$, *there exists a regular ground kernel* $\boldsymbol{\Theta}_{S_{\mathrm{T}}''}$ *of* $\widehat{\boldsymbol{S}_{\mathrm{T}}}''$ *such that* $\boldsymbol{\Theta}_{S_{\mathrm{T}}'} \subseteq \boldsymbol{\Theta}_{S_{\mathrm{T}}''}$.

The proof of this proposition is quite straightforward and shall not be presented here. For convenience, we shall use "group" — an algebra term[19] to describe our analyses in the remaining part of this section. However, to keep the discussion focusing, we will not point out which transformation is in the "regular ground kernel", although it is an important concept to be borne in mind along the analyses.

Proposition 7

$$\left(\left\{ \begin{array}{l} I\,,\ T_{\mathrm{INV}}\,,\ T_{\mathrm{VEX}}\,,\ T_{\mathrm{VEX}} \circ T_{\mathrm{INV}}\,,\ T_{\mathrm{INV}} \circ T_{\mathrm{VEX}}\,, \\ T_{\mathrm{VEX}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{VEX}} \end{array} \right\},\ \circ \right) \tag{3.90}$$

is a non-Abelian group. Hence

$$(\{T_{\mathrm{INV}}\,,\ T_{\mathrm{VEX}}\}^{*}\,,\ \circ) \tag{3.91}$$

[19] A <u>*semi-group*</u> is a non-empty set $\boldsymbol{S}$ on which is defined a binary operation $\circledast$ such that (1) $\boldsymbol{S}$ is close under $\circledast$ and (2) $\circledast$ is associative on $\boldsymbol{S}$. A semi-group is written as $(\boldsymbol{S}, \circledast)$. If $\circledast$ is commutative, a semi-group $(\boldsymbol{S}, \circledast)$ is called a <u>*commutative semi-group*</u>. If (1) there is an identity element $e \in \boldsymbol{S}$ and (2) every element in set $\boldsymbol{S}$ has an inverse, then a semi-group $(\boldsymbol{S}, \circledast)$ is called a <u>*group*</u>. A group is called an <u>*Abelian group*</u> if it is a commutative group $(\boldsymbol{S}, \circledast)$. The <u>*order*</u> of a group is the number of elements in the group.

is a non-Abelian group of order six.

[*Proof*] Since $T_{\mathrm{INV}} \circ T_{\mathrm{INV}} = I$, and $T_{\mathrm{VEX}} \circ T_{\mathrm{VEX}} = I$, the first part of this proposition can be directly derived from Proposition 1. The second part is the result of the first part. □

Proposition 8

$$\left(\left\{ \begin{array}{l} I, T_{\mathrm{INV}}, T_{\mathrm{REC}}, T_{\mathrm{INV}} \circ T_{\mathrm{REC}}, T_{\mathrm{REC}} \circ T_{\mathrm{INV}}, \\ T_{\mathrm{INV}} \circ T_{\mathrm{REC}} \circ T_{\mathrm{INV}}, T_{\mathrm{REC}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{REC}}, \\ T_{\mathrm{INV}} \circ T_{\mathrm{REC}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{REC}} \end{array} \right\}, \circ \right) \tag{3.92}$$

is a non-Abelian group. Hence

$$(\{T_{\mathrm{INV}}, T_{\mathrm{REC}}\}^*, \circ) \tag{3.93}$$

is a non-Abelian group of order eight.

The similar result of 2-variable problem was proved by Wong[65] and we will not repeat the proof here. Eight variable triples corresponding to the transformations set in Expression (3.92) are tabulated below.

I	(u, v, w)
T_{INV}	(w, v, u)
T_{REC}	$(u, v, 1/w)$
$T_{\mathrm{INV}} \circ T_{\mathrm{REC}}$	$(1/w, v, u)$
$T_{\mathrm{REC}} \circ T_{\mathrm{INV}}$	$(w, v, 1/u)$
$T_{\mathrm{INV}} \circ T_{\mathrm{REC}} \circ T_{\mathrm{INV}}$	$(1/u, v, w)$
$T_{\mathrm{REC}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{REC}}$	$(1/u, v, 1/w)$
$T_{\mathrm{INV}} \circ T_{\mathrm{REC}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{REC}}$	$(1/w, v, 1/u)$

I, Λ, Π, Θ	see definitions	$\Lambda\Pi\Theta\Pi\Lambda\Theta\Pi\Theta$	$(1/w,\ 1/v,\ 1/u)$
$\Lambda\Theta$	$(v,\ u,\ 1/w)$	$\Pi\Lambda$	$(w,\ u,\ v)$
$\Pi\Lambda\Theta$	$(1/w,\ u,\ v)$	$\Lambda\Pi\Lambda$	$(u,\ w,\ v)$
$\Lambda\Pi\Lambda\Theta$	$(u,\ 1/w,\ v)$	$\Theta\Pi\Lambda$	$(w,\ u,\ 1/v)$
$\Pi\Theta$	$(1/w,\ v,\ u)$	$\Pi\Theta\Pi\Lambda$	$(1/v,\ u,\ w)$
$\Lambda\Pi\Theta$	$(v,\ 1/w,\ u)$	$\Lambda\Pi\Theta\Pi\Lambda$	$(u,\ 1/v,\ w)$
$\Theta\Pi\Lambda\Theta$	$(1/w,\ u,\ 1/v)$	$\Lambda\Theta\Pi\Lambda$	$(u,\ w,\ 1/v)$
$\Lambda\Theta\Pi\Lambda\Theta$	$(u,\ 1/w,\ 1/v)$	$\Pi\Lambda\Theta\Pi\Lambda$	$(1/v,\ w,\ u)$
$\Pi\Lambda\Theta\Pi\Lambda\Theta$	$(1/v,\ 1/w,\ u)$	$\Lambda\Pi\Lambda\Theta\Pi\Lambda$	$(w,\ 1/v,\ u)$
$\Pi\Theta\Pi\Lambda\Theta$	$(1/v,\ u,\ 1/w)$	$\Lambda\Pi$	$(v,\ w,\ u)$
$\Lambda\Pi\Theta\Pi\Lambda\Theta$	$(u,\ 1/v,\ 1/w)$	$\Theta\Pi$	$(w,\ v,\ 1/u)$
$\Pi\Lambda\Pi\Theta\Pi\Lambda\Theta$	$(1/w,\ 1/v,\ u)$	$\Pi\Theta\Pi$	$(1/u,\ v,\ w)$
$\Theta\Pi\Theta$	$(1/w,\ v,\ 1/u)$	$\Lambda\Pi\Theta\Pi$	$(v,\ 1/u,\ w)$
$\Pi\Theta\Pi\Theta$	$(1/u,\ v,\ 1/w)$	$\Lambda\Theta\Pi$	$(v,\ w,\ 1/u)$
$\Lambda\Pi\Theta\Pi\Theta$	$(v,\ 1/u,\ 1/w)$	$\Pi\Lambda\Theta\Pi$	$(1/u,\ w,\ v)$
$\Lambda\Theta\Pi\Theta$	$(v,\ 1/w,\ 1/u)$	$\Lambda\Pi\Lambda\Theta\Pi$	$(w,\ 1/u,\ v)$
$\Pi\Lambda\Theta\Pi\Theta$	$(1/u,\ 1/w,\ v)$	$\Theta\Pi\Theta\Lambda\Pi$	$(1/u,\ w,\ 1/v)$
$\Lambda\Pi\Lambda\Theta\Pi\Theta$	$(1/w,\ 1/u,\ v)$	$\Lambda\Theta\Pi\Theta\Lambda\Pi$	$(w,\ 1/u,\ 1/v)$
$\Theta\Pi\Lambda\Theta\Pi\Theta$	$(1/u,\ 1/w,\ 1/v)$	$\Pi\Lambda\Theta\Pi\Theta\Lambda\Pi$	$(1/v,\ 1/u,\ w)$
$\Lambda\Theta\Pi\Lambda\Theta\Pi\Theta$	$(1/w,\ 1/u,\ 1/v)$	$\Pi\Theta\Pi\Theta\Lambda\Pi$	$(1/v,\ w,\ 1/u)$
$\Pi\Lambda\Theta\Pi\Lambda\Theta\Pi\Theta$	$(1/v,\ 1/u,\ 1/w)$	$\Lambda\Pi\Theta\Pi\Theta\Lambda\Pi$	$(w,\ 1/v,\ 1/u)$
$\Lambda\Pi\Lambda\Theta\Pi\Lambda\Theta\Pi\Theta$	$(1/u,\ 1/v,\ 1/w)$	$\Pi\Lambda\Pi\Theta\Pi\Theta\Lambda\Pi$	$(1/u,\ 1/v,\ w)$
$\Pi\Theta\Pi\Lambda\Theta\Pi\Theta$	$(1/v,\ 1/w,\ 1/u)$		

Remarks: † Λ, Π and Θ stand for $\mathcal{T}_{\text{VEX}}$, $\mathcal{T}_{\text{INV}}$, and $\mathcal{T}_{\text{REC}}$ respectively.
‡ " $\circ$ " is the only implicit operator which connects two transformations.

Table 3.9: Different Non-redundant Combinations of $\mathcal{T}_{\text{REC}}$, $\mathcal{T}_{\text{INV}}$ and $\mathcal{T}_{\text{REC}}$.

Apparently the combination of independent variable exchange, functional inverse and reciprocal result in different reciprocal format transformations. All possible variations are tabulated in Table 3.9. Thus the following proposition is given without a detailed proof.

Proposition 9

$$(\{I\,,\ T_{\mathrm{VEX}}\,,\ T_{\mathrm{INV}}\,,\ T_{\mathrm{REC}}\}^{*}\,,\ \circ) \tag{3.94}$$

is a non-Abelian group of order 48.

From Table 3.9 we can see that the maximum number of consequently applying of the three transformations is nine. Any more than that would be redundant. Furthermore, there are 29523 different symbolic combinations of that three transformations up to the length of 9. It means that more than 99.8% of them are redundant. Therefore, the importance of redundancy analysis is demonstrated.

Proposition 10 *The two transformation sequences $T_{\mathrm{REC}} \circ T_{\mathrm{DIF}}$ and $T_{\mathrm{INV}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{INV}}$ are equivalent, provided each transformation is applicable. Thus the sequence $T_{\mathrm{INV}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{INV}}$ is redundant.*

[*Proof*] First we should note the identity of the applicabilities of the two transformation sequences. If there is a point in the observation domain where $\partial w(u_0, v_0)/\partial u = 0$, both sequences will not be applicable. Otherwise, both of them are applicable.

Let $w = f(u, v)$ be a function which is differentiable with respect to variable u, and $w = f(u, v)$ is invertible with respect to variable u with a inverse function $u = g(w, v)$ such that $g(w, v)$ is differentiable with respect to variable w. Then by applying the transformations in the corresponding sequence one by one according to the definitions of the data transformations, the original image will be transformed into the following two images

$$T_{\mathrm{REC}} \circ T_{\mathrm{DIF}} \quad : \quad (u,\ v,\ w) \longmapsto \left(u,\ v,\ \frac{1}{f'_u(u,\ v)}\right) \tag{3.95}$$

$$T_{\mathrm{INV}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{INV}} \quad : \quad (u,\ v,\ w) \longmapsto (u\,,\ v\,,\ g'_w(\,f(u,\ v),\ v\,)\,) \tag{3.96}$$

Let $p_0 = (u_0, v_0, w_0) \in \Re^3$, $p = (u, v_0, w) \in \Re^3$ such that $w_0 = f(u_0, v_0)$ and $w = f(u, v_0)$. By definition,

$$\left.\frac{\partial w}{\partial u}\right|_{p_0} = f'_u(u_0, v_0) = \lim_{p \to p_0} \frac{f(u, v_0) - f(u_0, v_0)}{u - u_0} \tag{3.97}$$

Considering the continuity and differentiability, (3.97) can be rewritten as

$$\begin{aligned} \left.\frac{\partial w}{\partial u}\right|_{p_0} &= \lim_{p \to p_0} \frac{w - w_0}{u - u_0} \\ &= \frac{1}{\lim\limits_{p \to p_0} \dfrac{u - u_0}{w - w_0}} \\ &= \frac{1}{\left.\dfrac{\partial g(w, v)}{\partial w}\right|_{p_0}}. \end{aligned} \tag{3.98}$$

Thus $T_{\text{REC}} \circ T_{\text{DIF}}(p_0) = T_{\text{INV}} \circ T_{\text{DIF}} \circ T_{\text{INV}}(p_0)$. The first part of the proposition is proved.

Since $\overline{Order}\ (T_{\text{INV}} \circ T_{\text{DIF}} \circ T_{\text{INV}}) = 3 > \overline{Order}\ (T_{\text{REC}} \circ T_{\text{DIF}}) = 2$, $T_{\text{INV}} \circ T_{\text{DIF}} \circ T_{\text{INV}}$ is redundant. □

Proposition 11 *Transformation sequence $T_{\text{LOG}} \circ T_{\text{REC}}$ is identical to the transformation T_{LOG} regarding to the simplicity of any given functional form. Thus $T_{\text{LOG}} \circ T_{\text{REC}}$ could be viewed as redundant in a discovery system.*

[*Proof*] Let (u, v, w) be a variable triple, where a functional relationship $w = f(u, v)$ exists. Then,

$$T_{\text{LOG}}(u, v, w) = (u, v, \log(|w|)),$$

$$T_{\text{LOG}} \circ T_{\text{REC}}(u, v, w) = (u, v, -\log(|w|)).$$

Since T_{LOG} and T_{REC} are applicable under the same constraint, therefore simple one-to-one correspondence between the image $T_{\text{LOG}}(u, v, f(u, v))$ and the image $T_{\text{LOG}} \circ T_{\text{REC}}(u, v, f(u, v))$ exists (if applicable), and the two images are identical in terms of discoverability and complexity. □

There are three similar propositions concerning transformation T_{VEX}. We neglect the similar repeating proofs.

Proposition 12 *Transformation sequence* $T_{\mathrm{FAC}} \circ T_{\mathrm{VEX}}$ *is identical to transformation sequence* $T_{\mathrm{VEX}} \circ T_{\mathrm{FAC}}$ *. Thus* $T_{\mathrm{FAC}} \circ T_{\mathrm{VEX}}$ *could be viewed as redundant in a discovery system.*

Proposition 13 *Transformation sequence* $T_{\mathrm{LOG}} \circ T_{\mathrm{VEX}}$ *is equivalent to transformation sequence* $T_{\mathrm{VEX}} \circ T_{\mathrm{LOG}}$ *. Thus* $T_{\mathrm{LOG}} \circ T_{\mathrm{VEX}}$ *could be view as redundant.*

Proposition 14 *Transformation sequence* $T_{\mathrm{REC}} \circ T_{\mathrm{VEX}}$ *is equivalent to transformation sequence* $T_{\mathrm{VEX}} \circ T_{\mathrm{REC}}$ *. Thus* $T_{\mathrm{REC}} \circ T_{\mathrm{VEX}}$ *could be view as a redundant.*

Last three propositions can be summarized by "For any transformation sequence composed of the transformations in the transformation set of language $\tilde{\mathcal{L}}$, if it is not redundant and it contains T_{VEX}, then the transformation to the left of T_{VEX} (if any) can only be T_{INV} or T_{DIF}". The following two redundancy propositions concerning transformation T_{FAC} are apparent, thus the proofs are neglected.

Proposition 15 *Let* $T_{\mathrm{F1}}, T_{\mathrm{F2}}, \cdots, T_{\mathrm{F}n} \in \boldsymbol{T}_{\mathrm{FAC}}$ *be* n *different factorization transformations defined by Equation (3.46). Then,* $\forall T_1, T_2 \in \{T_{\mathrm{F1}}, T_{\mathrm{F2}}, \cdots, T_{\mathrm{F}n}\}^*$*,* $T_1 \equiv T_2$ iff *sequences* T_1 *and* T_2 *contain exactly the same number of each transformation* $T_{\mathrm{F1}}, T_{\mathrm{F2}}, \cdots$ *and* $T_{\mathrm{F}n}$*. Thus, if let*

$$\mathcal{F} = \left\{ T_{\mathrm{F1}}^{\mathrm{k1}} \circ T_{\mathrm{F2}}^{\mathrm{k2}} \cdots \circ T_{\mathrm{F}n}^{\mathrm{k}n} \,\middle|\, \forall \mathrm{k1}, \mathrm{k2}, \cdots, \mathrm{k}n \in \boldsymbol{Z}^+ \right\} \tag{3.99}$$

where $\boldsymbol{Z}^+$ *is the set of all non-negative integers, then*

$$\mathcal{F} = \overline{\boldsymbol{\Theta}}_{\{T_{\mathrm{F1}}, T_{\mathrm{F2}}, \cdots, T_{\mathrm{F}n}\}^*} \tag{3.100}$$

Proposition 16 *Let* $T_{\mathrm{F1}}, T_{\mathrm{F2}}, \cdots, T_{\mathrm{F}n} \in \boldsymbol{T}_{\mathrm{FAC}}$ *be* n *different factorization transformations defined by Equation (3.46), and let* $\mathcal{F}$ *be the transformation set*

$$\mathcal{F} = \left\{ T_{\mathrm{F1}}^{\mathrm{k1}} \circ T_{\mathrm{F2}}^{\mathrm{k2}} \cdots \circ T_{\mathrm{F}n}^{\mathrm{k}n} \,\middle|\, \forall \mathrm{k1}, \mathrm{k2}, \cdots, \mathrm{k}n \in \boldsymbol{Z} \right\} \tag{3.101}$$

where $\boldsymbol{Z}$ denotes the set of all integers, and $T_{\mathrm{F}i}^{\mathrm{k}i}$, $(i = 1, 2, \cdots, n)$ denote the transformation sequences composed of $\mathbf{k}i$ *factorizations*

$$T_{\mathrm{F}i}^{\mathrm{k}i} = \begin{cases} I & \text{if } \mathrm{k}i = 0 \\ \prod_{i=1}^{\mathrm{k}i} T_{\mathrm{F}i} & \text{if } \mathrm{k}i > 0 \\ \prod_{i=1}^{\mathrm{k}i} T_{\mathrm{F}i}^{-1} & \text{otherwise} \end{cases}$$

respectively. Then,

$$\mathcal{F} = \overline{\Theta}_{\{T_{\mathrm{REC}},\, T_{\mathrm{F1}},\, T_{\mathrm{F2}}, \cdots, T_{\mathrm{F}n}\}^{*}} \tag{3.102}$$

The Expressiveness

The key idea of data transformation based function form discovery methodology is that the system can recognize some primitive forms and can transform others into what it can recognize. The following proposition is an interpretation of this idea by "recursion".

Proposition 17 *Let $\mathcal{L} = (\mathcal{S}_{\mathrm{T}}, \mathcal{S}_{\mathrm{P}})$ be a function form description language, (u_1, u_2, u_d) and (v_1, v_2, v_d) be two variable triples related by a transformation in $\mathcal{S}_{\mathrm{T}}^{*}$, i.e.*

$$T(u_1, u_2, u_d) = (v_1, v_2, v_d) \;:\; \begin{cases} v_1 &= G_1(u_1, u_2, u_d) \\ v_2 &= G_2(u_1, u_2, u_d) \\ v_d &= G_3(u_1, u_2, u_d) \end{cases}$$

where $T \in \mathcal{S}_{\mathrm{T}}^{*}$ and G_1, G_2 and G_3 are three functional expressions which specify the relationship between the variable triples. If the functional image respects to (v_1, v_2, v_d) matches with a function form $f(v_1, v_2, v_d) = 0$ in $\mathcal{L}$, then the functional image respects to (u_1, u_2, u_d) matches with a function form in $\mathcal{L}$. In other words, the function form $f(G_1(u_1, u_2, u_d), G_2(u_1, u_2, u_d), G_3(u_1, u_2, u_d)) = 0$ is discoverable by a system based on language $\mathcal{L}$, provided $f(v_1, v_2, v_d)$ is discoverable by the same system.

[*Proof*] From given, we can assume that (D_T, D_P) is a function form description of variable triple $(v_1,\ v_2,\ v_d)$ and its corresponding functional image $\mathcal{O}_{(v_1,v_2,v_d)}$. Since the functional image $\mathcal{O}_{(u_1,u_2,u_d)}$ respects to $(u_1,\ u_2,\ u_d)$ can be transformed into $\mathcal{O}_{(v_1,v_2,v_d)}$ by T, i.e. $T(\mathcal{O}_{(u_1,u_2,u_d)}) = \mathcal{O}_{(v_1,v_2,v_d)}$, the underlying function form of $\mathcal{O}_{(u_1,u_2,u_d)}$ is thus $(D_T \circ T, D_P)$. $\boxed{}$

Proposition 17 provides us with a simple way to evaluate the expressiveness of a language, or more specifically, the constitution of the transformation class set employed by a function form description language. If a number of meaningful variable triples could be enumerated via the applications of the transformations defined in the transformation class set, generally speaking, it is a positive supporting fact for the language to be a language with good expressiveness.

Trans. Name	Transformation Sequence	New Triple	Rank
T_{D0}	$T_{\mathrm{DIF}} \circ T_{\mathrm{VEX}}$	$(u,\ v,\ w'_v)$	0
T_{D1}	$T_{\mathrm{REC}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{INV}}$	$(w,\ v,\ w'_u)$	3
T_{D2}	$T_{\mathrm{VEX}} \circ T_{\mathrm{REC}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{VEX}}$	$(u,\ w,\ w'_v)$	3
T_{D3}	$T_{\mathrm{REC}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{LOG}} \circ T_{\mathrm{INV}}$	$(w,\ v,\ u \cdot w'_u)$	4
T_{D4}	$T_{\mathrm{VEX}} \circ T_{\mathrm{REC}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{LOG}} \circ T_{\mathrm{INV}} \circ T_{\mathrm{VEX}}$	$(u,\ w,\ v \cdot w'_v)$	4
T_{D5}	$T_{\mathrm{DIF}} \circ T_{\mathrm{LOG}}$	$(u,\ v,\ w'_u/w)$	3
T_{D6}	$T_{\mathrm{VEX}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{LOG}} \circ T_{\mathrm{VEX}}$	$(u,\ v,\ w'_v/w)$	3

Table 3.10: Some Attainable Triples of The Language $\tilde{\mathcal{L}}$.

As an example, Table 3.10 tabulates a number of the transformation attainable variable triples of $\tilde{\mathcal{L}}$. Since our goal is to develop a general-purpose function form discovery system without any specified application domain, we examine language $\tilde{\mathcal{L}}$ in this dimension by list only those triples with terms which are widely confronted in mathematics textbooks. All of the triples listed in Table 3.10 are related to partial derivatives of up to the order 1. The major variations of equations of the first order derivatives contain five theoretical terms, i.e. u, v, w, w'_u, and w'_v. Each resultant triple listed in Table 3.10 contains only one of the terms w'_u and w'_v. This is unfortunately correct. Not all triples that contain three of the five elementary terms can be generated by a transformation sequence specified by $\tilde{\mathcal{L}}$'s transformation class set. Let us examine the triple (v, w'_u, w'_v) as an example.

Using the transformation sequence T_{D1} in Table 3.10, we have

$$T_{\text{D1}}(u, v, w) = (w, v, w'_u)$$

We can continue manipulate the triple by applying the transformations functional inverse and independent variable exchange

$$T_{\text{VEX}} \circ T_{\text{INV}}(w, v, w'_u) = (v, w'_u, w)$$

It seems that we are one step away from our goal triple (v, w'_u, w'_v). Let us now try transformation T_{DIF}

$$(u_1, u_2, u_d) \xmapsto{T_{\text{DIF}}} (v_1, v_2, v_d) \ : \ \begin{cases} v_1 = u_1 \\ v_2 = u_2 \\ v_d = \dfrac{\partial u_d}{\partial u_1} \end{cases}$$

on the triple

$$u_1 = v \ , \ u_2 = w'_u(u, v) \ , \ u_d = w(u, v),$$

we find that

$$\begin{cases} v_1 = v \ , \ v_2 = w'_u(u, v) \\ v_d = \lim\limits_{\Delta v \to 0} \left.\dfrac{\Delta w(u, v)}{\Delta v}\right|_{\Delta w'_u(u,v)=0} \end{cases}$$

Since

$$\Delta w'_u(u, v) = w''_{uu}\Delta u + w''_{uv}\Delta v$$

we have

$$\lim_{\Delta v \to 0} \frac{\Delta u}{\Delta v} = -\frac{w''_{uv}}{w''_{uu}}$$

Thus

$$\begin{aligned} v_d &= \lim_{\Delta v \to 0} \left. \frac{\Delta w(u,v)}{\Delta v} \right|_{\Delta w'_u(u,v)=0} \\ &= \lim_{\Delta v \to 0} \left. \frac{w'_u(u,v)\Delta u + w'_v(u,v)\Delta v}{\Delta v} \right|_{\Delta w'_u(u,v)=0} \\ &= -\frac{w''_{uv}}{w''_{uu}} w'_u + w'_v \end{aligned}$$

Instead of (v, w'_u, w'_v), the process results in the triple

$$(v\,,\; w'_u\,,\; -\frac{w''_{uv}}{w''_{uu}} w'_u + w'_v).$$

What happened is that when we try to get the second first order partial derivative, two requirements should be satisfied at the same time. First, variable v should be at the first place of the variable triple. This has been met in our calculation. Second, the variable u must be at the second place of the variable triple for holding it constance. On the other hand, the term w'_u must appear in the same variable triple at the same time. This is a contradiction to the constraint that we can only put three terms in a variable triple. Similarly, we cannot find a way to get a variable triple of the form (w, w'_u, w'_v) that is important for presenting a whole class of first order partial differential equations[20]. Furthermore, handling second order partial differential equations requires that the eight terms, u , v , w , w'_u , w'_v , w''_{uu} , w''_{uv} and w''_{vv}, be put into a tuple.

The above analysis shows us an important conclusion. The function form description language $\tilde{\mathcal{L}}$ has a major limitation in expressing general partial differential equations. Considering the size of the transformation class set and primitive set, it is not a surprise. However, the analysis shows us also the large room to improve. Intuitively, we can modify the transformation set to enhance the ability to transform. For example, we can include a new transformation that directly generate the triple (w, w'_u, w'_v). The side effects of doing so is the introduction of computational redundancy and the increase of the search space. The other way is to introduce the use of extended primitive functions (page 71). We can

[20] The general non-linear partial differential equation has the form $F(u\,,\, v\,,\, w\,,\, w'_u\,,\, w'_v) = 0$.

also encode other human expertise knowledge with a set of prototypes and put them into the discovery system's tool-box as extended primitives. This is a more flexible way.

Though both extensions are worthwhile research directions, especially combined with a particular application domain, they will be viewed as future research subjects for keeping this research reasonably focussed.

Chapter 4

FFD-II —— A Function Form Discovery Model and Its Implementation

We have discussed the theoretical issues of function form discovery by data transformation in the last chapter. A function form description language $\tilde{\mathcal{L}}$, which describes a three-variable function form with a transformation sequence and a simple specific functional pattern — functional primitive or compositional primitive, has been introduced. In this chapter, we will move to the issue of the design and implementation of the new function form discovery system, the FFD-II system.

FFD-II is designed to discover function forms with three variables. By taking data transformation approach, pioneered by Wong with his FFD system, FFD-II can find complex function forms that are not restricted to a few specified function form classes. As has been pointed out in the review of related works, most previous systems share a common limitation of finding function forms only in a small number of function form classes. Thus the new system surpasses all those systems in the categories of formula construction and data analysis, such as BACON and E^*. However, unlike FFD, FFD-II's discovery model is a direct model. To overcome the difficulties of increased complexities and large computational error, an adaptive error control technique is employed by the new system.

In the first part of this chapter, I will discuss the major challenge from the multi-variable function form discovery problems. The discussion introduces the general background of the design of FFD-II. Following the introduction, I will show that an indirect system has only limited capability in finding multi-variable function forms and why it is possible that a direct model can work better in solving the problems. This discussion explains why we chose the direct model for our new system. The third section is an overview of the architecture of FFD-II . In the fourth section, the design choices of numeric recipes will be presented. And in the final section, our discussion will focus on the implementation of adaptive error control.

4.1 The Major Challenge From Multi-variable Problems

Multi-variable function form is one that has three or more variables. To discover multi-variable function forms is much more challenging than two-variable problems for a number of reasons.

1. There are many more diverse function forms in high dimension problems than in low dimension problems.

2. Observing elementary features such as monotonicity and periodicity is more difficult in high dimension cases. In two-variable problems, basic analytic and geometric properties, such as slope and curvatures can be expressed by ordinary derivatives. In multi-variable problems, elementary analytic and geometric properties, such as slope, gradient and curvatures, can only be expressed by special combinations of partial derivatives. The difficulty of analyzing these properties increases with the dimension of the problem.

3. The complexity of multi-variable approximation is greater than that of two-variable cases. Fitting a surface or hyper-surface is much more difficult than fitting a curve. Usually, estimating partial derivatives from numeric observation data needs more effort and is less accurate than estimating ordinary derivatives.

4. In multi-variable problems, to describe deterministic conditions, such as initial values and boundary conditions, are usually more difficult than in two-variable problems. In two-variable problems, describing these conditions is simply the problem of finding parameter values and that can be done using simple numeric approximation tools. In high dimension problems, deterministic conditions usually can only be expressed by a functional relationship between selected variables. As such, they can only be handled either as a sub-discovery-task in lower dimension or within a small range under certain simplification assumptions.

5. The problem size increases dramatically as the increase of the dimension of the problem. First, to meet the need of dealing with wider diversity of function forms, the search space in solving function form discovery problems must be extended. Second, the size of the observation data set in multi-variable problems is much larger than that in two-variable problems. For example, suppose a sample data set of N observations with $2N$ floating-point numbers can provide us with sufficiently fine step accuracy in a two-variable function form discovery problem. To achieve same accuracy level in a 3-variable function form discovery problem, an observation data set of N^2 samples with $3N^2$ floating-point numbers is necessary. That means the needs of both larger memory space and more arithmetic operations in processing the data set.

Existing discovery systems are still very poor at solving multidimensional problems. Most are implemented in an indirect way using variable freezing technique. They reduce the dimension by holding all but one independent variables constant at one time. Once all subtasks in lower dimension have been solved, a unification strategy is used to combine them into a uniform result. This approach were taken by BACON and all its followers. It is also the strategy with which FFD was extended to discover families of functions. However, this technique cannot cope with the rich forms of multi-variable functions. This research tackles the challenge of multi-variable function form discovery problems with a new data transformation based direct approach. The system performs direct three-dimensional data transformations and recognizes functional patterns directly from the transformed three-dimensional image.

4.2 The Direct Model

FFD-II is designed on the base of the function form description language $\tilde{\mathcal{L}}$ introduced in the last chapter. As such, it searches in the space of three dimensional transformations (the operation) and matches the transformed functional image with a primitive pattern with three variables. In other words, it is a direct method. Figure 4.1 depicts the direct model in general. As a direct model, either a single node or a set of nodes connected with an "*And Arch*" can be created under a node. In the figure, A, B and D are independent nodes, and nodes C and c are a pair of nodes connected by an "*And Arch*". In general, when two or more nodes are connected with an "*And Arch*", the search algorithm must find a goal node as a sub-goal under each of them. The solution is the unification of all sub-goals. However, FFD-II 's search strategy is simpler. When the initial discovery task is split into two sub-tasks at a search node, one of them is viewed as a "*dominant*" sub-task, while the other is a sub-task associated with the dominant sub-task ("*subordinate*" sub-task). All the dominant sub-tasks are function form discovery problems of three variables, whereas all subordinate sub-tasks are function form discovery problems in a dimension reduced space, and could be solved by an existing two-variable function form discovery system. In other word, this research focuses only on finding the dominant solution path in three-dimension space. As depicted in Figure 4.2, FFD-II aimed to find the goal node "G". It passes subordinate dimension reduced subtasks, finding `Sol-1` through `Sol-k`, to a supporting two-variable function form discovery system[1]. The solution to the original discovery problem is the combination of all discovery results.

Indirect models are contrary to direct models. They arbitrarily split the original multi-variable problem into subproblems of lower dimension and solve them separately one at a time (Figure 4.2). When all of solutions to the subproblems are obtained, labeled by `Sol-1` through `Sol-k` in Figure 4.2, the system uses some certain methods to combine them together and generates the solution to the original problem of high dimension.

Most of the previous multi-variable function form discovery models are indirect models

[1]FFD and polynomial fitting are chosen to carry out such discovery tasks.

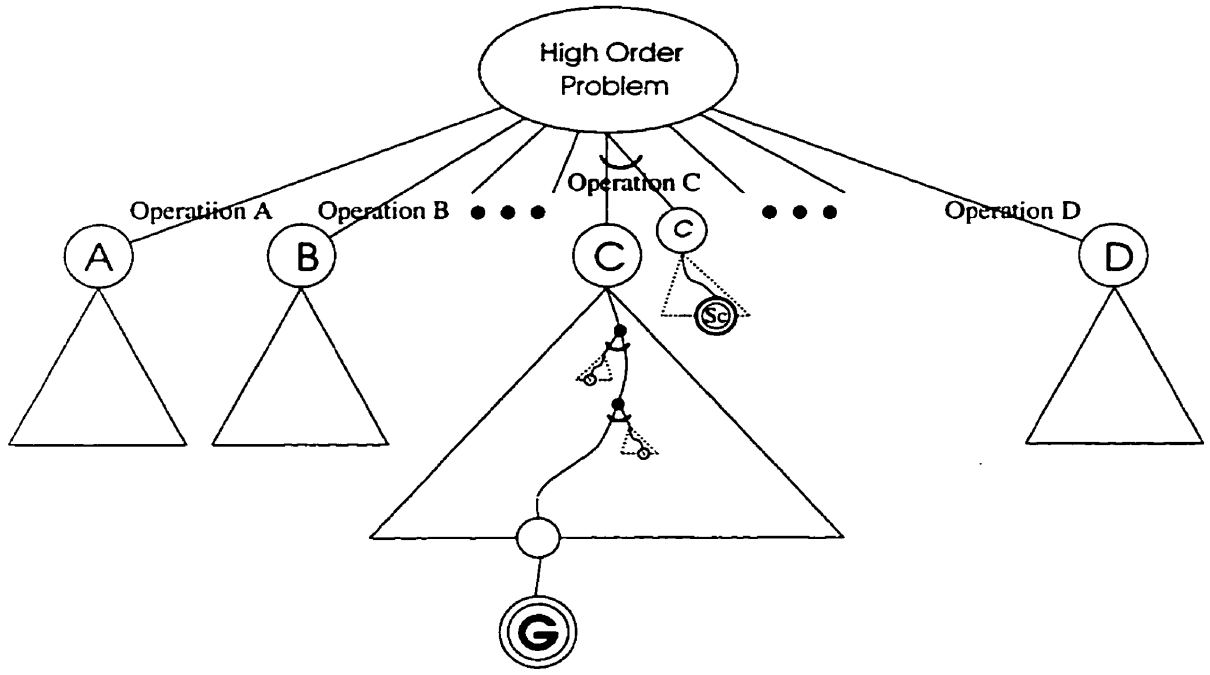


Figure 4.1: Function Form Discovery by Data Transformation
— the And-Or-Search-Tree of the Direct Model

(The node labeled by italic lowercase "c" is a subproblem in lower dimension, and the dotted triangles denote the processes of problem solving in lower dimension.)

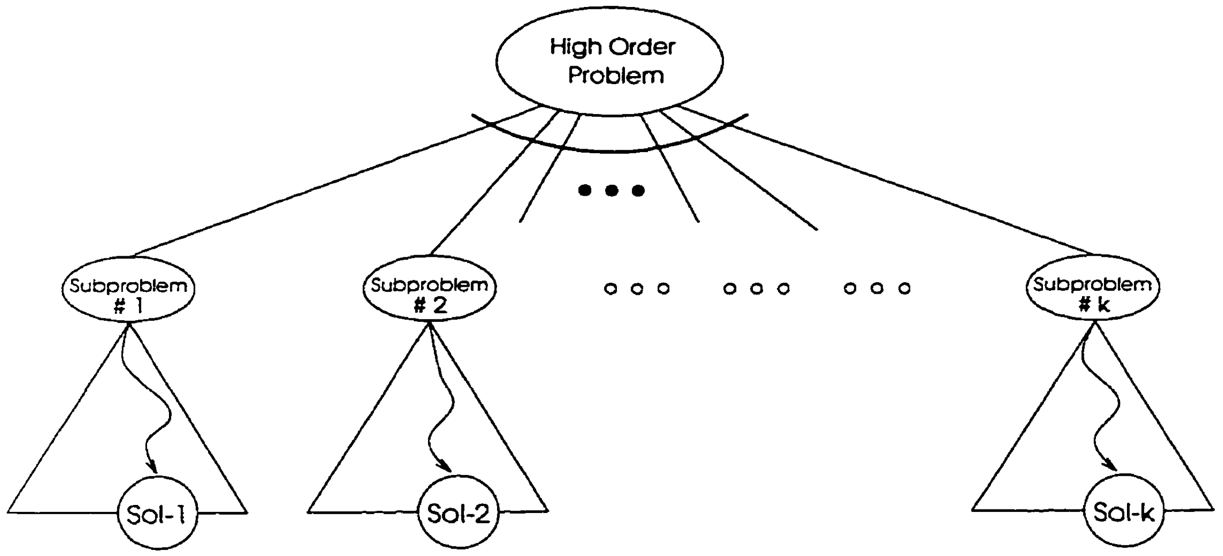


Figure 4.2: Function Form Discovery by Data Transformation
— the And-Or-Search-Tree of the Indirect Model

(Each subproblem is a dimension reduced problem,
and only the root node is an *And* node.)

utilizing a technique called variable freezing. The detailed example of this method will soon be presented in the analyses of the FFD family of functions discovery system.

In this part, I am going to discuss the advantages of direct models over indirect models. Since the FFD family of functions discovery system[2] is the only system in the category of data transformation approach that can handle multi-variable problems, the discussion will be based on the comparisons between theFFD family of functionsdiscovery system and FFD-II . We shall be able to see why it is important to create a new system that takes direct approach to the problem at the end of this section.

[2] family of functions discovery problem is a special type of multi-variable function form discovery problems. From now on, FFD refers to the extended version of FFD that handles family of functions discovery problems wherever it is used to solve three-variable function form discovery problems.

4.2.1 The Indirect Model of FFD

FFD was originally developed to discover two-variable function forms $y = f(t)$. As an extension, it can deal with parameterized two-variable function forms $y = f(t, \phi)$ (family of functions), where $\phi \subset \Re^n$ is a parameter vector. Generally speaking, parameters are just another kind of variables. Hence, FFD family of functions discovery system can be viewed as a special three-variable function form discovery system when the dimension of the control parameter vector ϕ equals one[3]. In the discussion of this section, FFD refers to the FFD family of functions discovery system.

Before going to details, I shall describe some terminologies that are necessary for introducing the discovery methodology of FFD. It should be mentioned that terminology listed below only applies to the discussion of FFD, and should not confuse us in the rest of this thesis.

Function — A function f is a single variable function $y = f(t)$ in class C^∞ within a specified domain $t \in D \subseteq \Re$.

Function Form — A function form (F, Φ) is made up of a set of parameters[4] $\Phi \subset \Re^n$ and a mapping $F : \Re \mapsto \Re$, where $\Re$ is the set of real numbers. In other words, a function form is a collection of one dimensional functions indexed by a set of parameters $\Phi \in \Re^n$.

[3] However, the difference between a parameter and a variable is that a parameter usually reflects only one simple functional dependency "pattern" or "feature", while the function value could be related to an independent variable in a more complex way. For example, in the formula of uniformly accelerated motion $s = at^2 + v_0t$, it is easy to identify a and v_0 as the parameters and t as the variable since the function value depends on a and v_0 linearly while on t quadratically. From the application point of view, parameters could be identified by the context of the application and by the fact that the parameter space is sampled much more coarsely than the partitioning of the domain interval.

[4] Though in principle multi-parameter is allowed, FFD considered only the case of function forms with a single parameter. That is only slightly different to the three-variable function form discovery problems addressed by this research. Thus hereafter, we consider the parameter ϕ as values instead of vectors and $\Phi \subset \Re$.

Function Form Instance — An instance f_ϕ of a function form (F, Φ) is a function $f_\phi : \Re \mapsto \Re$, where ϕ is a particular element of Φ, such that $f_\phi(t) = F(t, \phi)$, $\forall t \in \Re$.

Sample — A sample S_P of a given function f is a set of ordered pairs of real numbers

$$S_P(f) := \{(t_i, y_i) \mid t_i \in P \ \ and \ \ y_i = f(t_i)\},$$

where $P = \{t_i \mid i = 1, \cdots, N \ \ and \ \ t_1 < t_2 < \cdots < t_N\}$ is a partitioning of the function domain of f.

Observation — An observation $\mathcal{O}_{\Phi'}$ of the form (F, Φ) is a set of samples

$$\mathcal{O}_{\Phi'}\left[(F, \Phi)\right] := \left\{S_{P_\phi}(f_\phi) \mid \phi \in \Phi' \subset \Phi\right\},$$

where: (1) Φ' is a finite subset of Φ,

(2) f_ϕ is the instance of (F, Φ) corresponding to a valued control parameter in a partitioned control parameter set Φ', $\phi \in \Phi'$, and

(3) P_ϕ is the partitioning of the domain of f_ϕ.

Notice that Φ' is a partitioning of parameter space Φ, and we are interested in only the function forms with one parameter, $\Phi \subseteq \Re$.

Fitting of A Function Form — A fitting of a function form $(\hat{F}, \hat{\Phi})$ to an observation $\mathcal{O}_\Phi$ is a mapping

$$\mathcal{M} : \Phi' \mapsto \hat{\Phi}$$

so that for each $\phi \in \Phi'$, the sample $S_{P_\phi}(f_\phi)$ is identified with the instance $\hat{f}_{\mathcal{M}(\phi)}$ of $(\hat{F}, \hat{\Phi})$. $\Phi' \subset \Phi \subseteq \Re$ is referred to as the *control parameter* and $\hat{\Phi} \subseteq \Re^n$ for any integer n, is the *descriptive parameters*[5].

Transformation and Primitive — A transformation is a parameterized continuous mapping

$$\mathcal{T} : \Re^2 \mapsto \Re^2$$

[5] Notice that the number of control parameter can only be one for the case studied here, while the number of descriptive parameters could be zero to any give integer number.

and a primitive is a continuous one dimensional function $y = P(t)$. FFD employs five basic transformations, tabulated below, in its operation tool-box.

Trans.	**Definition**	**Trans.**	**Definition**
Inverse	$\Theta : (t, y) \mapsto (y, t)$	Logarithm	$\Lambda : (t, y) \mapsto (t, \log\|y\|)$
Reciprocal	$\Psi : (t, y) \mapsto (t, 1/y)$	Factorization	$\Pi : (t, y) \mapsto (t, \frac{y}{x-x_0})$
Differential	$\Delta : (t, y) \mapsto (t, y')$		

A primitive is a parameterized one dimensional continuous function that a transformed observation can match with. The primitive function set of FFD consists the following three quadratic functions.

$$\begin{aligned} y^2 + c_1t^2 + c_2t + c3 &= 0, \\ ty + c_1t^2 + c_2t + c3 &= 0, \\ y + c_1t^2 + c_2t + c3 &= 0. \end{aligned}$$

All the parameters appeared in a transformation or a primitive are viewed as descriptive parameters.

Unification Transformations — Let U be an invertible transformation which is parameterized by a parameter vector $\vec{\alpha}$ in $\Re^k$. An observation

$$\mathcal{O}_{\Phi'} = \{S_{\phi_1}, \cdots, S_{\phi_N}\}$$

of function form (F, Φ) is said to be *unified* by the class of transformations $U[\vec{\alpha}_\Phi]$ if there exists a set of vectors

$$\mathcal{V} = \{\vec{\alpha}_{\phi_1}, \cdots, \vec{\alpha}_{\phi_N}\},$$

in which the vector $\vec{\alpha}_{\phi_i}$ corresponds to the control parameter value ϕ_i , such that the image

$$U_\mathcal{V}(\mathcal{O}_{\Phi'}) = \{ \ (t_j, y_j) \ \mid \ \exists \ 1 \le i \le N, \quad (t_j, y_j) \in U[\vec{\alpha}_i](S_{\phi_i}) \ \},$$

is a single smooth function. U will be referred to as the *unification transformation*. $\vec{\alpha}$ will be referred to as the *descriptive parameter*; and, in particular, $\vec{\alpha}_i$ is said to contain the descriptive parameter values specifically associated with the function sample S_{ϕ_i}.

Similarization Transformations — The member samples of an observation

$$\mathcal{O}_{\Phi'} = \{S_{\phi_1}, \cdots, S_{\phi_N}\}$$

are said to be similar to one another if there exists an invertible transformation class T and an associated parameter vector set

$$\mathcal{V} = \{\vec{\alpha}_{\phi_1}, \cdots, \vec{\alpha}_{\phi_N}\}$$

such that the images

$$T[\vec{\alpha}_i](S_{\phi_i}), \ \ 1 \leq i \leq N,$$

are primitive functions of the same form. T is referred to as the *similarization transformation.*

Single Control Parameter Function Form Synthesis Problem

Let F denote an unknown process with a specified control parameter space $\Phi \subset \Re$.

<u>Given</u>

- an observation $\mathcal{O}_{\Phi'}$ of (F, Φ);
- a set of primitive functions $\mathcal{F}$;
- a set of basic operators — transformations $\mathcal{T}$;
- a form fitting accuracy requirement ϵ_F; and
- a maximum dimension of descriptive parameter vector O_{max},

<u>Construct</u> a function form $(\hat{F}, \hat{\Phi})$ and a fitting $\mathcal{M} : \Phi \mapsto \hat{\Phi}$ such that

1. $(\hat{F}, \hat{\Phi})$ is a subset of the search space generated by $\mathcal{F}$ and $\mathcal{T}$;
2. ord $[(\hat{F}, \hat{\Phi})] = \dim(\hat{\Phi}) \leq O_{max}$;
3. the deviation of $(\hat{F}, \hat{\Phi})$ from $\mathcal{O}_{\Phi'}$ is less than ϵ_F, i.e.

$$\mathcal{E}_{\mathcal{M}}[(\hat{F}, \hat{\Phi}), \mathcal{O}_{\Phi'}] < \epsilon_F.$$

This statement can be understood from the implementation point of view as the problem of finding a transformation sequence[6] $\mathcal{T}_\phi$ which is composed of the transformations defined in the set $\mathcal{T}$, and a matching primitive function form $\mathcal{F}_\phi$ from the set $\mathcal{F}$, such that by consequently applying the inverse each transformation in the sequence to the corresponding primitive function sample S_{ϕ_i} results in a functional image that matches with the given observation within a tolerable error level. This approach follows the idea of BACON's multi-variable function form discovery strategy — variable freezing, i.e. hold all but one independent variable constant and find a solution for the subtask then goes to the second variable.

In practice, to find the mapping $\mathcal{M}$ is a very difficult task. FFD simplifies this task by making two simplification assumptions.

Assumption 1 (Primitive Union) *It is assumed that the intermediate goal coincides with the final goal. In other words, if the samples can be unified then there exists a unifying sequence such that the resultant image S_U is a sample of a primitive function.*

Assumption 2 (Simple Descriptive Parameters) *Each descriptive parameter can be accurately represented as a primitive function of the control parameter.*

Apparently, Assumption 1 ensures that the unification could be easily detected and Assumption 2 ensures that the expression of each descriptive parameter could be easily found with a small number of parameters.

To discover a parameterized function form, FFD first acquires a set of observation upon $\{\phi_1, \cdots, \phi_N\}$ — a partitioning of the control parameter ϕ. This is the way FFD arbitrarily divides the original three-variable function form discovery task into a set of subtasks in lower dimension (Refer to the indirect model depicted in Figure 4.2). Two-variable function form discovery processes are then carried out upon each sample S_{ϕ_i}. And the discovered 2-variable function form hypotheses are grouped according to the corresponding identified

[6] The subscript ϕ signifies that the possible descriptive parameters are expressed in terms of functions of the control parameter ϕ

transformation sequences. At last, if certain population of identical transformations and matching primitives are found, the final function form hypothesis will be given based on the assumption of *simple descriptive parameters.* If the found hypothesis applies to all samples, a successful discovery is reported, otherwise, the system will choose from continuing search for new hypothesis or terminating the process and reporting as a failure. The following is an example of using FFD to discover a simple three-variable function form.

<u>An Example of FFD 's Discovery of a Three-variable Function Form</u>

Underlying Function Form : $y = e^{\phi t}$

Sampling : The observation contains five samples corresponding to $\phi_1 = -2$, $\phi_1 = -1$, $\phi_1 = 0$, $\phi_1 = 1$ and $\phi_1 = 2$. Each sample contains 101 uniformly placed partitioning points in the domain $t \in [0.0, 2.0]$.

Discovered Solutions to Each Subtask :

ϕ	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5
Trans.	Λ_+	Λ_+	I	Λ_+	Λ_+
Primitive	$y = -2t$	$y = -t$	$y = 0$	$y = t$	$y = 2t$

Finalization : FFD finds that 4 out of 5 samples can be transformed into a linear function by same transformation Λ_+. Thus Λ_+ is a similarization transformation. Applying this transformation to sample ϕ_3 results in an identical primitive $y = 0 \cdot t$. Therefore an unifying transformation has been confirmed, i.e. $\Lambda_+(\mathcal{O}_\phi)$ which matches with the primitive $y = \alpha t$, where α is the only descriptive parameter. The pairs (ϕ_i, α_i), $i = 1, \cdots, 5$ can be easily fitted to a primitive $\alpha = \phi$. Thus the discovered transformation sequence and primitive are: $(\Lambda_+,\ y = \alpha t)$. The system terminated with a successful discovery.

Although it is a very simple example, we can see the discovery methodology clearly enough. From the practical point of view, the major difficulty of taking this approach is finding the unifying transformation and the mapping from control parameter to descriptive parameters. This is the reason why simplification assumptions are needed.

The method of simplifying a high dimension problem into low dimension by taking into account of one variant at a time is a simple and quite widely used technique. When the variables involved in a problem are not highly coupled, this methodology could be the best choice. For example, if the underlying function form of an unknown process is of form $z = f(x) + g(y)$, there will be no doubt that the variable freezing method is surely the best and simplest method for finding the solution. Moreover, since all the descriptive parameters are identified arbitrarily, in certain application situations, it might be the most effective way to simplify the discovery problem.

FFD , as the first attempt to solve function form discovery problems using the data transformation technique, simplified the unification of solutions of sub-tasks with two simplification assumptions, *Primitive Union* and *Simple Descriptive Parameters.* These assumptions are indeed constraints on how the two independent variables are coupled. It is possible to relax these constraints to a certain degree by upgrading the system with new strategies. However, as a system that takes indirect approach, there are a number of limitations concerning the system's ability. I will analyze the general limitations of the indirect approaches in the following section. Before doing that, let us first summarize the function classes that cannot be handled by the FFD system.

There are four situations under which the current FFD system may fail to solve a three-variable function form discovery problem. They are: *Unsatisfied Simplification Assumption*, *Failure in Finding a Transformation*, *Failure in Verify a Solution* and *Incomplete Language.* For simplicity, we refer a function form under those situations as belonging to `USA-Class`, `FFT-Class`, `FVS-Class` or `ICL-Class` respectively if it cannot be discovered by FFD for one of the corresponding reasons as named above. All function forms that cannot be discovered by FFD fall into these four categories. Let us examine them one by one.

The first class, `USA-Class`, is easy to understand. The system is designed based on certain simplification assumptions, i.e. *Primitive Union* and *Simple Descriptive Parameters*. It is obvious that a function form which does not satisfy one of the assumptions is certainly beyond the system's discovery scope. One example function in this class is $z = e^y \cdot \log x$. The indirect FFD method cannot find the form by either freezing x or y as the control parameter, since using any transformation, the functional relationship between the required descriptive parameter and the chosen control parameter would not be a simple primitive function.

Concerning the second class, `FFT-Class`, some transformations can only be triggered when certain functional features are observed from a given sample data set. For example, the factorization is based on the observation of roots. There are two possible reasons that inhibit the discovery system to find such critical information from an observation corresponding to a certain partitioning scheme, i.e. (1) improper parameter partitioning and (2) infinity of the underlying function. Since all the transformations defined in the transformation set of FFD are fundamental and usually necessary for the system to discover function forms, failure in applying one important transformation will largely reduce the chance for the system to find the solution. I will discuss this issue more in the case studies in next chapter. Moreover, sometimes the observed roots for triggering the factorization are more complicated than we may have expected, for example, more than one root to a single sample is not a rare situation. Grouping the roots obtained from different samples becomes a very challenging task, especially when large error being introduced in the estimation of those roots.

As it has been pointed out that the application of some transformations are subjected to certain constraints. For example, *Inverse* (refers to the table on page 4.2.1) can only be applied to a monotonic curve, whereas *Logarithm* and *Reciprocal* can only be applied to constant sign curve. Such constraints may cause problems for FFD to verify an function form hypothesis made upon some samples. These are the cases pertaining to `FVS-Class`. In some cases, FFD may successfully find a correct two variable function form solution from a population of the samples of an observation data set. However, when it tries to confirm the hypothesis, it may find that it is not a valid solution since the associated transformation sequence is not applicable to some samples. This will cause the system to drop the hypothesis finally and search for a new one or to terminate the task.

The last class, `ICL-Class`, of un-discoverable function forms is due to the FFD's intrinsic capability to discover 2-variable function forms. The function form discoverable to a data transformation based function form discovery system is determined by the transformation set and the primitives available to the system. Due to the large variety of function forms, no system can guarantee to discover all of them from numeric input. In other words, there is not a complete language that can make all functional relationships expressible. FFD employed only five fundamental transformations in its transformation set and three quadratic functions in its primitive function set. Though the performance of the system in carrying out function form discovery is significantly superior to its predecessors, there are still a number of two-variable function forms that cannot be found[65]. As we have already known, FFD bases its parameterized function form discovery on its abilities in discovering two-variable function forms. Hence, there exist many three-variable functions which are not discoverable by FFD simply because FFD cannot handle the subtasks of finding those necessary two-variable function forms.

From the methodology perspective, some of the function form classes summarized above are less critical than others. In general, the class `USA-Class` is the easiest to be changed by replacing the strict constraints with some others. Let us examine the function form example $z = e^y \cdot \log x$ again. Assume that the variable x is viewed as the control parameter. It is easy for the system to find a unified function form description $z = c(x) \cdot e^y$ to describe the given observation, where $c(x)$ is the only descriptive parameter. If a second round of function form discovery is carried out upon the discrete data set $\{(c_i, x_i)\}$, (instead of using the simple descriptive parameter assumption and fitting to a primitive function), it will not be hard to find the correct mapping $c(x) = log(x)$. However, automatically combining the results of subtasks could be a very difficult task. Thus, any implemented system that uses variable freezing technique, more or less, will come with some necessary simplification assumptions.

Overcoming the transformation applicability is one of the major improvements made by the LINUS system to the FFD system. Although it is not an easy extension, yet it is possible to enhance the current FFD system's ability of dealing with three-variable function forms belonging to `FVS-Class` by employing LINUS's discovery strategies. However, on the other hand, the identification of unification or similarization transformations will become a real challenge.

The other two classes, `FFT-Class` and `ICL-Class` are more critical. It is relatively harder to improve the current FFD system to solve these two classes of problems which manifest the shortcoming of the indirect approach.

4.2.2 Direct Vs. Indirect

To solve multi-variable problems, it is usually important to find a way to simplify the problem by reducing the dimension of the problems. The variable freezing method used by FFD is one of the most widely used indirect methods for solving this problem. This method partitions a multi-variable problem into components by holding all relevant variables, but one, constant so that each component is a clearly defined and easy to solve one dimensional problem. By recursively combining the results of low dimension into solutions of higher dimension, the original multi-variable problem is solved recursively. The dimension reduction scheme used in this technique can be viewed as an "*arbitrary dimension reduction scheme*". We say it is *arbitrary* because the reduction takes place "blindly" without considering the specialty of an individual problem to be solved. A direct method, in contrast, usually does not employ any arbitrary dimension reduction scheme. It views the problem as a whole and uses only "*problem-driven dimension reduction schemes*", if any. *Problem-driven* means that a dimension reduction is triggered only if certain evidence is found in the process of solving a specific problem. For multi-variable function form discovery problems, the differences of these two approaches are outlined as below.

1. Indirect approaches sometimes are simpler and more powerful than direct approaches, depending on the properties of the problem to be solved. For example, to solve the discovery problem of family of functions, an indirect method may be the best choice.

2. The major difficulty for an indirect approach is to combine the results of the subtasks in lower dimensions into the solution of the original problem. The major difficulty

to a direct approach is to handle the multi-dimension problem as a whole and extract necessary information to form subtasks and conduct simplification. Solving a multi-dimension problem usually needs two steps: (1) simplifying the problem into subproblems, and (2) combining all the solutions of the corresponding subproblems into the solution to the original problem. Indirect approaches use the simplest schemes to handle the first task but employ complex strategies to handle the second. Direct approaches invest more work on the first task. They rely on the discovered evidence to divide the problem. Hence less work will be needed for the second step. In an indirect approach, the difficulties in combining the solutions of the subtask include:

(a) identifying identical descriptive parameters. When the system includes more sophisticated primitives and data transformations, this will become a more serious problem.

(b) identifying the uniform transformation sequence. There could be several solutions to the same problem, some are accurate and some are less. If each subproblem terminates with a different solution, the system must choose one from a set of different transformation sequences. This may not be successful since some transformations can only be identified based on successful estimates of the associated parameter value. Such transformations include differential transformation and factorization transformation. Moreover, certain transformations can only be applied under condition.

(c) finding an expression to express the descriptive parameter in terms of control parameters. Sometimes the system has to handle large scale errors due to inaccurate estimation of the descriptive parameter from a single sample. Sometimes it has to deal with incomplete observation data set. The missing information could be due to the continuity constraint, the application of data transformations or the specific application problem.

3. The computer resources required for carrying out direct or indirect discovery are significantly different. For an indirect approach, each subtask is related to only one

sample. When conducting a subtask, the system needs to process only a small portion of the observation data set (a single sample). For a direct approach, the system has to process all the observation data throughout the discovery process. Thus both the time and memory space complexities are usually much higher.

4. Since the computer resources are fixed, a direct method has to work with relatively coarse sampling scheme. Thus handling error propagation in a direct system is more critical than in an indirect system.

5. An indirect approach usually does not make use of the "Cross-effect" in solving the discovery problems[7]. Cross-effect can sometimes provide key information to a successful function form discovery. Isolatedly carrying out the subtask largely limits the capabilities of the function form discovery system constructed on the bases of indirect approach.

6. Indirect approaches are unnecessarily sensitive to some secondary factors. The system may be too fragile to the partition scheme. FFD may be confronted with difficulties if it cannot observe all necessary functional patterns from a single sample. BACON, a system that uses variable freezing technique, was reported being sensitive to the order of which variable were put on hold first [20]. FFD shares the same drawback.

7. Direct approach provides more flexibilities for constructing the discovery system. We have wider choices of transformations and primitives.

8. We may suffer particular difficulties when we want to provide an indirect discovery system with certain domain knowledge. It is a common sense that not all domain knowledge are meaningful in the dimension reduced situation. That kind of domain knowledge are sometimes not usable to a direct approach.

[7] In multi-dimension problems, the changes in one dimension affect the system in a different dimension. This is called "Cross-effect". For example, in non-linear theory, shear strains alone demand the application of normal stresses as well as shear stresses.

9. In many application circumstances, holding a variable constant is not practically possible. When the data is calculated by a simulator, the original control parameters can be easily held constant. However, if the data is collected from experiments in a lab, it is sometimes difficult to set the control parameters exactly the same. When a parameter is generated by a process based on some other control parameters, it is sometimes impossible to hold that parameter constant. Moreover, sometimes the experiments through which we collect observation data are not repeatable. It means that we have to deal with inadequate observations with missing data. These application related constraints limit the applicability of indirect systems.

Direct models are more general than indirect models. Although from the theoretical point of view, an indirect model might be better in solving some specific problems, such as the discovery of families of functions, an available direct model is still important for practical reasons we have discussed above. However, to create a direct model system, we will be confronted with great challenge in computational complexity, language design and error control.

To accept the challenge, the FFD-II system was developed. It performs three dimension transformations and recognizes three dimension functional patterns. Hence it demonstrates the application of data transformation based approach as a direct model. From the language design point of view, by taking direct approach, the underlying functional patterns can be revealed by capturing the "cross effects". From a single sample data set we can observe only the regularity in a certain direction. To extract two-dimensional patterns from a set of one-dimensional regularities is very hard. FFD does the extraction by summarizing only the *similarities* among those one-dimensional patterns when it conducts a discovery task. However, there are some equally important relationships among one-dimensional patterns, for example, the differences between adjacent samples. Since a direct method can analyze those more complex relationships utilizing the cross effects in between individual samples, a better performance can be expected. The FFD-II system views the observation data set as a whole, so that when a functional pattern is observed in an area of the functional image, it will enable the system to successfully apply certain rules so as to form a functional hypothesis corresponding to that observed pattern. Moreover, the formation of subtasks in lower dimension is fully automatic and only triggered by those already recognized patterns during the discovery process. The final discovery can be made once the dominant solution is found. The way to combine the solutions of subordinate sub-tasks with the solution of the dominant sub-task is clearly specified by the output of the system.

The challenge of computational complexity and error propagation control are closely related. To attain the same fine step observation data set, a three-variable functional image will contain $3N^2$ real numbers, compared with only $2N$ for a two-variable functional image, where N is the number of observation data points. During the search, each search node is associated with a transformed functional image. Thus the computer memory space complexity is much higher for carrying out the search in a direct model than in an indirect model. Apparently, the time complexity is also significantly increased in a direct model since more data will be processed in transforming from one state into another and in performing functional pattern recognition. Moreover, with limited computer resources, we are restricted to use only relatively low order approximation tools in a direct model system. Considering polynomial fitting as an example, to fit a curve to the second order polynomial needs only three sample points, to the third order only four sample points are necessary. To fit a surface using polynomial fitting, at least six sample points are necessary to get a second order polynomial fitting and ten sample points must be used to get a third order polynomial. If the available computer resources are the same for carrying out the search, the direct model system will have to deal with a relatively small sampling size and poor approximation accuracies. Hence, selecting proper numeric tools and preventing the propagated errors from exploding are two crucial issues in the design of FFD-II.

4.3 An Overview of the System

diagram and major components Up to now, I have discussed the superiority of data transformation approach as a general discovery model, the benefit of taking the direct approach, as well as the importance of controlling the computational errors in a direct multi-variable function form discovery system. Now, I will introduce the design of the FFD-II system. FFD-II is a data transformation based direct function form discovery system with adaptive error control. The system finds function descriptions in the language $\tilde{\mathcal{L}}$ to describe an observation data set with three variables. The key idea of data transformation based function form discovery is depicted in Figure 3.1. Its details could be found in Section 2.1 and 2.4. The function form description language $\tilde{\mathcal{L}}$ has been defined in Section 3.3.3. Direct multi-variable function form discovery model has been shown in Figure 4.1. And the adaptive error control will be introduced soon in this chapter.

Here, I will follow the common practice of first giving a diagram that overviews the architecture of the entire system and then describing the system components one by one. It is beneficial to do so for the purpose of clarifying the design and reserving an easy access to the system for future extensions. When there is a need of more detailed specifications concerning the numeric computations or implementations, a separate section follows.

The algorithmic architecture of the FFD-II system is illustrated in Figure 4.3. In the figure, each box represents a major module of the system. Dashed box is used to represent a group of modules that join together to achieve a major function. The arrows represent the flow of data or function call with passing parameters. Main routines are organized accordingly. I will summarize the system according to groups of modules.

There are four major groups of modules in the system. They are "*Error Control*", "*Data Selection*", "*Search Engine*" and "*Post Processing*". Descriptions are as following.

Data Selection: Before the execution of a discovery task, the original observation data set is stored in a formated data file. The precision of the observation is given as an input. The program starts with the *Data Selection* module. The module selects a subset of the observation data set from the observation data collection, an evenly distributed mesh grid of the size 101×101 sample triples[8] to initialize the search tree.

[8] It is assumed that the user has the full control of the experiments from that the original observation data are collected. If the observation data collection does not contain enough sample data points, the system will ask the user to provide new observation data. However, the user can also force the system to carry out the discovery upon whatever is provided.

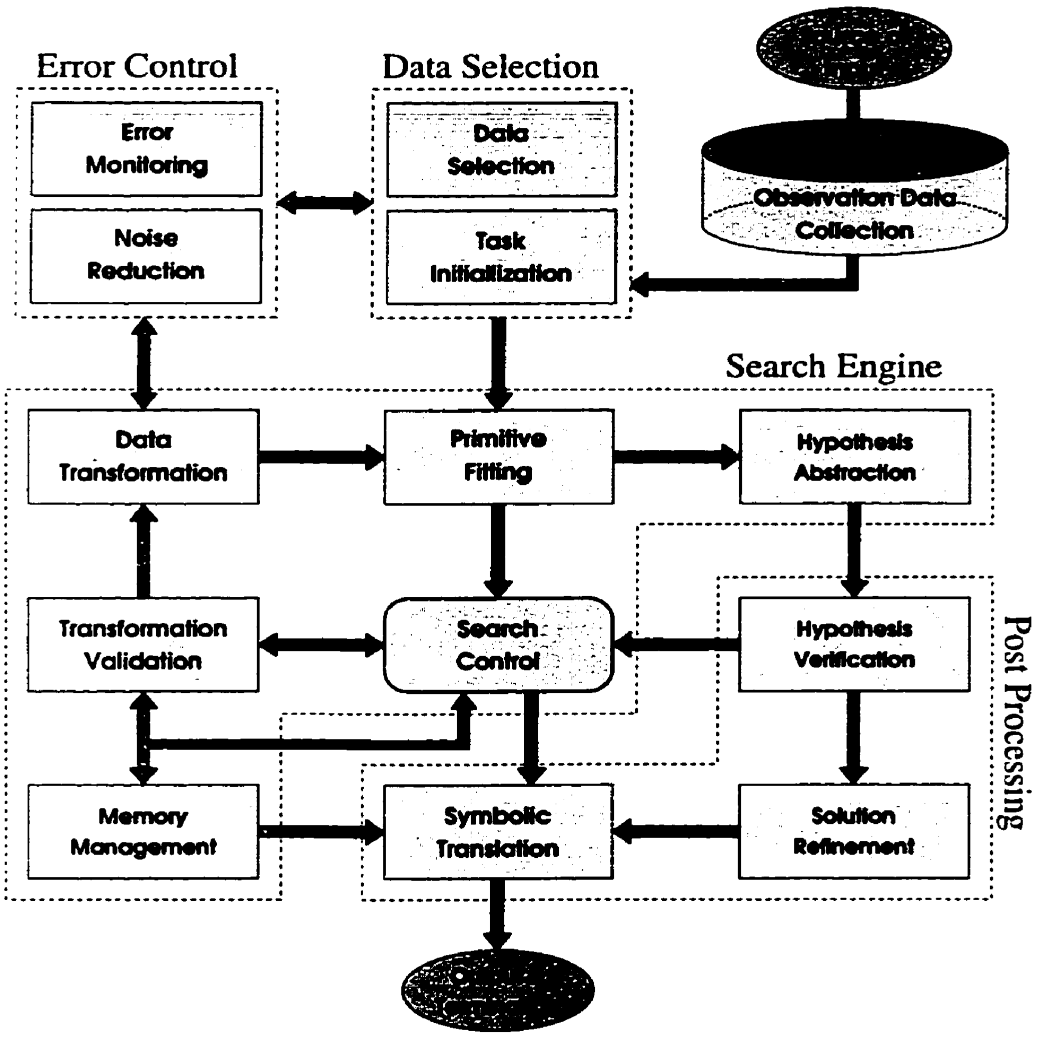


Figure 4.3: An Overview of the FFD-II Discovery System

Besides initializing the root node with the functional image, the *Task Initialization* module also generates a number of child nodes attached to the root. However, when a child node is first generated, it contains only the specification of the transformation that might be applied to obtain the associated functional image from the functional image of the parent node. When the node is selected to be investigated and the transformation is confirmed as valid, i.e. applicable and non-redundant, the system computes the functional image and updates the corresponding attribute with it. If the corresponding transformation is not applicable, the node will be a *Dead End Node* of the search tree. An "OPEN" list is constructed at the stage of task initialization for carrying out the best first search. All generated child nodes are added to the list. During the execution of the program, the Data Selection module is also called by the *Noise Removing* module. It monitors the data selection scheme entry of a node and selects an adequate data set for the noise removing process upon requests.

Search Engine: This is the central part of the system. In this part, the *Data Transformation* module is a set of numeric implementations of the data transformations defined in $\tilde{\mathcal{L}}$. The *Primitive Fitting* module recognizes primitive patterns. These two modules are the discovery tools of the system. The *Transformation Validation* module checks the validity of a transformation that is going to be applied to a functional image associated with a specific node, so that redundant transformations and non-applicable transformations are prevented[9]. The *Hypothesis Abstraction* module abstracts a functional description hypothesis. The function of the *Resource Management* module is to ensure that sufficient computer resources are available to continue the search. It dynamically allocates memory for new nodes and releases the memory allocated to dead end nodes. A *Dead End Node* is a node that does not have any valid child node because either maximum search depth has been reached or the associated transformation is invalid. If the module determines that computer resources are running out, it calls the *Symbolic Translation* module to terminate the job and generate

[9] Redundant transformation macros have been discussed in Section 3.3.4. The applicability conditions were described along with the definitions of each transformation class in Section 3.3.1.

output. The *Search Control* module selects the node to be expanded from the OPEN list according to the "Best-First" rule. The search heuristics for carrying out heuristic search will be introduced in Section 4.4.2. If all possibilities have run out, or in other words, the system has completed an exhaustive search, *Symbolic Translation* module is called to terminate the discovery job and generate the corresponding output. This could happen for the following reasons

- The incompleteness of the description language $\tilde{\mathcal{L}}$,
- The noisy input and computational errors,
- Inadequate control parameter settings — such parameters include maximum rank (or depth) and the tolerable matching error[10].

If it is not for the first reason, we can re-configure the system or improve the quality of the input data and perform a new discovery. Putting all together, the pseudo code of the core search algorithm is described in Figure 4.4.

Post **Processing:** Post processing consists of three modules. The *Hypothesis Verification* module verifies an abstracted function form hypothesis through comparing the original functional image with the functional image generated by consequently applying the inverse transformations to the corresponding fitting primitive. If the deviation[11] between these two images exceeds a tolerable level (a parameter input to the system), the hypothesis is rejected and the Search Control module invokes the search for new hypotheses. Otherwise, the function form hypothesis is confirmed as a discovered solution and the solution is sent to the Symbolic Translation module. The *Symbolic Translation* is an interface that prepares the output of the discovery result. It receives a solution from the Hypothesis Verification module, or particular parameter values from the Resource Monitor or the Search Control module. In the first case, a discovered function form is obtained.In the latter cases, the module will indicate the reason the system failed to find a solution (either resource running out or the under-

[10] Refer to Section 3.2.2 on page 58.

[11] The deviation is referred to as the *Matching Error* or *Verification Error* throughout this thesis.

```
PROCEDURE TREE-SEARCH

  INPUT:   Initialized search tree which has a root node and its
             associated child nodes.

           Initialized search tree node list OPEN with a set of tree
             nodes.

  OUTPUT:  An expanded search tree and a node where the search
             terminated.

  1.  if OPEN = null, terminate the search.
  2.  Remove one node $N$ with the lowest cost from OPEN.
  3.  Check the validity of $N$.
        if $N$ is invalid
          then label $N$ as Dead, and repeat from step 1.
  4.  Compute new functional image according to the
      transformation and the parent image.
  5.  Perform primitive matching upon the current node.
  6.    if there is an acceptable fitting.
          then Suspend the search.  Abstract and verify
               the hypothesis based on the current node
               fitting.  If the hypothesis is rejected by
               the verification process, continue.
               Otherwise the discovery task is ended
               successfully.
  7.  Generate a set of child nodes under current node $N$
      according to the available transformation classes in
      system's transformation tool-box.
  8.  Add the new generated nodes into OPEN, and repeat from
      step 1.
```

Figure 4.4: The Best-First Search Algorithm

lying function form of the observation data set being undescribable within the given error tolerance). A best matching form along with matching error will be output by checking the system's record of all fittings that have been tried.

Error Control: This part of the system adaptively controls the error propagation of repeatedly applying data transformations. The *Error Monitor* module measures the quality of a functional image by its expected error level and roughness value. If either of these exceeds a corresponding preset threshold, the *Noise Reduction* module will be invoked to improve the quality of the image.

1. If it is the expected error level that exceeds the threshold, an image *Refinement* is required. Noise Reduction first consult the data selection module to see if there is any unused original observation data that could help to improve the quality of the current functional image. If so, image refinement will be performed based on those unused observation data and the transformation history of the current node. If there is no more original observation data that could be used to improve the quality of the current image, the Noise Reduction module will inform the search engine to adjust the heuristic value for those nodes generated under this node, so that low priorities will be given to the investigations under the node whose underlying functional pattern has been distorted by the input noise or computational error so badly that it could not be recovered by the system.
2. If it is the image roughness[12] exceeds the threshold, a polynomial smoothing scheme will be used to smooth the image. Recall that one of the basic assumptions is that the underlying function form of any functional image is a class C^∞ function in the observation domain (page 59). As such, the pattern of an exceedingly rough surface must not be the true functional pattern of the underlying function form but a pattern of noise or computational error that should be removed. Similar to the image refinement, the availability of additional observation sample points, in the form of a high resolution image are required for carrying

[12] The measurement of roughness of a surface will be defined later in this chapter.

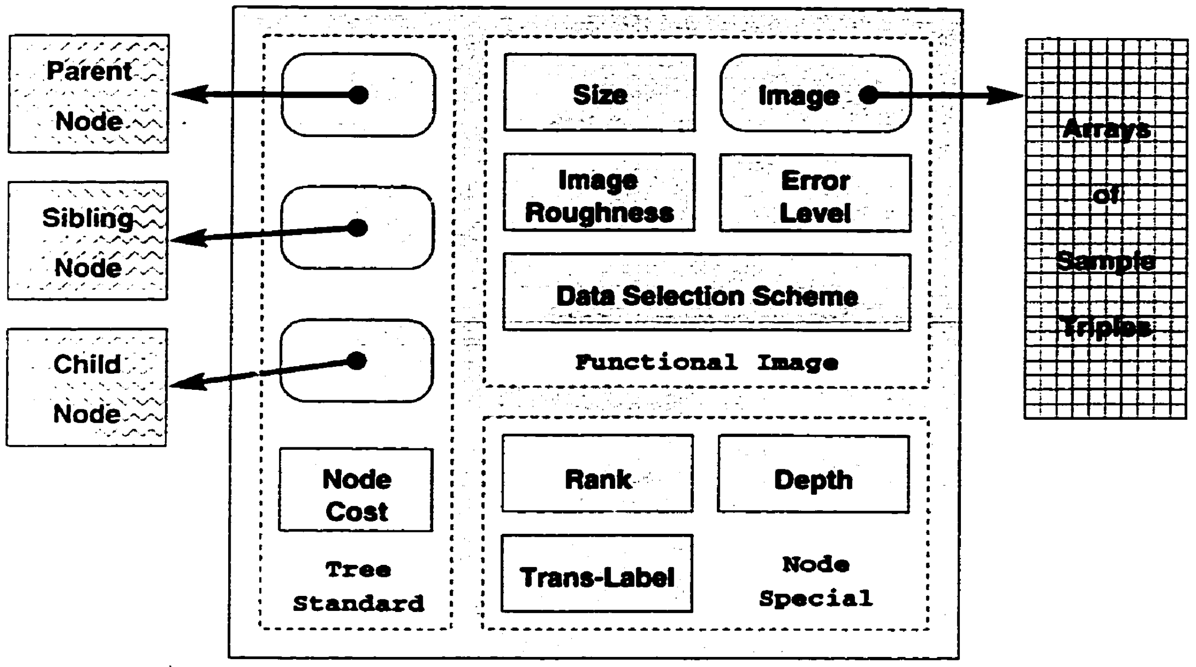


Figure 4.5: The Data Structure of A Searching Node

out the image smoothing process.

The Error Monitoring module is called by both the Task Initialization module and the Data Transformation module for controlling the error adaptively. Corresponding fields of the search node data structure will be updated accordingly.

The modules introduced above are implemented with a data structure shown in Figure 4.5. There are three groups of information in the structure. The *Standard Tree* group represents the standard tree structure, the *Functional Image* group contains the primary node content that describes the associated functional image of the search node. In this group, three entities are designed for controlling the noise. They are *Image Roughness*, *Error Level* and *Data Selection Scheme*. The *Node Special* group, provides information concerning the transformation history. The *Trans-Label* entity specifies the last transformation that has been applied to generate the current node. It also distinguishes the current node as unexplored, explored or dead end node with different values.

4.4 Numerical Recipes

FFD-II is designed to find function forms from numeric data. Numeric computations are involved in the discovery process. In this section, I shall specify the numeric tools that have been used to achieve the goal. Since the major concern of this research is the discovery methodology, only simple numeric methods with sufficient accuracy have been chosen. Polynomial least-squares fitting is a widely used simple method. It is flexible and easy to use. It also has relatively simple analytic properties. As such, it will be utilized to solve several numeric computation problems in the implementation of the FFD-II system.

4.4.1 Numeric Data Transformations

Table 3.6 on page 76 lists the transformations employed by FFD-II. Among them, most algebraic transformations are easy to compute. The formulas to compute T_{LOG}, T_{LOG}^{-1} are given by Equation (3.42) and (3.43). And similarly, T_{REC}, T_{REC}^{-1}, T_{INV}, T_{INV}^{-1}, T_{VEX} and T_{VEX}^{-1} can be easily implemented using the equations used to define them. Thus I shall not give more details concerning their computations here. The implementations of the remaining two transformation classes are not that straightforward. I will describe them one by one in this part.

Numeric Differentiation and Integration

Differential transformation T_{DIF} and its inverse T_{DIF}^{-1} are the most challenging transformations to be numerically implemented. The classic methods of computing derivatives are based on certain difference schemes. However, this method only works well when there is no noise on the sample data set and the partitioning of the observation domain is uniformly distributed in a rectangular mesh grid. Unfortunately, the transformed functional image does not satisfy such a constraint. Although we can assume that the original input observation data could be in whatever form or distribution we like (sufficient observation assumption, page 59), the distribution of a transformed functional image may not be distributed as expected. It could be distorted by the transformations that have been applied. In other words, the original uniformly distributed observation mesh grid may no long be uniformly distributed in a planar region formed by the two axes corresponding to the two independent variables. Moreover, the data we used to compute the partial derivatives may contain input noise and computational errors. Thus the traditional difference method could not be used to compute T_{DIF} for FFD-II.

The computation of T_{DIF} in FFD-II is simple and intuitive. The method is based on polynomial surface fitting. It is described as the following. Let $\mathcal{O}_{(u,v,w)}$ be an observation data set and $w = f(u,v)$ be the underlying function. Let $P_0 = (u_0,\, v_0,\, w_0) \in \mathcal{O}_{(u,v,w)}$ be a sample point of the image. We need to compute the partial derivative $\left.\frac{\partial f}{\partial u}\right|_{u=u_0,v=v_0}$. Let

$$S_k = \{\, (u_0,\, v_0,\, w_0),\, (u_1,\, v_1,\, w_1),\, \cdots,\, (u_k,\, v_k,\, w_k) \,\} \in \mathcal{O}_{(u,v,w)} \tag{4.1}$$

be a set of distinct sample points such that

$$\forall\, p(u,v,w) \in \mathcal{O}_{(u,v,w)},\; p \notin S_k,\; \Longrightarrow \tag{4.2}$$
$$\left\{ \begin{array}{r} \sqrt{(u-u_0)^2 + (v-v_0)^2} \geq \sqrt{(u_i-u_0)^2 + (v_i-v_0)^2} \\ \text{for all } \; i = 1, \cdots, k \end{array} \right.$$

S_k defines the set of k nearest neighbors of the planar point (u_0, v_0). Now let $w = g(u, v, \vec{\varphi})$ be a parameterized continuous function (a function template), where $\vec{\varphi}$ is the parameter vector. The expression of fitting k nearest neighbors with template function g is a specification $\vec{\phi}$ of the vector $\vec{\varphi}$ such that

$$\sum_{(u_i,v_i,w_i)\in S_k} \left| w_i - f(u_i, v_i, \vec{\varphi}) \right|^2 \tag{4.3}$$

is minimized. If we write the resultant fitting function as $w = \bar{g}(u,v)$, then the estimated

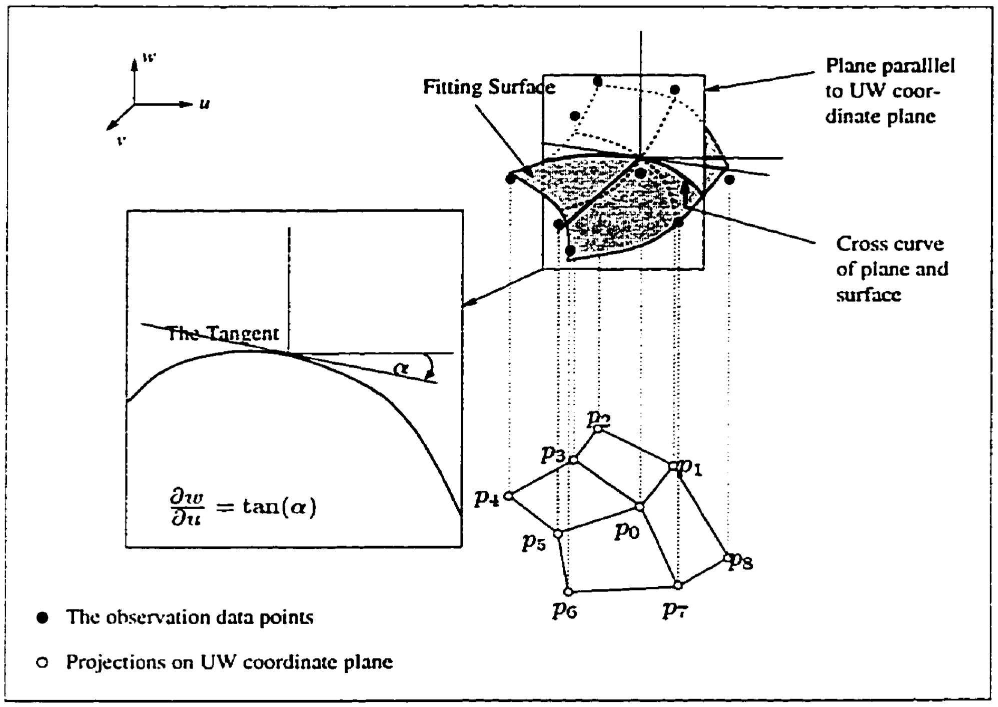

w
u
v
Fitting Surface
Plane paralllel to UW coordinate plane
Cross curve of plane and surface
The Tangent
α
$\frac{\partial w}{\partial u} = \tan(\alpha)$
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
● The observation data points
○ Projections on UW coordinate plane

Figure 4.6: Approximation of Partial Derivatives by Surface Fitting

partial derivative value of the underlying function f at point (u_0, v_0) is

$$\left.\frac{\partial f}{\partial u}\right|_{(u_0,v_0)} \approx \frac{\partial\, \bar{g}(u_0, v_0)}{\partial u} \tag{4.4}$$

Figure 4.6 illustrates an eight-nearest-neighbor approximation scheme. In the implementation of FFD-II the complete second order polynomial function

$$g(u,v) = a_{11} \cdot u^2 + a_{12} \cdot uv + a_{22} \cdot v^2 + a_1 \cdot u + a_2 \cdot v + a_0 \tag{4.5}$$

is used as the template function and eight nearest neighbors are taken into account as a local fitting point set (adjustable).

Having solved the problem of numeric computation of transformation T_{DIF}, I will then describe the method to conduct the inverse transformation T_{DIF}^{-1} numerically. First, let us see what the problem is.

We are given two functional images $\mathcal{O}$ and $\hat{\mathcal{O}}$

$$\begin{cases} \mathcal{O} = \left\{(u_i, v_i, w_i) \,\middle|\, i = 1, \cdots, N\right\} \\ \hat{\mathcal{O}} = \left\{(u_i, v_i, \hat{w}_i) \,\middle|\, i = 1, \cdots, N\right\} \end{cases} \tag{4.6}$$

and a hypothetical relationship between these two images

$$T_{\mathrm{DIF}}(\mathcal{O}) \equiv \hat{\mathcal{O}} \tag{4.7}$$

We want to compute a functional image

$$\overline{\mathcal{O}} = \left\{(u_i, v_i, \bar{w}_i) \,\middle|\, i = 1, \cdots, N\right\} \tag{4.8}$$

from $\mathcal{O}$ and $\hat{\mathcal{O}}$, such that if the underlying functions of image $\mathcal{O}$, $\overline{\mathcal{O}}$ and $\hat{\mathcal{O}}$ are $w = f(u,v)$, $w = \bar{f}(u,v)$ and $w = \hat{g}(u,v)$ respectively, then

$$\frac{\partial \bar{f}}{\partial u} \approx \hat{g}(u,v). \tag{4.9}$$

Furthermore, if hypothetical relationship (4.7) holds, we need

$$\bar{f}(u,v) \approx f(u,v) \tag{4.10}$$

To solve this problem, we first reduce the dimension of the problem with a data grouping scheme that groups a planar point set into subsets each of which represents a planar curve. To do the grouping, we notice that all functional images are transformed images of the original functional image, $\mathcal{O}_I$, which is partitioned into a rectangular observation mesh grid (Figure 4.7). That means $\mathcal{O}$ can be rewritten as

$$\mathcal{O} = \left\{(u_{i,j}\,,\ v_{i,j}\,,\ w_{i,j}) \,\middle|\, i = 1, \cdots, \mathrm{N}x, j = 1, \cdots, \mathrm{N}y\right\},$$

and so do images $\hat{\mathcal{O}}$ and $\overline{\mathcal{O}}$. Therefore, we group the sample point indices of each image into N_x or N_y groups according to the result of coordinates comparison as shown in

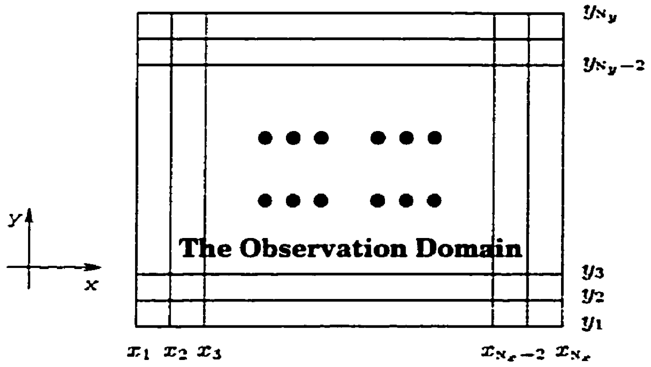


Figure 4.7: Partitioning of the Input

Equation (4.11).

$$
\begin{aligned}
S_k &= \{(1,k),\,(2,k),\,\cdots,\,(\mathrm{N}x,k)\}\,,\quad \text{for } k = 1,2,\cdots,\mathrm{N}y\,,\\
&\qquad\qquad \text{if } |u_{(N_x,1)} - u_{(1,1)}| \geq |v_{(N_x,1)} - v_{(1,1)}|\\
S_k &= \{(k,1),\,(k,2),\,\cdots,\,(k,\mathrm{N}y)\}\,,\quad \text{for } k = 1,2,\cdots,\mathrm{N}x\,,\\
&\qquad\qquad \text{otherwise.}
\end{aligned}
\tag{4.11}
$$

Thus each planar point set corresponding to an integer k, $\left\{(u_{i,j},\ v_{i,j}\}\,\middle|(i,j)\in S_k\right\}$, represents a planar curve. We now define the image $\overline{\mathcal{O}}$ as the image of a function $w = \bar{f}(u,v)$ such that

$$
\begin{cases}
\bar{f}'_u = \hat{g}(u,v)\\
\bar{f}'_v = f'_v.
\end{cases}
\tag{4.12}
$$

As such, the image $\overline{\mathcal{O}}$ can be easily computed by classic numeric integration along each curve indexed by the corresponding set S_k [13].

In the process of verification [14], FFD-II views image $\overline{\mathcal{O}}$ as the reversed image of $\widehat{\mathcal{O}}$ when

[13] The numeric values of the derivatives along each curve is computed based on the partial differentials $\partial w/\partial u$ and $\partial w/\partial v$. In the implementation of FFD-II, $\partial w/\partial u$ is approximated with the sample point values of image $\widehat{\mathcal{O}}$, and $\partial w/\partial v$ is approximated by computing the corresponding partial derivatives of image $\mathcal{O}$. The initial values for numeric integration are calculated from the image $\mathcal{O}$ in a way such that the deviations between $\mathcal{O}$ and $\widehat{\mathcal{O}}$ is minimized.

[14] The verification problem is as such: "*Given* an original input functional image data set $\mathcal{O}_I$, a function

the last hypothetical transformation applied to generate $\hat{\mathcal{O}}$ is T_{DIF}. This is a compromise between simple computation and the satisfaction of Equation (4.9). It is obvious that Equation (4.9) is satisfied only if Equation (4.7) is satisfied. As a necessary condition, this method meets the needs of hypothesis verification.

Linear Factors

FFD-II employs the factorization transformation described by Equation (3.48). The factor $(u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C)$ is detected from the corresponding functional image. The factorization transformation is only applied when it is hypothesized that the underlying function of a functional image contains a linear factor, i.e.

$$f(u_1, u_2) = g(u_1, u_2) \cdot (u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C). \tag{4.13}$$

Therefore, we must have evidence that shows the existence of the factor. FFD-II extracts the hypothetical factor through detecting lines in the contour image of $u_d = 0$, called "*0-contour*" image. The algorithm is designed based on three important observations:

1. There is a factor $u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C$ only if we can observe a correspondent line

$$u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C = 0 \tag{4.14}$$

   in the 0-contour image when the observation domain is properly placed.

2. The observation domains of all functional images, original or generated, are simple connected planar regions since the original observation domain is a simple connected planar region and all the applied transformations are one-to-one continuous.

3. If $D \in \Re^2$ is a simple connected planar region within the observation domain, there exists a factor $u_1 \cdot \cos\theta + u_2 \cdot \sin\theta + C$ and a planar point $p(\bar{u}_1, \bar{u}_2) \in D$ that is on

form description $\mathcal{D}_{\mathcal{C}} = (D_T, D_P) \in \mathcal{L}$, and a sequence of consequently transformed images corresponding to each transformation in sequence D_T. ***Find Out*** how well $\mathcal{D}_{\mathcal{C}}$ matches with $\mathcal{O}$ by numeric computing the deviation between the two image $\mathcal{O}$ and $D_T^{-1}(D_P)$".

the contour line, i.e.

$$\bar{u}_1 \cdot \cos\theta + \bar{u}_2 \cdot \sin\theta + C = 0,$$

then the line must cross the boundary of D. Furthermore, if point p is not on the boundary of D, the line will cross the boundary of D at least twice.

A brief description of the algorithm is as follows.

Step 1: Iterate through the boundary of the observation domain, examine the value of the dependent variable on the boundary sample point and compare the signs of the values of each adjacent point pair. A zero value contour points is found directly from the sample point value (if the function value is zero) or by a linear interpolation (if the two adjacent points have different signs). Step 2 and 3 are carried out when such a point is found.

Step 2: Starting from a point found in step 1, trace the 0-contour point into the observation domain to form a tree presentation of the 0-contour curves. Figure 4.8(a) illustrates an example of the tracking, where the arrows show the tracking order. In the figure, there are two 0-contour curves of the underlying function shown as light dotted curves, a straight line and an ellipse. Point p_1 is the starting point on the boundary obtained from step 1. The shaded quadrilaterals are the observation cells[15] in the coordinate plane $u_d = 0$. New 0-contour points are identified through cell by cell sign examinations and interpolations. The key idea of the tracing process is that any 0-contour line will not end in an inner cell.

Step 3: Split the found 0-contour tree into simple curve pieces and fit each piece to a line. If the fitting is acceptable, a straight line equation corresponding to a curve piece is identified. Otherwise, the curve piece is discarded. Figure 4.8(b), (c) and (d) show the groups of split 0-contour points and the fitting results. Only (c) is an acceptable fitting.

[15] An observation cell is a quadrilateral whose four corner points are $(u_{1(i,j)}, u_{2(i,j)})$, $(u_{1(i+1,j)}, u_{2(i+1,j)})$, $(u_{1(i+1,j+1)}, u_{2(i+1,j+1)})$ and $(u_{1(i,j+1)}, u_{2(i,j+1)})$.

Step 4: If the iteration of step 1 has not been completed, continue from step 1. This enables the algorithm to find multi-contour-lines.

Apparently, this simple method is relatively sensitive to noise. When noise level exceeds a limit, the 0-contour tracking procedure will not be successfully completed. However, the current research focuses more on methodology issues, only simplest numeric recipes that do not require too much computer resources are chosen. Identifying a set of complex curves from a set of planar points of its own is an interesting research topic. FFD-II employs only linear factor to demonstrate the ideal of factorization transformations. Other factorization transformation classes may be considered according to the domain knowledge of the application.

4.4.2 Primitive Fitting and Search Heuristics

Primitive Fitting and The Error

Recall that we have defined two types of primitives, i.e. functional primitive and compositional primitive. Fitting a functional image to a functional primitive is simply a linear fitting of least-squares. Let

$$\mathcal{O}_V = \left\{ \left(v_1^{(i,j)}, v_2^{(i,j)}, v_d^{(i,j)}\right) \,\middle|\, i = 1, \cdots, Nx;\ j = 1, \cdots, Ny \right\}$$

be a functional image data set and $v_d = P(v_1, v_2)$ be the functional primitive fitting resultant function of $\mathcal{O}_V$. The fitting error is the mean-square distance between surfaces $v_d = P(v_1, v_2)$ and $\mathcal{O}_V$ as described below.

Let $p_0 = \left(v_1^{(i,j)}, v_2^{(i,j)}, v_d^{(i,j)}\right)$ be any sample point of the observation image data set $\mathcal{O}_V$. The normal vector of the fitting surface at point $\left(v_1^{(i,j)}, v_2^{(i,j)}\right)$ is

$$\vec{n}_{ij} = \left(P'_{v_1}, P'_{v_2}, -1\right) \Big/ \sqrt{\left(P'_{v_1}\right)^2 + \left(P'_{v_2}\right)^2 + 1}\ \Bigg|_{v_1 = v_1^{(i,j)},\ v_2 = v_2^{(i,j)}} \tag{4.15}$$

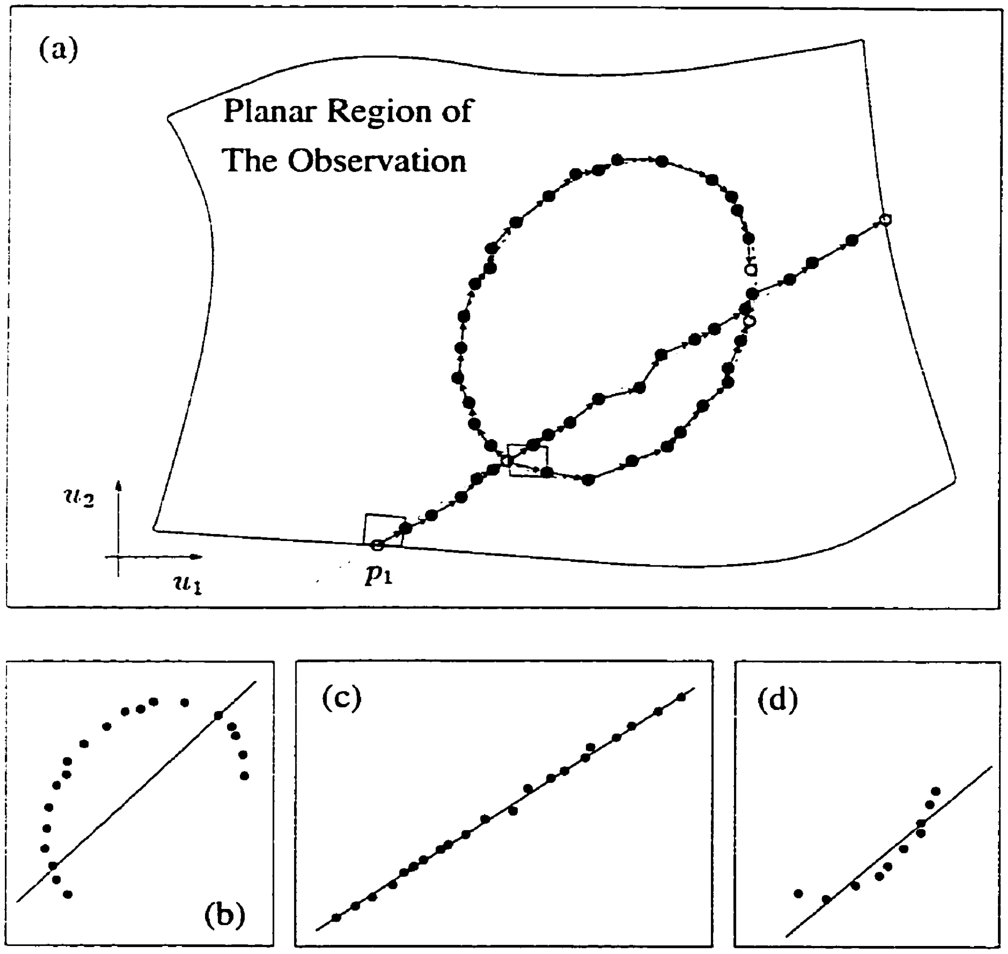


Figure 4.8: Extracting Straight Line in A Contour Image

and the error distance vector is defined as

$$\begin{aligned}\vec{\varepsilon}_{ij} &= \left(v_1^{(i,j)}, v_2^{(i,j)}, v_d^{(i,j)}\right) - \left(v_1^{(i,j)}, v_2^{(i,j)}, P\left(v_1^{(i-1,j-1)}, v_2^{(i-1,j-1)}\right)\right) \\ &= \left(0\,, 0\,, v_d^{(i,j)} - P\left(v_1^{(i,j)}, v_2^{(i,j)}\right)\right).\end{aligned} \tag{4.16}$$

The deviation at point (i,j) is therefore defined as the dot product of the two vectors

$$d_{ij} = |\, \vec{\varepsilon}_{ij} \cdot \vec{n}_{ij} \,| \tag{4.17}$$

and the mean-square distance between the fitting surface and the image surface is

$$E_f = \sqrt{\sum_{i,j} d_{ij}^2 \Big/ Nx \cdot Ny} \;. \tag{4.18}$$

E_f is viewed as both the primitive function fitting error and the error between the initial observation image and the verification image which is generated by numerically inverting the transformation sequence starting from the hypothetical primitive pattern.

The recognition of the primitive pattern of linear compositional primitive is achieved by a multi-line fitting scheme. Let us first give the problem statement.

Linear Composition Component Discovery Problem

Given : a functional image observation data set

$$\mathcal{O}_V = \left\{ \left(v_1^i\,, v_2^i\,, v_d^i\right) \middle| \, i = 1, \cdots, N \right\}. \tag{4.19}$$

Find : a control parameter θ so that the data set generated by

$$\hat{\mathcal{O}}_V = \left\{ \left(t^i, v_d^i\right) \middle| \begin{array}{c} t^i = v_1^i \cdot \cos\theta + v_2^i \cdot \sin\theta \\ i = 1, \cdots, N \end{array} \right\} \tag{4.20}$$

represents a smooth one dimensional function $v_d = g(t)$.

A weighted multi-line least square fitting is used to solve this problem. The algorithm includes three main steps.

Step 1: Find the range of the observation $\mathcal{O}_v$ and partition the range into N_D adjacent close intervals

$$\begin{aligned} S_i &= [\, v_{\min} + (i-1)\cdot\Delta \,,\ v_{\min} + i\cdot\Delta \,] \\ &\quad \text{for} \quad i = 1, \cdots, N_D \,; \ \text{ and } \Delta = \frac{v_{\max} - v_{\min}}{N_D} . \end{aligned} \tag{4.21}$$

where

$$v_{\min} = \min_{(v_1^i,\, v_2^i,\, v_d^i) \in \mathcal{O}_v} \left\{ v_d^i \right\} \,, \quad v_{\max} = \max_{(v_1^i,\, v_2^i,\, v_d^i) \in \mathcal{O}_v} \left\{ v_d^i \right\},$$

and number N_D is set to be $\sqrt{N}$ for $N = |\mathcal{O}_v|$.

Step 2: Segment the sample points into N_D groups

$$G_k = \left\{ (v_1^i,\ v_2^i,\ v_d^i) \;\middle|\; \begin{array}{c} \text{for } v_d^i \in S_k \,; \text{ and} \\ (v_1^i,\ v_2^i,\ v_d^i) \in \mathcal{O}_v \end{array} \right\} \tag{4.22}$$

$$(\text{ for } k = 1, \cdots, N_D \,).$$

Note that the groups may not be disjoint.

Step 3: Jointly fit all sample points in each group G_k to a common format straight line $v_1 \cos\theta + v_2 \sin\theta - C_k = 0$, where θ and C_k (for $k = 1, \cdots, N_D$) are the fitting parameters. To reduce the chance of the fitting result undesirably influenced by the segmentation scheme and the distribution of the sample points, a weight function is defined as

$$w_i = w\left(v_1^i,\ v_2^i,\ v_d^i \right) = \left(\delta + (1 - 2\frac{|v_i - c_j|}{\Delta}) \right)^2 \ , \tag{4.23}$$

where (1) it is assumed that the ith sample point is in group G_j ; (2) c_j is the center of the interval S_j and Δ is the dimension of the interval; and (3) δ , a small positive real number, is the offset that is used to adjust the shape of the weight function.

The fitting problem is then a classic minimization problem, i.e. minimizing the objective function

$$F(\theta, C_1, \cdots, C_k) = \sum_{k=1}^{N_D} \sum_{(v_1^j, v_2^j, v_d^j) \in G_k} \left(w_j \cdot (v_1^j \cos\theta + v_2^j \sin\theta - C_k) \right)^2 . \tag{4.24}$$

Clearly, it is a linear problem that could be easily solved.

Once the linear component is obtained, the fitting error is calculated based on the measurement of the smoothness of two-parameter functional image represented by Equation (4.20). We assume that the sample points in the set $\widehat{\mathcal{O}}_V = \{(t^i, v_d^i)\}$ are sorted according to their t values, where $t^i = v_1^i \cos\theta + v_2^i \sin\theta$. The computing scheme is described as the following.

Let (t^i, v_d^i) , $1 < i < N$ be any point in the set $\widehat{\mathcal{O}}_V$, define the δ–neighborhood point set S_i as:

$$S_i = \left\{ \left(t^j, v_d^j\right) \;\middle|\; (t^j, v_d^j) \in \widehat{\mathcal{O}}_d , \quad \text{and } |t^j - t^i| \le \delta \right\} , \tag{4.25}$$

where δ is a small positive real number[16]. Since $\widehat{\mathcal{O}}_V$ is sorted according to the corresponding t values, we can assume that there exist integers k_1 and k_2 , $1 \le k_1 \le i \le k_2 \le N$, such that

$$S_i = \left\{ (t^{k_1}, v_d^{k_1}), (t^{k_1+1}, v_d^{k_1+1}), \cdots , (t^{k_2}, v_d^{k_2}) \right\} . \tag{4.26}$$

Now define a line that crosses points $(t^{k_1}, v_d^{k_1})$ and $(t^{k_2}, v_d^{k_2})$ as:

$$\begin{cases} Av_d + Bt + C = 0 \\ \quad A = t^{k_2} - t^{k_1} \ , \ B = v_d^{k_1} - v_d^{k_2} \ , \ C = t^{k_1} \cdot v_d^{k_2} - t^{k_2} \cdot v_d^{k_1} \ . \end{cases} \tag{4.27}$$

The maximum and minimum deviation values are defined as

$$\begin{cases} e_{\max} = \max\limits_{k_1 < j < k_2} \left\{ \dfrac{Av_d^j + Bt^j + C}{\sqrt{A^2 + B^2}} \right\} \\ e_{\min} = \min\limits_{k_1 < j < k_2} \left\{ \dfrac{Av_d^j + Bt_j + C}{\sqrt{A^2 + B^2}} \right\} \ . \end{cases} \tag{4.28}$$

[16] In the implementation of FFD-II, $\delta = \frac{|t^N - t_1|}{2\sqrt{N}}$.

Thus the fitting error and the curve segment length at the $i^{\underline{\text{th}}}$ point are given as

$$\delta_i = \begin{cases} \max\left\{v_d^j|_{j=k_1}^{k_2}\right\} - \min\left\{v_d^j|_{j=k_1}^{k_2}\right\}, & \text{if } A \approx 0 \\ \max(|e_{\max}|, |e_{\min}|, |e_{\max} - e_{\min}|), & \text{otherwise} \end{cases} \tag{4.29}$$

$$s_i = \sqrt{(v_d^{k_2} - v_d^{k_1})^2 + (t^{k_1} - t^{k_2})^2}. \tag{4.30}$$

Finally, the compositional primitive fitting error is defined as

$$E_c = \frac{\sum_{i=3}^{N-2} \delta_i}{\sum_{i=3}^{N-2} s_i} \ . \tag{4.31}$$

Note that the denominator in Equation (4.31) is not exactly the length of the two dimensional curve. It is closely related to the length and less sensitive to noise, thus it is a better choice than using classic discrete curve length formula.

Searching Heuristics

To carry out heuristic search, a cost function is used to identify the most preferred node to be explored in each state. The following rules are considered in the construction of the cost functions.

Rule 1 The node with the simplest functional image should be explored first.

Rule 2 The node with the functional image that is easier to be obtained from the original functional image should be consider first.

Rule 3 The transformed functional image that has smaller expected error should be more preferred.

The simplicity of an image is measured by how close it could be fitted to a primitive. The accumulated rank values of the transformations that have been applied to obtain the transformed image reflects the complexity of the possible solution in the corresponding branch. The expected error level, denoted by ϵ, which is traced automatically by the system, could be considered as a factor of the quality of the solution that could be expected to be obtained by further transforming the current functional image. Therefore, the cost function could be define as

$$\text{Cost} = 10^{(-\log\epsilon)\cdot(R_s+R_t)} \cdot E_f \cdot E_c + \delta_e, \tag{4.32}$$

where E_f is the functional primitive fitting error, E_c is the compositional primitive fitting error, R_s is the accumulated rank value of the current node, R_t is the rank value of the transformation that will be applied to generate the image for a new node, and δ_c is an arbitrarily selected small real number[17]. The cost value obtained from Equation (4.32) will be assigned to a new generated node at Step 7 in the search procedure (List 1 on page 119). Apparently, the designed cost function is only a rough estimation of how likely we can find a relatively simple solution in a branch of the search tree. It is not monotonic since a transformed image may have larger E_f and E_c values than the image it is transformed from. This design demonstrates a way to conduct heuristic search in a tough real problem.

4.5 Achieving Adaptive Error Control

It has been pointed out Section 4.2.2 that the error control is an important issue in the design of a direct three-variable function form discovery system. To achieve adaptive error control, we need to compute the expected error level of a transformed image and know when the image is not a smooth image. In this section I will discuss these two issues. First, the theoretical propagated errors corresponding to each transformation will be analyzed. Next, I will define the "roughness value" of an image. Lastly, I will summarize the results with the adaptive error control scheme used by FFD-II.

Before going into details, let me first introduce the symbols which will be used. In this section, an error ϵ is referred to as the *Relative Error*. Let c be a numeric, and $\hat{c}$ be an

[17] In the implementation of FFD-II, δ_e equals 0.1 when the node is an unfavorable node, which means that the associated image contains large scale uncoverable errors, and otherwise it equals 0.

approximation of c, then the error of estimating c with $\hat{c}$ is

$$\epsilon(\hat{c}, c) = \begin{cases} |\hat{c}|\,, & \text{if } \ c \approx 0 \\ \left|\frac{\hat{c}-c}{c}\right|\,, & \text{otherwise}\,. \end{cases} \tag{4.33}$$

Viewing ϵ as the expected error level, which is a positive number, we also write

$$\hat{c} \approx (1 \pm \epsilon) \cdot c\,. \tag{4.34}$$

Without losing generality, we replace $\pm$ with $+$ in the above equation in the formulation of the propagated errors. In the discussion of errors, the following conventions are in effect:

1. The triple $(\tilde{u}_1,\, \tilde{u}_2,\, \tilde{u}_d)$ denotes the accurate sample point of the functional image of an underlying function $u_d = f(u_1,\, u_2)$.

2. The triple $(\bar{u}_1,\, \bar{u}_2,\, \bar{u}_d)$ denotes the approximation of $(\tilde{u}_1,\, \tilde{u}_2,\, \tilde{u}_d)$. The error associated with each parameter is denoted by ϵ_1, ϵ_2 and ϵ_d respectively:

$$\begin{cases} \epsilon_1 = \epsilon(\bar{u}_1, \tilde{u}_1)\,, \\ \epsilon_2 = \epsilon(\bar{u}_2, \tilde{u}_2)\,, \\ \epsilon_d = \epsilon(\bar{u}_d, \tilde{u}_d)\,. \end{cases} \tag{4.35}$$

3. Let T be a transformation and $\bar{T}$ be a numeric implementation of T, and $(\bar{u}_1,\, \bar{u}_2,\, \bar{u}_d)$ be an approximation of $(\tilde{u}_1,\, \tilde{u}_2,\, \tilde{u}_d)$. Then the triple

$$(\tilde{v}_1,\, \tilde{v}_2,\, \tilde{v}_d) = T(\tilde{u}_1,\, \tilde{u}_2,\, \tilde{u}_d) \tag{4.36}$$

denotes the accurate transformed functional image sample point, and the triple

$$(\bar{v}_1,\, \bar{v}_2,\, \bar{v}_d) = \bar{T}(\bar{u}_1,\, \bar{u}_2,\, \bar{u}_d) \tag{4.37}$$

denotes the approximated transformed functional image sample point, where $(\bar{u}_1,\, \bar{u}_2,\, \bar{u}_d)$ is an approximation of $\tilde{u}_1,\, \tilde{u}_2,\, \tilde{u}_d$. The expected errors associated with $\bar{v}_1$, $\bar{v}_2$ and

$\tilde{v}_d$ are $\bar{\epsilon}_1$, $\bar{\epsilon}_2$ and $\bar{\epsilon}_d$ respectively, where

$$\begin{cases} \bar{\epsilon}_1 = \epsilon(\bar{v}_1, \tilde{v}_1) \, , \\ \bar{\epsilon}_2 = \epsilon(\bar{v}_2, \tilde{v}_2) \, , \\ \bar{\epsilon}_d = \epsilon(\bar{v}_d, \tilde{v}_d) \, . \end{cases} \tag{4.38}$$

4.5.1 Error Propagation Analyses

To formulate the error propagations is to find the expressions that express the estimated errors $\bar{\epsilon}_1$, $\bar{\epsilon}_2$ and $\bar{\epsilon}_d$ in terms of ϵ_1 , ϵ_2 and ϵ_d corresponding to each specific transformation sequence T . In other words, it is to find a mapping $\mathcal{E}_T$ of the following

$$(\, \epsilon_1 \, , \, \epsilon_2 \, , \, \epsilon_d \,) \stackrel{\mathcal{E}_T}{\longmapsto} (\, \bar{\epsilon}_1 \, , \, \bar{\epsilon}_2 \, , \, \bar{\epsilon}_d \,) \; . \tag{4.39}$$

From the error propagation perspective, there are two types of transformations — transformations whose propagated error are related to the coordinates of the sample point, and transformations whose propagated error are not related to the coordinates of the sample point. Let us start with the examinations of the simpler type first.

Transformations With Propagated Error Not Related To The Sample Point Coordinates

Transformations T_{VEX} defined by Equation (3.53) and T_{INV} defined by Equation (3.50) do not change the error levels associated with each parameter since they only exchange the position of the corresponding parameters. By the definition of T_{VEX} , the accurate transformed triple is

$$\begin{aligned} (\, \bar{v}_1, \, \bar{v}_2, \, \bar{v}_d \,) &= T_{\text{VEX}} \left(\bar{u}_1, \, \bar{u}_2, \, \bar{u}_d \right) \\ &= (\, \bar{u}_2, \, \bar{u}_1, \, \bar{u}_d \,) \, . \end{aligned}$$

Thus applying T_{VEX} to a data triple containing noises results in[18]

$$(\, \tilde{v}_1, \, \tilde{v}_2, \, \tilde{v}_d \,) \;=\; \left(\bar{v}_1(1+\bar{\epsilon}_1) \, , \, \bar{v}_2(1+\bar{\epsilon}_2) \, , \, \bar{v}_d(1+\bar{\epsilon}_d) \right)$$

[18] For the case of $\bar{u} \approx 0$, similar expressions can be used to obtain the error estimations. Same announcement will not be made in the rest part of this section.

$$\begin{aligned}
&= T_{\mathrm{VEX}}\left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\tilde{u}_d\right)\\
&= (\tilde{u}_2\,,\,\tilde{u}_1\,,\,\tilde{u}_d)\\
&= \left(\bar{u}_2(1+\epsilon_2)\,,\,\bar{u}_1(1+\epsilon_1)\,,\,\bar{u}_d(1+\epsilon_d)\right)\\
&= \left(\bar{v}_1(1+\epsilon_2)\,,\,\bar{v}_2(1+\epsilon_1)\,,\,\bar{v}_d(1+\epsilon_d)\right).
\end{aligned}$$

This proves that the new error levels associated with each of the new parameters (obtained from applying T_{VEX}) are

$$\bar{\epsilon}_1 = \epsilon_2\,,\ \bar{\epsilon}_2 = \epsilon_1\,,\ \bar{\epsilon}_d = \epsilon_d\,. \tag{4.40}$$

Similarly, we can prove that the new error levels associated with each of the new parameters (obtained from applying T_{INV}) are

$$\bar{\epsilon}_1 = \epsilon_d\,,\ \bar{\epsilon}_2 = \epsilon_2\,,\ \bar{\epsilon}_d = \epsilon_1 \tag{4.41}$$

Since the error level are not enlarged, we call the T_{VEX} and T_{INV} transformations error-preserving transformations.

T_{REC} defined by Equation (3.44) is another error-preserving transformation. Let us examine the reason now. According to the definition and using Maclaurin power series expansion, we can express

$$\begin{aligned}
T_{\mathrm{REC}}\left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\tilde{u}_d\right) &= \left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\frac{1}{\tilde{u}_d}\right)\\
&= \left(\bar{u}_2(1+\epsilon_2)\,,\,\bar{u}_1(1+\epsilon_1)\,,\,\frac{1}{\bar{u}_d(1+\epsilon_d)}\right)\\
&= \left(\bar{u}_1(1+\epsilon_1)\,,\,\bar{u}_2(1+\epsilon_2)\,,\,\frac{1}{\bar{u}_d}\left(1-\epsilon_d+O({\epsilon_d}^2)\right)\right)\\
&\approx \left(\bar{v}_1(1+\epsilon_1)\,,\,\bar{v}_2(1+\epsilon_2)\,,\,\bar{v}_d(1-\epsilon_d)\right).
\end{aligned}$$

Therefore, we have proved that the new error levels associated with each of the new parameters (obtained from applying T_{REC}) are

$$\bar{\epsilon}_1 = \epsilon_1\,,\ \bar{\epsilon}_2 = \epsilon_2\,,\ \bar{\epsilon}_d = \epsilon_d \tag{4.42}$$

Transformations With Propagated Error Related To The Sample Point Coordinates

The other three transformations employed by FFD-II, T_{LOG}, T_{FAC}, and T_{DIF}, are different to those we have just seen in their error propagations. The error level through transformation will be changed depending on both the transformation applied and the functional image itself. In the following formulations of error propagations, Taylor power series expansion will be used whereas it is required without mention.

For the transformation T_{LOG} defined by Equation (3.42), the propagated error level estimation could be obtained from[19]

$$\begin{aligned} T_{\text{LOG}}\left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\tilde{u}_d\right) &= (\tilde{u}_1\,,\,\tilde{u}_2\,,\,\log\tilde{u}_d) \\ &= \left(\bar{u}_1(1+\epsilon_1)\,,\,\bar{u}_2(1+\epsilon_2)\,,\,\log\left(\bar{u}_d(1+\epsilon_d)\right)\right). \end{aligned}$$

Note that when $0 \le \epsilon_d \ll 1$

$$\begin{aligned} \log\left(\bar{u}_d(1+\epsilon_d)\right) &= \log\bar{u}_d + \log(1+\epsilon_d) \\ &\approx \log\bar{u}_d + \epsilon_d \\ &= \log\bar{u}_d\left(1+\frac{\epsilon_d}{\log\bar{u}_d}\right). \end{aligned}$$

Since $\bar{v}_1 = \bar{u}_1$, $\bar{v}_2 = \bar{u}_2$ and $\bar{v}_d = \log\bar{u}_d$

$$\begin{aligned} T_{\text{LOG}}\left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\tilde{u}_d\right) &= \left(\bar{u}_1(1+\epsilon_1)\,,\,\bar{u}_2(1+\epsilon_2)\,,\,\log\bar{u}_d\left(1+\frac{\epsilon_d}{\log\bar{u}_d}\right)\right) \\ &= \left(\bar{v}_1(1+\epsilon_1)\,,\,\bar{v}_2(1+\epsilon_2)\,,\,\bar{v}_d\left(1+\frac{\epsilon_d}{\log\bar{u}_d}\right)\right). \end{aligned}$$

It is therefore concluded with the error mapping corresponding to transformation T_{LOG} as

$$\bar{\epsilon}_1 = \epsilon_1\,,\ \bar{\epsilon}_2 = \epsilon_2\,,\ \bar{\epsilon}_d = \frac{\epsilon_d}{\log\bar{u}_d}\,. \tag{4.43}$$

For the transformation T_{FAC} defined by Equation (3.48), the propagated error level is

[19] Without losing generality, we can neglect the absolute operator.

identified as

$$T_{\text{FAC}}\left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\tilde{u}_d\right) \;=\; \left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\frac{\tilde{u}_d}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\right)$$

$$= \left(\tilde{u}_1(\,1+\epsilon_1\,)\,,\,\tilde{u}_2(\,1+\epsilon_2\,)\,,\right.$$

$$\left.\frac{\tilde{u}_d(\,1+\epsilon_d\,)}{\tilde{u}_1(\,1+\epsilon_1\,)\cdot\cos\theta+\tilde{u}_2(\,1+\epsilon_2\,)\cdot\sin\theta+C}\right).$$

Since $\tilde{v}_1=\tilde{u}_1$, $\tilde{v}_2=\tilde{u}_2$ and $\tilde{v}_d=\dfrac{\tilde{u}_d}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}$,

$$\frac{\tilde{u}_d(\,1+\epsilon_d\,)}{\tilde{u}_1(\,1+\epsilon_1\,)\cdot\cos\theta+\tilde{u}_2(\,1+\epsilon_2\,)\cdot\sin\theta+C}$$

$$= \frac{\tilde{u}_d}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\cdot\frac{1+\epsilon_d}{1+\dfrac{\tilde{u}_1\epsilon_1\cdot\cos\theta+\tilde{u}_2\epsilon_2\cdot\sin\theta}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}}$$

$$\approx \tilde{v}_d\cdot(\,1+\epsilon_d\,)\cdot\left(1-\frac{\tilde{u}_1\epsilon_1\cdot\cos\theta+\tilde{u}_2\epsilon_2\cdot\sin\theta}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\right)$$

$$\approx \tilde{v}_d\cdot\left(1+\epsilon_d-\frac{\tilde{u}_1\epsilon_1\cdot\cos\theta+\tilde{u}_2\epsilon_2\cdot\sin\theta}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\right),$$

where $\left|\dfrac{\tilde{u}_1\epsilon_1\cdot\cos\theta+\tilde{u}_2\epsilon_2\cdot\sin\theta}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\right| \ll 1$ is assumed, thus

$$T_{\text{FAC}}\left(\tilde{u}_1\,,\,\tilde{u}_2\,,\,\tilde{u}_d\right) \;=\; \left(\tilde{v}_1(\,1+\epsilon_1\,)\,,\,\tilde{v}_2(\,1+\epsilon_2\,)\,,\right.$$

$$\left.\tilde{v}_d\cdot\left(1+\epsilon_d-\frac{\tilde{u}_1\epsilon_1\cdot\cos\theta+\tilde{u}_2\epsilon_2\cdot\sin\theta}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\right)\right).$$

We can therefore conclude the analysis with the error mapping corresponding to transformation T_{FAC} as

$$\bar{\epsilon}_1=\epsilon_1\,,\;\bar{\epsilon}_2=\epsilon_2\,,\;\bar{\epsilon}_d=\epsilon_d+\left|\frac{\tilde{u}_1\epsilon_1\cdot\cos\theta+\tilde{u}_2\epsilon_2\cdot\sin\theta}{\tilde{u}_1\cdot\cos\theta+\tilde{u}_2\cdot\sin\theta+C}\right|\,. \qquad (4.44)$$

The transformation T_{DIF} is relatively hard to analyze. Since the first two parameters in the variable triple will not be changed by the transformation, it is obvious that

$$\bar{\epsilon}_1 = \epsilon_1 \ , \ \bar{\epsilon}_2 = \epsilon_2 \, . \tag{4.45}$$

To figure out the propagated error $\bar{\epsilon}_d$, we should note that the error can be split into two parts:

Computational Error of Differentials which is the numeric computation errors introduced by the approximation method described in Section 4.4.1;

Propagated Error which is the theoretical difference between the underlying function's derivative values of a clean image and the noisy image.

Let us analyze them separately.

Propagated Error of Differentials

Let (x, y, z) denote the accurate sample points of a function form discovery problem and $z = f(x, y)$ be the underlying function in class C^∞. Let $(\bar{x}, \bar{y}, \bar{z})$ denote the corresponding noisy sample points. Assume that:

$$\begin{cases} x = \bar{x} + \alpha(\bar{x}, \bar{y}) \\ y = \bar{y} + \beta(\bar{x}, \bar{y}) \\ z = \bar{z} - \gamma(\bar{x}, \bar{y}) \end{cases} \tag{4.46}$$

where α, β and γ are the absolute error functions, and $\bar{z} = g(\bar{x}, \bar{y})$ is the corresponding underlying function. We would like to find out the propagated error, i.e. the difference between $g'_{\bar{x}}(\bar{x}, \bar{y})$ and $f'_x(\bar{x}, \bar{y})$.

Substitute Equation (4.46) into $z = f(x, y)$ yields:

$$g(\bar{x}, \bar{y}) = f\,(\bar{x} + \alpha(\bar{x}, \bar{y}) \, , \, \bar{y} + \beta(\bar{x}, \bar{y})\,) + \gamma(\bar{x}, \bar{y}).$$

Therefore, if we assume that $|\alpha|$, $|\beta|$, $|\gamma|$, $|\alpha'_{\bar{x}}|$, $|\beta'_{\bar{x}}|$, $|\gamma'_{\bar{x}}| \ll 1$, which means that the noise level is relatively small, we obtain

$$g'_{\bar{x}}(\bar{x}, \bar{y}) \ = \ f'_x\left(\bar{x} + \alpha \, , \, \bar{y} + \beta\right) \cdot (1 + \alpha'_{\bar{x}}) + f'_y\left(\bar{x} + \alpha \, , \, \bar{y} + \beta\right) \cdot \beta'_{\bar{x}} + \gamma'_{\bar{x}}$$

$$
\begin{aligned}
&\approx \left\{ \left(f'_x + \alpha f''_{xx} + \beta f''_{xy} \right) \cdot \left(1 + \alpha'_{\bar{x}} \right) + \right. \\
&\qquad \left. \left(f'_y + \alpha f''_{yx} + \beta f''_{yy} \right) \cdot \beta'_{\bar{x}} + \gamma'_{\bar{x}} \right\} (\bar{x}, \bar{y}) \\
&\approx \left\{ f'_x + \left(\alpha'_x f'_x + \beta'_x f'_y + \gamma'_x + \alpha f''_{xx} + \beta f''_{xy} \right) \right\} (\bar{x}, \bar{y})
\end{aligned}
$$

Transferring these to relative errors and for the simplicity reason, we assume that, for a majority number of observation sample points[20],

$$|\bar{x}\alpha'_{\bar{x}}| \approx \epsilon_1 \,, \; |\bar{y}\beta'_{\bar{x}}| \approx \epsilon_2 \,, \; |\bar{z}\gamma'_{\bar{x}}| \approx \epsilon_d \tag{4.47}$$

where ϵ_1, ϵ_2, $\epsilon_3 \ll 1$, we have the simple propagated error estimation

$$\delta_p = \left| \frac{\frac{\partial \bar{z}}{\partial \bar{x}} - f'_x}{f'_x} \right| \approx \frac{\frac{\epsilon_1}{x} f'_x + \frac{\epsilon_2}{y} f'_y + \frac{\epsilon_d}{z} + \frac{\epsilon_1}{x} f''_{xx} + \frac{\epsilon_2}{y} f''_{xy}}{f'_x} \,. \tag{4.48}$$

Computational Error

The error analysis we have seen solved the problem of estimating the difference between $T_{\text{DIF}}(\,\bar{u}_1\,, \bar{u}_2\,, \bar{u}_d\,)$ and $T_{\text{DIF}}(\,\bar{u}_1\,, \bar{u}_2\,, \bar{u}_d\,)$. It is one of the two parts of the error that contribute to the propagated error of the differential transformation. Recall that all transformations in the discovery system are implemented numerically. The second part of the propagated error is the error introduced by the computation of $T_{\text{DIF}}(\,\bar{u}_1\,, \bar{u}_2\,, \bar{u}_d\,)$ using the chosen numerical method. In the implementation of FFD-II, T_{DIF} is calculated by a fitting scheme (Refer to Section 4.4.1). We now analyze the error associated with this computation scheme.

Let:

(1) $p_0 = (0,0)$ be the point at which we numerically compute the partial derivative of a C_∞ function $z = f(x,y)$ using our fitting method;

(2) p_i, $i = 1, \cdots, 8$ be the eight nearest points involved in the fitting, and $\boldsymbol{F_P} = \{\, p_i \mid i = 0, \cdots, 8 \,\}$ denote the fitting point set;

(3) Δ be the maximum distance $\overline{p_0 p_i}$ (for $i = 1, \cdots, 8$);

[20] Considering that the error is also monitored by the measurement of surface roughness that will be discussed soon, the assumption will not mislead the proposed error control strategy.

(4) $-\Delta \leq x \leq \Delta$, $-\Delta \leq y \leq \Delta$ be a small planar region;

(5) $\bar{P}(x,y) = \bar{a}_{11}x^2 + \bar{a}_{12}xy + \bar{a}_{22}y^2 + \bar{a}_1 x + \bar{a}_2 y + \bar{a}_0$ be the least-squares fitting polynomial; and

(6) $P(x,y) = a_{11}x^2 + a_{12}xy + a_{22}y^2 + a_1 x + a_2 y + a_0$ be the polynomial of truncated Maclaurin series of function $z = f(x,y)$. Thus,

$$a_1 = \left.\frac{\partial f}{\partial x}\right|_{p_0}, \quad a_2 = \left.\frac{\partial f}{\partial y}\right|_{p_0} .$$

Since

(i) function $f(x,y)$ is pertaining to class C^∞ , we can assume that there is a positive constant K such that the corresponding derivative values are bounded by K ;

(ii) $f(x,y) - P(x,y) = \left(\Delta\frac{\partial}{\partial x} + \Delta\frac{\partial}{\partial y}\right)^3 f(\xi,\xi) = O(\Delta^3)$,
where $-\Delta \leq \xi \leq \Delta$; and

(iii) $\bar{P}(x,y)$ is the least-squares fitting and considering the existence of the polynomial function $P(x,y)$, it must satisfies:

$$\begin{aligned} f(x,y) - \bar{P}(x,y) &= O(\Delta^3) \\ P(x,y) - \bar{P}(x,y) &= O(\Delta^3) \end{aligned} \tag{4.49}$$

at each point p_i .

we would like to prove that the second order polynomial

$$\hat{P}(x,y) = \bar{P}(x,y) - P(x,y) = a + bx + cy + dx^2 + exy + fy^2, \tag{4.50}$$

where the coefficients are the subtractions of the corresponding coefficients in $\bar{P}$ and P , satisfies

$$b = O(\Delta^2), \tag{4.51}$$

provided that the fitting point set is subjected to certain constraints.

It is obvious that if the fitting points is scattered very close to a line, as shown in Figure 4.9(b), the functional image of the resultant fitting polynomial may not be very even, even when Equation (4.49) are satisfied. To prevent this from happening, we assume that point p_0 adequately close to the center of the point set $\boldsymbol{F_P}$ (illustrated by Figure 4.9(c)), i.e. there exist five points E, N, W, S and V in set $\boldsymbol{F_P}$, such that

- the distances between any points pair are $O(\Delta)$;
- if I is the point where lines $\overline{NS}$ and $\overline{EW}$ cross, and κ denotes the distance between points I and p_0, then

$$\kappa = O(\Delta^2)\,; \tag{4.52}$$

- Angle ϕ is not close to either 0 or π.

Under these conditions, we now prove $b = O(\Delta^2)$.

Without loosing generality, we assume that $\phi = \pi/2$ (Refer to Figure 4.9(d)). Otherwise, a coordinate transformation can be applied to simplify the situation without changing the first order properties (of our interests) of the polynomial of Equation (4.50). For the convenience of discussion, we assume that the coordinate origin is initially at point p_0. Moving the origin to point I results in a transformed polynomial of (4.50):

$$\widehat{P}(\bar{x},\bar{y}) = \bar{a} + \bar{b}\bar{x} + \bar{c}\bar{y} + \bar{d}\bar{x}^2 + \bar{e}\bar{x}\bar{y} + \bar{f}\bar{y}^2\,. \tag{4.53}$$

From given, we have

$$\left\{\begin{array}{llll}
\widehat{P}(E) = & \bar{a} + \bar{b}D_E + \bar{d}D_E^2 & = O(\Delta^3) & \text{(E1)}\\
\widehat{P}(W) = & \bar{a} - \bar{b}D_W + \bar{d}D_W^2 & = O(\Delta^3) & \text{(E2)}\\
\widehat{P}(N) = & \bar{a} + \bar{c}D_N + \bar{f}D_N^2 & = O(\Delta^3) & \text{(E3)}\\
\widehat{P}(S) = & \bar{a} - \bar{c}D_S + \bar{f}D_S^2 & = O(\Delta^3) & \text{(E4)}\\
\widehat{P}(V) = & \bar{a} + \bar{b}\bar{x}_V + \bar{c}\bar{y}_V & & \\
 & \quad + \bar{d}\bar{x}_V^2 + \bar{e}\bar{x}_V\bar{y}_V + \bar{f}\bar{y}_V^2 & = O(\Delta^3) & \text{(E5)}\\
\widehat{P}(p_0) = & \bar{a} + \bar{b}\bar{x}_{p_0} + \bar{c}\bar{y}_{p_0} & & \\
 & \quad + \bar{d}\bar{x}_{p_0}^2 + \bar{e}\bar{x}_{p_0}\bar{y}_{p_0} + \bar{f}\bar{y}_{p_0}^2 & = O(\Delta^3) & \text{(E6)}
\end{array}\right.$$

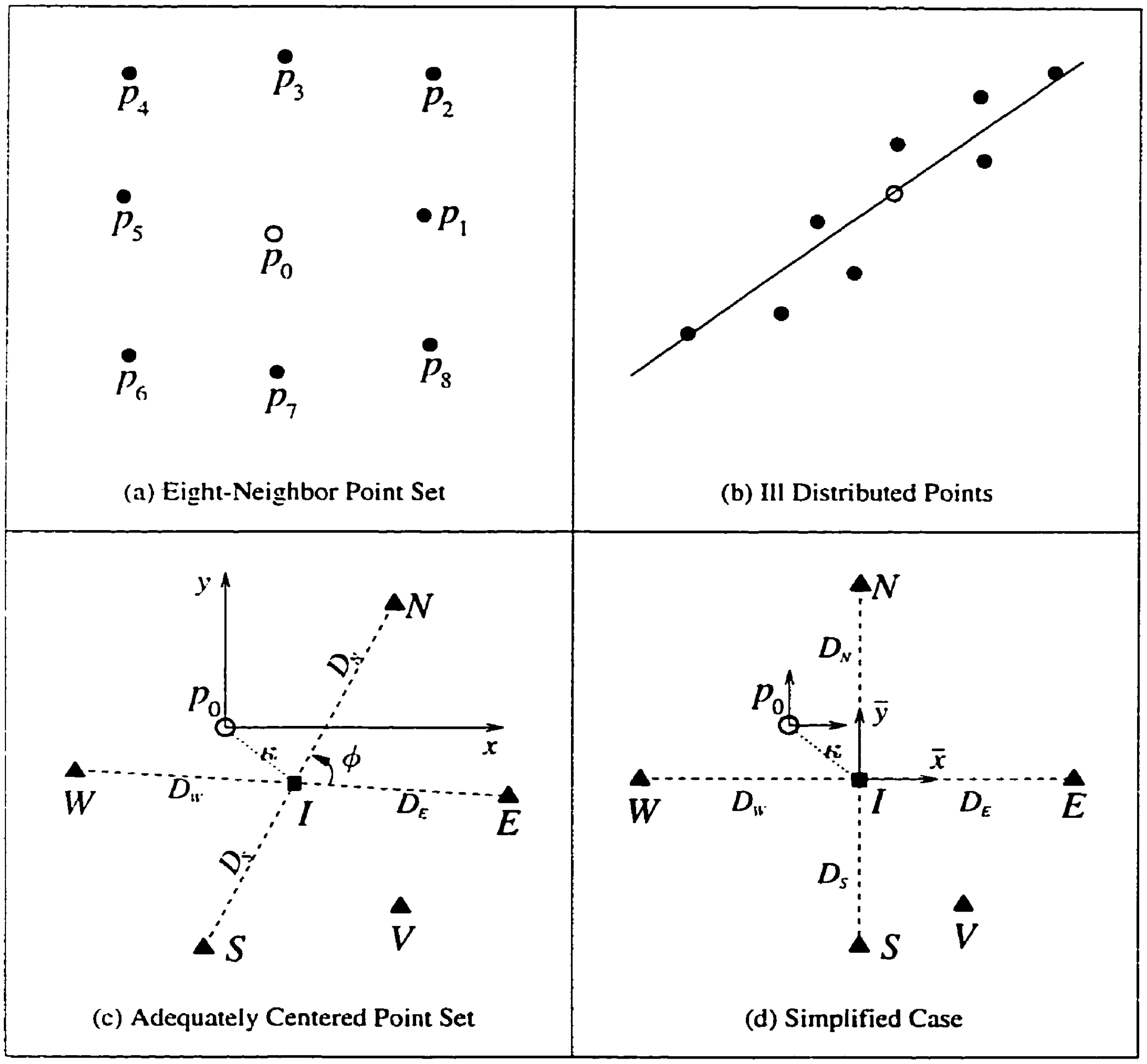


Figure 4.9: Polynomial Fitting Points

where D_E, D_W, D_N, D_S, x_v and y_v are infinitesimals of the same order of $O(\Delta)$, and x_{p0} and y_{p0} are infinitesimals of the order $O(\kappa) = O(\Delta^2)$. Now we prove $\bar{a}$ is an infinitesimal of order $O(\Delta^3)$ by contradiction. Let us suppose $\bar{a} = O(\Delta^n)$ for integer $n < 3$ (when $n < 0$ $\bar{a}$ is an infinite). Then, equations (E1) and (E2) imply

$$\begin{cases} \bar{b}D_E + \bar{d}D_E^2 &= O(\Delta^n) \qquad \text{(E7)} \\ -\bar{b}D_W + \bar{d}D_W^2 &= O(\Delta^n). \qquad \text{(E8)} \end{cases}$$

Eliminate $\bar{b}$ from above yields

$$\bar{d} \cdot (D_E^2 D_W + D_E D_W^2) = (D_E + D_W) \cdot O(\Delta^n). \qquad \text{(E9)}$$

Clearly, $\bar{d} = O(\Delta^{n-2})$ so that, by comparing the order of each term in equation (E7) or (E8), $\bar{b} = O(\Delta^{n-1})$. Similarly, using equation (E3) and (E4) it can be proved that $\bar{c} = O(\Delta^{n-1})$ and $\bar{f} = O(\Delta^{n-2})$. By comparing the orders of each term in equation (E5) we find the order of the only unknown symbol $\bar{e} = O(\Delta^{n-2})$. Now we check with equation (E6) to figure out the order of $\bar{a}$. In the equation

$$\bar{a} + (\bar{b}\bar{x}_{p0} + \bar{c}\bar{y}_{p0} + \bar{d}\bar{x}_{p0}^2 + \bar{e}\bar{x}_{p0}\bar{y}_{p0} + \bar{f}\bar{y}_{p0}^2) = O(\Delta^3)$$

considering the assumption of $\bar{x}_{p0}$, $\bar{y}_{p0} = O(\Delta^2)$ and the results we have proved, the terms $\bar{b}\bar{x}_{p0}$ and $\bar{c}\bar{y}_{p0}$ are of $O(\Delta^{n+1})$ and the terms $\bar{d}\bar{x}_{p0}^2$, $\bar{e}\bar{x}_{p0}\bar{y}_{p0}$ and $\bar{f}\bar{y}_{p0}^2$ are of $O(\Delta^{n+2})$. Considering the assumption of n being less than 3, term $\bar{a}$ must be of $O(\Delta^{n+1})$. This is contradictory to the assumption of $\bar{a} = O(\Delta^n)$. Therefore,

$$\bar{a} = O(\Delta^3).$$

According to the result of $\bar{a} = O(\Delta^3)$, we can use the same method to prove $\bar{b}$, $\bar{c} = O(\Delta^2)$ and $\bar{d}$, $\bar{e}$, $\bar{f} = O(\Delta)$. Since

$$\begin{cases} \bar{x} &= x + \bar{x}_{p0} \\ \bar{y} &= y + \bar{y}_{p0} \end{cases}$$

the following orders could be figured out:

$$\begin{cases} a &= \bar{a} + \bar{b}\cdot\bar{x}_{p_0} + \bar{c}\cdot\bar{y}_{p_0}\,, \\ & \quad + \bar{d}\cdot\bar{x}_{p_0}^2 + \bar{e}\cdot\bar{x}_{p_0}\bar{y}_{p_0} + \bar{f}\cdot\bar{y}_{p_0}^2 &= O(\Delta^3)\,, \\ b &= \bar{b} + 2\bar{d}\cdot\bar{x}_{p_0} + \bar{e}\cdot\bar{y}_{p_0} &= O(\Delta^2)\,, \\ c &= \bar{c} + \bar{e}\cdot\bar{x}_{p_0} + 2\bar{f}\cdot\bar{y}_{p_0} &= O(\Delta^2)\,, \\ d &= \bar{d} &= O(\Delta)\,, \\ e &= \bar{e} &= O(\Delta)\,, \\ f &= \bar{f} &= O(\Delta)\,. \end{cases}$$

Recalling the definitions of $\hat{P}(x,y)$ (Equation (4.50)), $\tilde{P}(x,y)$ and $P(x,y)$ (list items (5) and (6) on page 142), we can conclude the discussion with the estimated computational error

$$\delta_c = O(\Delta^2)\,. \tag{4.54}$$

The above proof is also valid when condition (4.52) is replaced by condition $\frac{\kappa}{\Delta} \ll 1$.

4.5.2 Surface Roughness and Smoothing

Roughness Measurement

Let $\mathcal{O} = \{\,(u_{i,j}\,,\, v_{i,j}\,,\, w_{i,j})\mid i = 1,\cdots,Nx\,;\, j = 1,\cdots,Ny\}$ be a functional image sample point set. We define the roughness of the surface at an inner point $(i\,,\,j)$, where $1 < i < Nx$, $1 < j < Ny$, as follows.

Let[21]

$$\begin{cases} f_0 = w_{i-1,j}\,,\ f_1 = w_{i,j} - f_0\,,\ f_2 = w_{i+1,j} - f_0\,, \\ t_1 = \sqrt{(u_{i,j} - u_{i-1,j})^2 + (v_{i,j} - v_{i-1,j})^2}\,, \\ t_2 = \sqrt{(u_{i+1,j} - u_{i,j})^2 + (v_{i+1,j} - v_{i,j})^2} + t_1\,, \\ \mu = \dfrac{|\,t_2 f_1 - t_1 f_2\,|}{\sqrt{f_2^2 + t_2^2}}\,. \end{cases}$$

Clearly, if

[21] See Figure 4.10 for graphical illustrations.

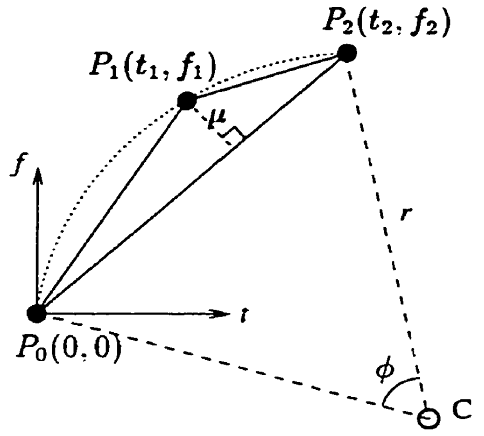


Figure 4.10: An Illustration of The Roughness Measure

- we let $f = F(t)$ be a continuous function such that $f_i = F(t_i)$ for $i = 0, 1, 2$;
- we assume that $t_2 = 2\,t_1$ and $\mu \ll \left|\overline{P_0P_2}\right|$; and
- we let r and ϕ denote the radius and angle of the circular arch $P_0P_1P_2$,

then

$$\frac{\mu}{\left|\overline{P_0P_2}\right|} = \frac{r - r\cos(\phi/2)}{2r\sin(\phi/2)} \approx \frac{\phi}{8} \approx \frac{\left|\overline{P_0P_2}\right|}{8r} \tag{4.55}$$

which is the approximated curvature value of $f = F(t)$ at point P_1 multiplied by the dimension of the length of the small curve segment and divided by 8. We assume that for a smooth curve point, $r = O(1)$, thus:

$$\frac{\mu}{\left|\overline{P_0P_2}\right|} = O\left(\left|\overline{P_0P_2}\right|\right) .$$

Similarly, let

$$\begin{cases} g_0 = w_{i,j-1}\,,\ g_1 = w_{i,j} - g_0\,,\ g_2 = w_{i,j+1} - g_0\,, \\ s_1 = \sqrt{(\,u_{i,j} - u_{i,j-1}\,)^2 + (\,v_{i,j} - v_{i,j-1}\,)^2}\,, \\ s_2 = \sqrt{(\,u_{i,j+1} - u_{i,j}\,)^2 + (\,v_{i,j+1} - v_{i,j}\,)^2} + s_1\,, \\ \nu = \dfrac{|\, s_2 g_1 - s_1 g_2 \,|}{\sqrt{g_2^2 + s_2^2}}\,. \end{cases}$$

Hence, the *Roughness Value* at the surface point (i, j) is defined as

$$\varrho_{ij} = \frac{\mu}{\sqrt{t_2^2 + f_2^2}} + \frac{\nu}{\sqrt{s_2^2 + g_2^2}}\,. \tag{4.56}$$

Roughly speaking, this measurement is a sum of second order curvatures of two un-parallel curves on the surface that cross at the surface point (i, j) multiplied by the corresponding length of the curve segments.

The *Roughness Value* of a surface is the averaged integration of the roughness elements at each surface sample points

$$\varrho = \left(\sum_{i=2}^{Nx-1} \sum_{j=2}^{Ny-1} \varrho_{ij} \right) /(Nx - 2)/(Ny - 2)\,. \tag{4.57}$$

The measurement given above is based on the measurement of curvatures of the curves in a curve set on the surface. The observation data set $\mathcal{O}$ presents the surface by a net of discretized curves on the surface. Although Equation (4.56) is not the exact curvature measurement of the surface at the corresponding point, it is necessary that $\varrho_{ij} = O(\Delta)$ for Δ being the averaged partitioning size of the mesh grid, provided that the curve net is reasonably close to a uniformly generated net[22]. The quantity of the defined roughness value at a mesh grid point responses to non-smooth noise pattern with a large value. As such, the defined surface roughness measurement can meet our need of capturing the roughness pattern of the surface caused by noises and errors.

[22] *Uniformly Generated* means that there is a small number Δ — the partitioning size, such that the observation data set is
$$\mathcal{O} = (u_{i,j}, v_{i,j}, w_{i,j}) = (u_1 + i \cdot \Delta\,,\ v_1 + j \cdot \Delta\,,\ w_{i,j})$$
for $i = 1, \cdots, Nx$; $j = 1, \cdots, Ny$.

Smoothing a Surface

Differential transformation is the most important transformation that allows a data transformation based function form discovery system to have the flexibilities of finding complex function form expressions. However, it is more difficult and inaccurate to compute the transformation numerically than other algebra transformation. To fully take advantages of data transformations, the ability to reduce the computational errors is an important issue. Before we chose the smoothing method, there are a few things that should be borne in our mind.

- The smoothing scheme must add, as less as possible, specific functional pattern of its own to the image to be smoothed. Or, at least no significant functional pattern will be added.
- In the three-variable cases the smooth scheme should be able to handle observation data set that is not regularly distributed. In other words, the mesh grid might not be uniformly distributed rectangles.
- Since the smoothing treatment will be called from time to time and the size of a single observation data set are usually large, it is better to be a simple method that works.

These are the criteria for designing the smoothing algorithm for FFD-II.

Moving window averaging is probably the simplest and the most widely used noise removing technique. If the mesh points are adequately placed and the underlying function is constant, or is changing linearly with the independent variables, no bias is introduced into the result. A bias is introduced, however, when the underlying function has a nonzero second derivatives. To prevent the bias introduced due to nonzero curvature, *Digital Smoothing Polynomial*[18] is an alternative. In one dimension cases where $\{(t_i, f_i), i = 1, 2, \cdots\}$ is the sample data set, instead of directly replacing data value f_i at each sample point i by a linear combination:

$$g_i = \sum_{\text{nearby of } i} c_k f_k$$

we replace f_i by p_i :

$$p_i = P_i^{[n]}(t_i)$$

where $P_i^{[n]}(t)$ is the nth order polynomial that is the result of fitting the sample points within the ith moving window by least-squares. This idea is borrowed to solve our smoothing problem.

Let $\mathcal{O} = \{ (u_{i,j} , v_{i,j} , w_{i,j}) \mid i = 1, \cdots, Nx ; j = 1, \cdots, Ny \}$ be a functional image sample point set that need to be smoothed. We choose a 5×5 moving window, i.e. let the fitting point set at a point (i, j) be[23]:

$$\left\{ (u_{i+l,j+h} , v_{i+l,j+h} , w_{i+l,j+h}) \,\middle|\, l, h = -2, -1, 0, 1, 2 \right\}, \tag{4.58}$$

and let the second order two-dimensional polynomial:

$$\begin{aligned} z = P_{i,j}(x, y) &= a (x - u_{i,j})^2 + b (x - u_{i,j})(y - v_{i,j}) + c (x - u_{i,j})^2 \\ &\quad + d (x - u_{i,j}) + e (y - v_{i,j}) + f . \end{aligned} \tag{4.59}$$

be the fitting polynomial. Clearly, second order curvatures of the underlying function do not introduce bias by choosing a complete second order fitting polynomial. However, bias are introduced by possible higher order curvatures. Choosing higher order polynomial may be a solution to overcome this problem. But at this moment, we limit the complexity of the entire system with the simplest possible choice.

The laborious least-squares fitting is linear. We solve it by LU decomposition. More luckily, we need only to find the constant term f. This simplifies the back-substitution procedure of LU decomposition process.

In general, a smoothing algorithm does not improve the precision of the data. In the FFD-II system, the smoothing method is applied to a non-smooth image recursively until the image is sufficiently smooth. Since the smoothing method introduced in this section is an averaging based method, it works better when there are sufficient sample points involved in a single computation of the average. Thus the smoothing process will only be triggered

[23] Note that special attentions must be paid to the points on and next to the observation boundary.

when there are extra sample points, available in the form of fine step image, that could be used to help the image smooth.

4.5.3 Error Monitor and Adaptive Control

Smoothing algorithms do not improve the precision of the data. It only remove the high frequency oscillations from the data. A more effective way to improve the precision of the observation data is to use more sample points with finer observation step size. In the FFD-II system, the smoothing method introduced in the last section is applied recursively to smooth the image until the image is sufficiently smooth. An image refinement scheme is used to improve the precision of the image when the estimated error level is too high. The refinement is implemented by using fine step observation data with larger fitting windows, i.e. windows with the same dimensional magnitude and more sample points. The system monitors the expected error level and the smoothness of the transformed functional image to decide which treatment is needed.

Summarizing the results of error analyses conducted in Section 4.5.1, the estimated error propagations of each transformations are tabulated in Table 4.1. The estimation formulas of T_{VEX}, T_{INV}, and T_{REC} are exactly their theoretical results. T_{LOG} and T_{FAC} are roughly estimated. In the error estimation of T_{LOG} , $\log(\tilde{u}_d)$ is replaced by $\log(\frac{\tilde{u}_{d_{\max}}-\tilde{u}_{d_{\min}}}{2})$, and in the error estimation of T_{FAC} , maximum error level of the three attributes is adopted. The error estimation of T_{DIF} is relatively rougher. In the theoretical result, Equation 4.48 is replaced with $\max\{\epsilon_1, \epsilon_2, \epsilon_3\}$, and Equation 4.54 is replaced by $\left(\frac{\tilde{u}_{d_{\max}}-\tilde{u}_{d_{\min}}}{(Nx+Ny)/2}\right)^2$. The reasons for estimating the errors in this way are list below.

- The purpose of estimating the propagated error level is to capture the order of the expected error level. We require only

$$O(\bar{\epsilon}) = O(\epsilon(p)) \tag{4.60}$$

where $\bar{\epsilon}$ is our estimated error level and $\epsilon(p)$ is the real error at any observation sample point p. When the functional image is sufficiently smooth and the mesh grid

Trans.	Theoretical Results(Eq.)	Estimated Error
T_{VEX}	(4.40)	$\bar{\epsilon}_1 = \epsilon_2$, $\bar{\epsilon}_2 = \epsilon_1$, $\bar{\epsilon}_d = \epsilon_d$
T_{INV}	(4.41)	$\bar{\epsilon}_1 = \epsilon_d$, $\bar{\epsilon}_2 = \epsilon_2$, $\bar{\epsilon}_d = \epsilon_1$
T_{REC}	(4.42)	$\bar{\epsilon}_1 = \epsilon_1$, $\bar{\epsilon}_2 = \epsilon_2$, $\bar{\epsilon}_d = \epsilon_d$
T_{LOG}	(4.43)	$\bar{\epsilon}_1 = \epsilon_1$, $\bar{\epsilon}_2 = \epsilon_2$, $\bar{\epsilon}_d = \dfrac{\epsilon_d}{\log \dfrac{\tilde{u}_{d_{\max}} - \tilde{u}_{d_{\min}}}{2}}$
T_{FAC}	(4.44)	$\bar{\epsilon}_1 = \epsilon_1$, $\bar{\epsilon}_2 = \epsilon_2$, $\bar{\epsilon}_d = \max\{\epsilon_1, \epsilon_2, \epsilon_d\}$
T_{DIF}	(4.45), (4.48), (4.54)	$\bar{\epsilon}_1 = \epsilon_1$, $\bar{\epsilon}_2 = \epsilon_2$, $\bar{\epsilon}_d = \left(\dfrac{\tilde{u}_{d_{\max}} - \tilde{u}_{d_{\min}}}{(\text{Nx+Ny})/2} \right)^2 + \max\{\epsilon_1, \epsilon_2, \epsilon_d\}$

Table 4.1: Estimated Error Propagations

has not been badly distorted, the requirement will be satisfied.

- Our interest is to compute the "overall averaged order" of the the propagated error. A rough estimation is sufficient for serving as an error treatment heuristics.
- The estimated error is largely different from the true value at those points where the functional image is not smooth. For that kind of exceptions, the designed smoothing heuristics and procedure will take charge of the situations.

FFD-II achieves the error control based on the following two heuristics.

Heuristics E1 : If a functional image associated with a searching node has too large expected propagated error level, it may not be desirable to find a solution based on the image.

Heuristics E2 : If a functional image is not sufficiently smooth, measured by the roughness measurement ϱ , the image should be treated with surface smoothing.

In the former case, the best way to reduce the error is to refine the image with fine step observations when the primitive fitting error is sufficiently small[24]. FFD-II first checks if there are more observation data available for refining the current image. If yes, image refinement is conducted. Otherwise, the system marks the corresponding child nodes as unfavorable nodes by increasing the cost value of the corresponding child nodes. In the latter case, surface smoothing will be called recursively to smooth the image until a smooth surface is obtained.

[24] If the primitive fitting error exceeds a certain limit, for example ten times the corresponding expected error level, we have evidence to believe that the image is not primitive. Smoothing such an image is not necessary.

Chapter 5

Experiments

5.1 The Organization and Common Background of the Experiment

The implementation of the proposed methodology is the FFD-II system. It is written in C++ programming language with over 13,000 lines of code. The experiments are run on a SUN SPARC Ultra-1 machine, that is equipped with a 167MHz CPU and has 62MB RAM. This chapter is a report of the experiment results.

Before the discussion of the experimental results, two detailed examples are presented in the section that follows. The purpose of presenting the examples is to help to understand the proposed methodology. The experiments are then organized into four categories.

The first group presented in Section 5.3 is designed to demonstrate the general discovery capability of the system based on the proposed function form description language described in Section 3.3. To minimize fortuitous results, a random test function form construction scheme is used to select function forms to be tested randomly.

The second group of experiments presented in Section 5.4 is a comparison between the proposed direct three variable function form discovery method and the indirect method using variable freezing technique. As mentioned before, FFD was designed to discover function forms from two variables. It has been extended to discover function forms with extra parameters, known as families of one dimensional functions, using the parameter freezing technique. The extension is subject to certain constraints such as the "Primitive Union" and "Simple Descriptive Parameters" assumptions. However, the comparisons in the second experiment group will be made against a more general indirect function form discovery system, namely "Indirect-FFD" which will be introduced before the experiments are discussed. To demonstrate the superiority of the proposed direct approach over the indirect approach, special function forms are chosen.

The third group of experiments, reported in Section 5.5, is designed for the purpose of demonstrating the system's ability to model observations from more complex function forms that cannot be expressed in terms of a few fundamental functions. Randomly generated two-dimension surfaces are chosen as test samples. The emphases is on the meaningfulness of the discovered expressions.

The last experiment group, described in Section 5.6, tests the performance of the error treatment design of the system. Noises are added to the simulated observation data set to produce input observations. Different statistics will be used to show the effectiveness of the proposed methodology.

The test function forms and all intermediate transformed function forms are assumed to represent continuous functions in their corresponding observation domains. The computational complexity of the algorithm is not reported[1]. Instead, I will report the number of nodes created and the number of primitive matches the system attempted before it reached the solution in a discovery task. These values reflect the efficiency of the designed search and redundancy elimination heuristics.

[1] Roughly speaking, the time consumed in computing a node is linear to the size of the observation data set associated with it, and the actual time complexity depends on the numeric tools we chose to carry out data transformations and primitive matchings. The current system stores and processes a functional image in the form of double precision floating-point data. This consumes large amounts of memory space and CPU time. The CPU consumption of the experiments reported in this chapter range from 2 seconds to about half an hour, depend on the complexity of the discovery task.

The system is run under the following default system parameter settings unless otherwise stated.

Sampling: In each experiment, the observation domain is carefully arranged to conserve the continuity assumption. The partitioning sizes are chosen to be from 0.008 to 0.02.

Accept A Fitting: The threshold $\epsilon_{\mathcal{P}}$ is set to $\max(5 \times \epsilon_n, 10^{-5})$, where ϵ_n is the expected error of the corresponding functional image associated with each node. The system estimates ϵ_n according to the original precision, the step size, transformation history of the node and the function range of the corresponding images[2]. When a transformed functional image can be fitted to a primitive with a fitting error[3] less than the threshold, the fit will be accepted, a hypothesis will be abstracted, and the verification process will be triggered.

Accept A Hypothesis: The threshold $\epsilon_{\mathcal{M}}$ for accepting a hypothesis is set to $\max(\epsilon_n, 10^{-5})$. This is referred to as the Matching Error Tolerance Level $\delta_{\max}$ in our problem statement (Section 3.2.2). When the deviation measured by the root-mean-square distance[4] between the matching image (obtained by numerically reversing a fitting image with corresponding reverse transformations) and the original observation functional image is less than this threshold, the system will terminate with a successful discovery.

Smooth Image and *Error Corrupted Image:* When the computed roughness value[5] of an image is greater than

$$0.1\sqrt{\frac{\text{Area of the Observation Domain}}{\text{Number of Sample Points}}},$$

the image will be viewed as a rough image. A surface smoothing process will be triggered. When the computed expected error of an image is greater than the

[2] Refer to Section 4.5.3.

[3] See Section 4.4.2 for details.

[4] Refer to Section 4.4.2.

[5] Refer to Section 4.5.2 for the definition of roughness value.

square root of the original input noise, the image will be viewed as an image with unacceptable level of errors. Image refinement will be triggered. However, in both of the cases, if there are no additional sample points, the treatment process will not be triggered. Instead, the system will assign low priority to all nodes in the current branch.

These settings are based on the consideration of the accuracies of the numeric tools that have been chosen. We should note that numeric integration is less sensitive to noise than numeric differentiation[6]. Hence it is reasonable to set $\epsilon_{\mathcal{M}} < \epsilon_{\mathcal{P}}$. The arbitrary value 10^{-5} represents the basic numeric fitting and integral accuracies and ϵ_n adaptively takes into account the accuracy of the observation data set upon which the numeric computations are carried out. Considering the time and memory space intensities of the system, we also set the maximum search depth to 7 and the maximum rank of a function form to 10. We will identify changes to these settings whenever necessary.

5.2 Two Detailed Examples

Examples are helpful for understanding the proposed discovery mechanism and the subsequent experiments. In this sections, we will see two detailed examples that demonstrate the two different termination primitive types of the system and how the system works.

5.2.1 Example 1: Termination by Primitive Function Fitting

The first function form to be discovered is

$$z = 1 + e^{x \cdot y} \tag{5.1}$$

The simulated observation data set is obtained by partitioning the observation domain $(x, y) \in [-1, 1; -1, 1]$ into a 101×101 mesh-grid. As such, the observation data set contains

[6] Major errors are introduced by the approximations of differential transformations in the process of finding a matching hypothesis that involves differential transformations. The verification process only contains algebraic and integral transformations.

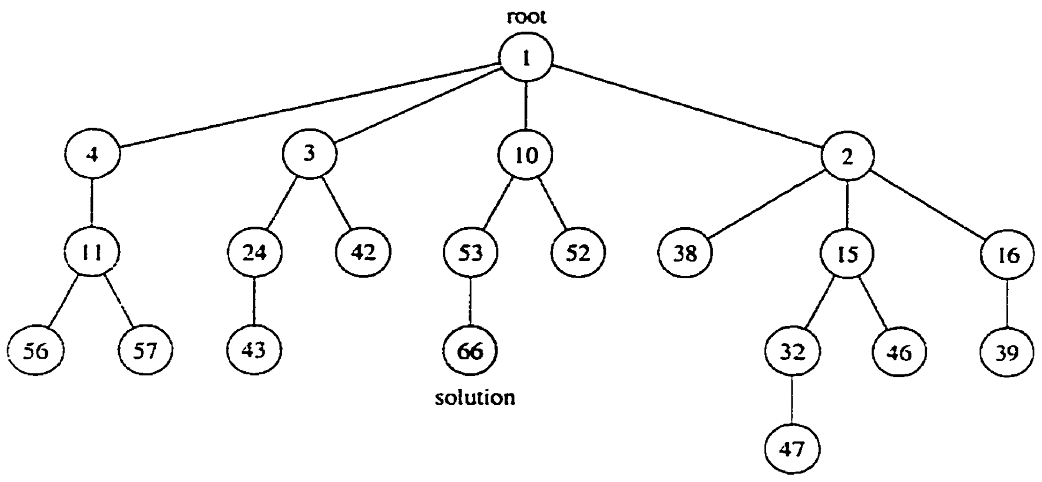


Figure 5.1: The Search Tree — Example 1

101 × 101 = 1021 real valued 3-tuples and can be expressed as:

$$\mathcal{O} = \left\{ (x_i, y_j, 1 + e^{x_i \cdot y_j}) \;\middle|\; \begin{array}{l} x_i = -1 + 0.02\,i\,,\ i = 0, 1, \cdots, 100\,, \\ y_i = -1 + 0.02\,j\,,\ j = 0, 1, \cdots, 100 \end{array} \right\}.$$

The discovery steps are illustrated in Figure 5.1 and detailed information is given in Table 5.1, where "Trans." shows the associated data transformations, "Cost" refers to the value of searching heuristic cost function corresponding to each node, "Exp-Error" stands for the estimated error propagation, "Pf-Error" is the functional primitive fitting error, "Pp-Error" is the compositional primitive fitting errors and "M-Error" is the matching error of hypothesis verification. The number shown in each node in Figure 5.1 represents the order in which it was explored. The discovery is terminated at step 66.

Let us describe the discovery process step by step. After initializing the tree root (Node 1) with the given observation data set $\mathcal{O}$, the system starts to search for the solution:

Step 1: The system attempts to find a primitive function matching the image $\mathcal{O}$ by fitting it to primitive functions. The best fitting is

$$z = 0.1778x^2 - 1.0641xy + 0.1778y^2 + 1.9388,$$

Step	Trans.	Cost	Exp-Error	Pf-Error	Pp-Error	M-Error
1	None	–	1.00e-12	5.130e-02	2.117e-05	–
2	T_{LOG}	9.01e-06	1.00e-12	1.121e-02	3.346e-05	–
3	T_{REC}	9.01e-06	1.00e-12	1.576e-03	5.830e-05	–
4	T_{VEX}	9.01e-06	1.00e-12	*5.130e-02* †	*2.117e-05*	–
10	T_{DIF}	9.01e-05	4.00e-04	1.917e-01	1.362e-03	1.1483e-02
11	T_{DIF}	9.01e-05	4.00e-04	1.917e-01	1.362e-03	1.1483e-02
15	T_{REC}	3.35e-04	1.00e-12	5.049e-02	2.071e-05	–
16	T_{VEX}	3.35e-04	1.00e-12	*1.121e-02*	*3.346e-05*	–
24	T_{VEX}	5.83e-04	1.00e-12	*1.576e-03*	*5.830e-05*	–
32	T_{VEX}	8.52e-04	1.00e-12	*5.049e-02*	*2.071e-05*	–
38	T_{DIF}	3.35e-03	4.00e-04	2.039e-02	4.537e-02	–
39	T_{DIF}	3.35e-03	4.00e-04	2.039e-02	4.537e-02	–
42	T_{DIF}	5.83e-03	4.00e-04	2.012e-03	4.484e-01	–
43	T_{DIF}	5.83e-03	4.00e-04	2.012e-03	8.201e-01	–
46	T_{DIF}	8.52e-03	4.00e-04	1.888e-01	3.270e-03	–
47	T_{DIF}	8.52e-03	4.00e-04	1.888e-01	3.270e-03	–
52	T_{VEX}	2.51e-01	4.00e-04	*1.917e-01*	*1.362e-03*	*1.1483e-02*
53	$T_{\text{FAC}}\|_{\underline{y}}$	2.51e-01	4.00e-04	4.698e-02	2.303e-05	1.346e-02
56	T_{VEX}	2.51e-01	4.00e-04	*1.917e-01*	*1.362e-03*	–
57	$T_{\text{FAC}}\|_{\underline{x}}$	2.51e-01	4.00e-04	4.698e-02	2.320e-05	1.3126e-02
66	T_{LOG}	2.30e+00	4.00e-04	1.355e-03	1.467e-05	1.3118e-06

†: italic denotes the value is obtained from parent node.

Table 5.1: Primitive Fitting and Matching of Each Step of Example 1

and the fitting error is 5.130E-2. It is not an acceptable fit considering that the expected error of $\mathcal{O}$ is $\epsilon_n = 1.0\text{E} - 12$, the precision of 64 a bit floating-point number. Thus $\epsilon_{\mathcal{P}} = 10^{-57}$.

To find a compositional primitive matching of the image $\mathcal{O}$, the system uses the multi-line fitting algorithm. The best fit is

$$-0.0029x + 1.0000y,$$

and the fitting error is 2.117E-5, which is not acceptable. It is concluded that $\mathcal{O}$ is not primitive.

The system then constructs six child nodes, corresponding to the transformations listed in Table 3.6, without computing the associated images. It assigns each new node a cost value computed accordingly, and puts them into a sorted list "OPEN".

Step 2: The node with the least cost value is chosen and removed from the "OPEN" list. The system determines that the associated transformation, T_{LOG}, is applicable. Data transformation is then carried out to generate a new associated functional image $\mathcal{O}_2$. Fitting processes are called to attempt to match a primitive to the image. When it is determined that $\mathcal{O}_2$ is not primitive, more nodes are added to the "OPEN" list.

Step 3 and 4: The system selects nodes from OPEN to explore. Step 3 and 4 explored the child nodes of the root with associated transformations T_{REC} and T_{VEX}, respectively.

Step 5: The popped node from OPEN is a child of Node 1 and requires applying transformation T_{FAC}. Since no line pattern can be found in the contour image of $\mathcal{O}$, this transformation is not applicable. The node is simply eliminated.

Step 6 through 9: Four more nodes are eliminated from search tree.

Step 10: At this node, the compositional primitive pattern fitting error is ϵ_p =1.362E-3. According to the error estimation scheme introduced in Section 4.5, ϵ_n =4.0E-4. $\epsilon_p < 5 \times \epsilon_n$, thus the associated functional image is primitive. As such, a function form

[7] Refer to page pg:sys setting for the threshold setting.

hypothesis is formed and a verification procedure is called. By numerically inverting the transformation T_{DIF} and comparing the inverted image with $\mathcal{O}$, the computed matching error is found to be 1.148E-2, which is larger than ϵ_n. Thus, the hypothesis is rejected and the search is continued.

. . . . . .

Step 66: The system fits the associated functional image of this node to

$$z = 0.0001x^2 + 1.0008xy + 0.0001y^2,$$

with a fitting error of ϵ_n =1.335E-3, while the expected error of this node is ϵ_n =4.0E-4. Since $\epsilon_p < 5 \times \epsilon_n$, the image is primitive. By numeric verification, the function form matching error is found to be 1.476E-5, which is less than ϵ_n. Thus a solution is found and the system reports the discovery results as shown in Figure 5.2.

TASK refers to the data file name that was input as the observation data set. TERMINATION STATUS indicates one of the cases Success, Failure and Out of Memory. INPUT IMAGE provides basic information about the input. ORIGINAL PRECISION is the precision of the input believed by the user. REFERENCE IMAGE(S) gives the name of data files created by the system for extracting necessary descriptive expressions using a lower dimension function form discovery system. NODES shows the information concerning the search process, where Total stands for total nodes created, **Explored** is the number of nodes upon which primitive matching were conducted and **Open** is the number of nodes left in the OPEN list at the time of termination. In this case, there are in total 109 nodes that have been created. Among them, 21 nodes have been explored, 45 nodes are dead end nodes and 43 nodes are left unexplored. The meanings of the remaining five attributes are quite straightforward.

To get an explicit function expression, FFD is used to find the boundary expression as shown below, from the recorded data in the file "Example-1.B1.dat".

$$\begin{cases} x = 0 \\ z = 2. \end{cases}$$

So that we can invert the data transformation sequence step by step as shown in Table 5.2.

```
TASK:  Example-1
TERMINATION STATUS:  Success
INPUT IMAGE:  101 x 101,
       Xmin=-1, Xmax=1, Ymin=-1, Ymax=1
ORIGINAL PRECISION:  1.0E-12
REFERENCE IMAGE(S):
        1. File=./Example-1.B1.dat, Type=Boundary
NODES:  Total 109, Explored 21, Open 43
RUNNING TIME:  4'35.47"
MEMORY USAGE:  4.54MB
TRANSFORMATIONS:
        1. Diff(z,x)
        2. Factor(z/(y))
        3. Log(+z)
MATCHING PRIMITIVE FUNCTION:
       z=x*y
ERRORS:  Fitting:1.335e-3, Matching:1.476e-5
```

Figure 5.2: The Report Card for Example 1

In the table, the column "Trans" shows the transformation to be inverted, the column "Inverse" gives the expression to invert each transformation, and the column "Expression" gives the underlying function of the corresponding search node. Step 0 is the primitive function that was accepted as a match at Node 66. Step 1 and 2 invert two algebraic transformations. At Step 3, the differential transformation is inverted according to the extracted boundary expression. The discovered function is:

$$z = 1 + e^{xy},$$

which is exactly the underlying function (Equation 5.1) that has been used to generate the simulated observation data set.

Step	Trans.	Inverse	Expression
0	–	–	$z = xy$
1	`Log(+z)`	$(x, y, z) \Rightarrow (x, y, e^z)$	$z = e^{xy}$
2	`Factor(y)`	$(x, y, z) \Rightarrow (x, y, z * y)$	$z = ye^{xy}$
3	`Diff(z,x)`	$(x, y, z) \Rightarrow (x, y, (\int_0^x z\,dx)) + 2$	$z = e^{xy} + 1$

Table 5.2: Manual Inversion for Example 1

5.2.2 Example 2: Termination by a Primitive Pattern Fitting

The second function form to be discovered is

$$z = ye^{x+2y} + x + y. \tag{5.2}$$

The simulated observation data set is obtained by partitioning the observation domain $(x, y) \in [-1, 1; -1, 1]$ by a 101×101 mesh-grid. For this example, I will not give the details of the discovery process. Instead, I will focus on the discovered function form representation.

The system terminates with a discovered function form as reported in Figure 5.3. It is easy to verify the correctness of the transformation sequence with the following forward transformation steps:

1. Apply transformation "`Factor(x+2y)`" to the function $z = ye^{x+2y} + x + y$. We have:
$$z = \frac{ye^{x+2y} + x + y}{x + 2y}.$$

2. Then, apply transformation "`Dif(z,x)`". The generated function is:
$$z = \frac{y(x + 2y - 1)e^{x+2y} + y}{(x + 2y)^2}.$$

3. Finally, apply transformation "`Factor(y)`". We obtain the function:
$$z = \frac{(x + 2y - 1)e^{x+2y} + 1}{(x + 2y)^2}.$$

```
TASK:  Example-2
TERMINATION STATUS:  Success
INPUT IMAGE:  101 × 101,
       Xmin=-1, Xmax=1, Ymin=-1, Ymax=1
ORIGINAL PRECISION:  1.0E-12
REFERENCE IMAGE(S):
        1.  File=./Example-2.B7.dat, Type=Boundary
        2.  File=./Example-2.P156.dat,
                          Type=Primitive Pattern Image
NODES:  Total 239, Explored 95, Open 27
RUNNING TIME:  10'18.50"
MEMORY USAGE:  6.73MB
TRANSFORMATIONS:
        1.  Factor(x+2y)
        2.  Dif(z,x)
        3.  Factor(y)
MATCHING PRIMITIVE PATTERN:
        x+2y
ERRORS:  Fitting:1.5026e-04, Matching:6.8137e-05
```

Figure 5.3: The Report Card for Example 2

The final result is a function of $x + 2y$.

Now let us invert the discovered result. First, we need to find a descriptive expression corresponding to the primitive pattern. The underlying function is known as

$$z = \frac{(t-1)e^t + 1}{t},$$

where t is the new variable which is related to the original variables x and y as $t = x + 2y$ according to the primitive pattern fitting result. FFD is called to find a function form regarding to the sample set "`Example-2.P156.dat`". Unfortunately, no acceptable description could be found. We now have two choices. First, we can discard the discovered function form and let the system find a new one, or second, we can use polynomial fitting to find an acceptable description. Let us take the second choice. By fitting the recorded sample set to polynomial of order 6 using least-squares method, the following expression can be obtained with the fitting error of `5.81E-04`:

$$z = P(t) = 10^{-3}(500 + 335t + 125t^2 + 31.8t^3 + 6.7t^4 + 1.55t^5 + 0.219t^6).$$

Thus the matching function is:

$$\begin{aligned} z = P(x+2y) = \ & 10^{-3}\Big(500 + 335(x+2y) + 125(x+2y)^2 + 31.8(x+2y)^3 + \\ & 6.7(x+2y)^4 + 1.55(x+2y)^5 + 0.219(x+2y)^6\Big). \end{aligned}$$

The remaining inversion steps are similar to those in the first example. The discovered function is:

$$\tilde{z} = \left\{\int_0^x yP(x+2y)\mathrm{d}x + (e^{2y}+1)/2\right\}(x+2y),$$

where the expression $(e^{2y}+1)/2$ is obtained by calling FFD upon the recorded sample set `Example-2.B7.dat`. Obviously, it is not identical to the underlying function presented in Equation 5.2.

The discovered function is an approximation of the function of Equation 5.2. Let us now compare the discovered function form with traditional surface fitting method. There are in total 12 fitting parameters: 7 in the descriptive primitive polynomial, 3 to represent the two factors and 2 for the boundary expression. The root mean-square error of the approximation is 2.08-04. Using traditional 2-variable least-squares polynomial fitting scheme, the given observation data set can be represented by a fourth order 2-variable polynomial which contains fifteen parameters. However, the accuracy is very poor. The root mean-square error of the fitting is 0.27. Besides the accuracy, the function discovered by FFD-II is also more compact (using three fewer parameters) and more meaningful. For example, we can easily tell that 1) the underlying function equals to zero on the line $x + 2y = 0$, 2) the exponential relationship exists between z and y and 3) there is a hidden functional regularity of $y\frac{\partial z(x+2y)}{\partial x}$ being a function of $t = x + 2y$. All these properties are exactly the properties of the function defined by Equation 5.2, and they cannot be easily tell from the polynomial surface fitting result. This example shows that the function form discovery methods is superior to traditional numerical analysis methods in terms of justification, parsimony and transparency.

5.3 Randomly Selected Functions

5.3.1 A Random Function Form Generation Scheme

The data transformation based function form discovery mechanism enables the system to overcome the major restriction of handling only the discovery tasks of a fix number of function form prototypes. To set up test cases free of user's biases, a random scheme to choose test function is introduced in this section.

An explicit function expression can be represented by an expression tree whose leaves are the operands (independent variables or constants) and non-leaf nodes are operators.

Definition 23 *An* Operator *is an unary operator or a binary operator. An* Unary Operator *is any functional operation in set $\mathcal{S}_F$:*

$$\mathcal{S}_F = \{\, -(\star),\ 1/(\star),\ \sqrt{\star},\ (\star)^2,\ \exp(\star),\ \log(\star),\ \tan(\star),\ \arctan(\star)\,\} \tag{5.3}$$

where $\star$ stands for a functional expression. A Binary Operator *is any arithmetic operation in the set $\mathcal{S}_A$:*

$$\mathcal{S}_A = \{\, +,\ \times\,\} \tag{5.4}$$

An Operand *is either an operator or an end operand. An* End Operand *is one of the element in set* $\mathcal{S}_v$*:*

$$\mathcal{S}_v = \{ 1, x , y \} \tag{5.5}$$

An Expression Atom *is an element in the set* $\mathcal{S}$*:*

$$\mathcal{S}_E = \mathcal{S}_F \cup \mathcal{S}_A \cup \mathcal{S}_v \tag{5.6}$$

The function forms to be discovered in the experiments are generated by the following recursive algorithm.

```
ConstructExpressionTree()
1: Randomly select an expression atom Atom $\in \mathcal{S}_E$
2: Construct a tree node data structure Root and assign the content[8] of Root
   with Atom selected in step 1.
3: ExpandNode(Root)
4: return Root

ExpandNode(node)
1: Atom=node.atom
2: if Atom $\in \mathcal{S}_F$
4:     Randomly select an atom NewAtom $\in \mathcal{S}_E$
5:     Construct a tree node data structure NewNode. Let NewNode.atom=NewAtom,
       NewNode.parent=node, and node.lChild=NewNode
6:     ExpandNode(NewNode)
7: else if Atom $\in \mathcal{S}_A$
8:     Randomly select two atoms lAtom, rAtom$\in \mathcal{S}_E$
9:     Construct two tree node data structures lNode and rNode. Let
       lNode.atom=lAtom, lNode.parent=node, node.lChild=lNode,
       rNode.atom=rAtom, rNode.parent=node, node.rChild=rNode
```

[8] The expression tree node data structure contains four fields, node.parent, node.lChild, node.rChild and node.atom.

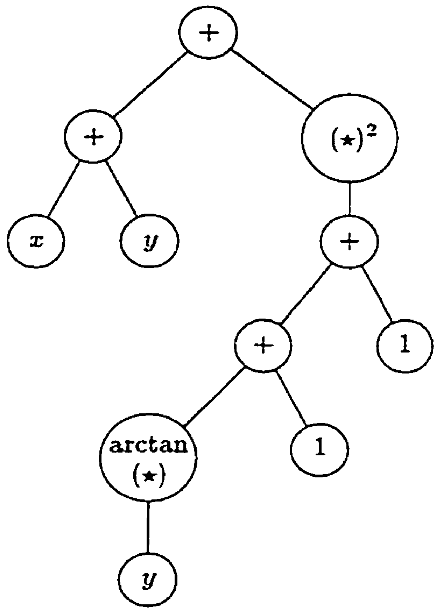


Figure 5.4: An Expression Tree of a Function Form

```
10:       ExpandNode(lNode)
11:       ExpandNode(rNode)
12:    return
```

The above pseudo-code represents the core of the algorithm. A complete algorithm must also includes a set of rules to remove redundancy and be able to terminate within a specified maximum depth. The maximum depth of an expression tree can be limited by restricting the selection of operand at a certain depth within $\mathcal{S}_v$. An example of redundant expression tree is the tree that has a node with the associated operator 'exp($\star$)', and the associated operator of its child node is 'log($\star$)'. Such redundancy also occurs when placing 'arctan($\star$)' immediately under 'tan($\star$)' (or vice versa), '$\sqrt{(\star)}$' immediately under '$(\star)^2$' (or vice versa) and '$-(\star)$' or '$1/(\star)$' immediately under itself. Figure 5.4 is an example of expression tree that represents the first test function in Table 5.3.

This algorithm can generate many of the explicit two dimensional analytic functions that can be found in a first year mathematics text book. Table 5.3 listed the first 18 functions generated by the algorithm with depth limited to six.

5.3.2 Experimental Results

Experiments were carried out to discover the function forms listed in Table 5.3. Observation domain of each task is kept as coincident as possible whenever the continuity is conserved (refers to Table 5.3). Partitioning mesh grid size were all fixed to 101×101. Unless reported otherwise, we selected $\epsilon_{\mathcal{P}} = \max(5 \times \epsilon_n, 10^{-5})$ and $\epsilon_{\mathcal{M}} = \max(\epsilon_n, 10^{-5})$, as we discussed in the beginning of this chapter.

Out of the 18 discovery tasks, FFD-II successfully discovered 17 solutions. The only failure was Task #9. The discussion of that case will be postponed to the end of this chapter. It must be pointed out that a successful discovery does not have to be in exactly the same form as given in Table 5.3. As stated in the problem statement, the goal of function form discovery is to discover a function form representation of the given numeric observation data set that satisfies the preset error tolerance threshold.

Table 5.4 tabulated the information of each discovery task. The column '*Solution*' shows the type of termination of the task, where '*exact*' means that a transformation sequence and primitive form leading to a function identical to the original underlying function used to generate the simulated observation data set (provided the necessary descriptive expressions can be obtained by other means)[9]. To see if a solution is exact, we can manually carry out the data transformation to the known underlying function or invert the transformation sequence starting from the matching primitive. However, to revert the discovered forms to the original functions, all necessary descriptive expressions must be figured out first.

The column '*Total Nodes*' is the total number of nodes generated during the discovery process, '*Explored*' is the number of nodes that had been expended and corresponding

[9] Notice that the discovery of the function form identical to the known underlying function is a sufficient but not necessary condition to test for the correctness of the discovered form. As long as the discovered solution represent the given observation data sufficiently well, in terms of Justification, Parsimony and Transparency, it is a correct solution.

Task #	Function	Observation Domain
1	$z = y + x + (2 + \arctan y)^2$	$[-1, 1;\ -1, 1]$
2	$z = \frac{y^2}{x} e^{3y}$	$[0.1, 1;\ -1, 1]$
3	$z = y((4x + y)^2 + (1 + y)^2)$	$[-1, 1;\ -1, 1]$
4	$z = \frac{y^2}{2x + 1} + 2y + 1$	$[0, 1;\ -1, 1]$
5	$z = ye^{x+2y} + x + y$	$[-1, 1;\ -1, 1]$
6	$z = y + x^2 + e^{y(x+1)}$	$[-0.5, 1.5;\ -0.5, 1.5]$
7	$z = 2y + 1 + x \tan \frac{\pi}{2}(2y + 1)$	$[-1, 1;\ -0.9, -0.1]$
8	$z = y + 2x + 3 + x\sqrt{2x + 2y + 3}$	$[-0.75, 2.75;\ -0.75, 2.75]$
9	$z = 2y + \tan \frac{\pi}{2}(2x + y) + \log(2y + x + 1)$	$[-0.45, 0.45;\ -0.25, 0.25]$
10	$z = 2x + 2y + \log(y + 3)$	$[-1, 1;\ -1, 1]$
11	$z = y + x \log(2 + y) + \arctan(x)$	$[-1, 1;\ -1, 1]$
12	$z = (2 + e^x)(1 + e^y)$	$[-1, 1;\ -1, 1]$
13	$z = x \log(y(2x + 3)^2)$	$[-1, 1;\ 1, 3]$
14	$z = 2 + e^{xy}/y$	$[-1, 1;\ 0.2, 2.2]$
15	$z = 1 + x + e^{2y(y+2x+2)}$	$[-1.3, 0.7;\ -1.3, 0.7]$
16	$z = (x + \frac{1}{y + 1}) \log(y + y^2)$	$[-1, 1;\ 1, 3]$
17	$z = \frac{1 + 2x}{\sqrt{x^2 + 2y^2}}$	$[-1, 1;\ 1, 3]$
18	$z = x + \log(x + 2y + 1)^2$	$[0, 2;\ 0, 2]$

Table 5.3: Random Selected Test Functions and Observation Domains

Task ID.	**Solution**	**Total Nodes**	**Ex-plored**	**Open**	**Expected Error**	**Fitting Error**	**Matching Error**
1	exact	13	3	3	4.00e-04	2.56e-12	1.87e-15
2	exact†	154	28	42	2.10e-04	6.84e-08	2.53e-03
3	exact	17	3	3	4.00e-04	8.19e-12	3.45e-04
4	exact	109	32	23	4.50e-04	9.88e-08	8.03e-06
5	exact	201	55	60	4.00e-04	2.28e-04	9.19e-05
6	exact	330	113	47	6.24e-04	5.00e-05	4.21e-04
7	exact	13	3	3	1.96e-04	1.82e-07	2.07e-04
8	exact	175	49	54	2.45e-03	7.03e-05	1.77e-05
9	none	432	139	0	-	-	-
10	exact	19	4	9	4.00e-04	9.89e-13	4.94e-16
11	exact	173	45	57	8.00e-04	4.47e-03	1.79e-05
12	exact	89	16	39	4.00e-04	2.31e-07	7.35e-06
13	exact	71	20	16	8.00e-04	6.42e-08	5.69e-06
14	exact	31	6	20	1.82e-04	7.08e-10	1.26e-05
15	exact‡	223	47	71	2.37e-04	5.75e-03	1.24e-02
16	exact	13	3	3	4.00e-04	2.28e-07	9.72e-06
17	exact	61	11	34	1.00e-12	5.30e-06	6.34e-18
18	exact	13	3	3	4.00e-04	3.24e-06	1.30e-05

†: $\epsilon_{\mathcal{M}}$ was increased to $10 \times \epsilon_n$.

‡: $\epsilon_{\mathcal{P}}$ was increased to $20 \times \epsilon_n$ and $\epsilon_{\mathcal{M}}$ was increased to $100 \times \epsilon_n$.

Table 5.4: Discovery Results of Experiment 1

primitive fitting had been performed, '*Open*' is the number of nodes left in the open list at the time of termination. These three numbers reflect the effort of searching for the solution. The column '*Expected Error*' is the computed expected error level estimations of the functional image associated with the termination node, '*Fitting Error*' is the primitive fitting error of the terminate node, and '*Matching Error*' is the verified error of the solution through numerical inversion. The order of the expected errors successfully bound the fitting errors. Also, the matching error for a discovered accurate function forms are mostly less than 10^{-4}.

Table 5.5 tabulated the solutions discovered by **FFD-II**. In two tasks, Task #2 and #15, the first attempts at solving the task with the common thresholds setting ended without solutions. Increased thresholds enabled the system to find the exact solutions. By carefully observing the underlying functions and the discovered transformation sequences of these two cases, it could be easily determined that the problems came from the inaccurate computing of T_{REC}, $T_{\text{FAC}}|\underline{x+y+1}$ combined with T_{DIF}. They suggest two future improvements: (1) more accurate estimation of the propagated errors so that ill points[10] could be identified; (2) development of new computing schemes that compute the transformed image around those ill points more accurately.

Since the underlying function forms are all known, it is easy to figure out the corresponding boundary conditions required for the differential transformations to be one-to-one mappings, and the matching primitive expressions associated with the matching primitive patterns (if applicable) by manually applying the discovered transformations to the original functions[11]. Table 5.6 tabulated the accurate underlying descriptive expressions[12] where

[10] Ill points are those points where the propagated errors might be extremely great, for example, the points where the values of the factor are close to zero when performing factorization, and the points where the function values are close to zero when performing reciprocal are two types.

[11] An example can be found in the analyses of the second detailed example presented in Section 5.2.2 on page 164. Refer also to the example on page 54 and Definition 17 on page 56 for more details concerning descriptive expression.

[12] Notice that the discovered function form for Task#17 does not include descriptive expressions, and there is no solution for Task#9 discovered by **FFD-II**.

Task ID.	Transformation Sequence	Matching Primitive
1	$\mathcal{T}_{\text{DIF}}$	$\mathcal{F}$: $w = 1$
2	$\mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{INV}} \circ \mathcal{T}_{\text{REC}} \circ \mathcal{T}_{\text{INV}}$	$\mathcal{P}$: v
3	$\mathcal{T}_{\text{DIF}}$	$\mathcal{F}$: $w = 32uv + 8v^2$
4	$\mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}}$	$\mathcal{P}$: u
5	$\mathcal{T}_{\text{FAC}}\|_{\underline{v}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{FAC}}\|_{\underline{u+2v}}$	$\mathcal{P}$: $u + 2v$
6	$\mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{LOG}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}}$	$\mathcal{F}$: $w = v + 1$
7	$\mathcal{T}_{\text{DIF}}$	$\mathcal{P}$: v
8	$\mathcal{T}_{\text{FAC}}\|_{\underline{v}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}}$	$\mathcal{P}$: $u + v$
9	-	-
10	$\mathcal{T}_{\text{DIF}}$	$\mathcal{F}$: $w = 2$
11	$\mathcal{T}_{\text{FAC}}\|_{\underline{v}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{REC}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}}$	$\mathcal{P}$: $u + v$
12	$\mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{LOG}}$	$\mathcal{P}$: u
13	$\mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}}$	$\mathcal{P}$: u
14	$\mathcal{T}_{\text{LOG}} \circ \mathcal{T}_{\text{DIF}}$	$\mathcal{F}$: $w = uv$
15	$\mathcal{T}_{\text{LOG}} \circ \mathcal{T}_{\text{FAC}}\|_{\underline{u+v+1}} \circ \mathcal{T}_{\text{DIF}} \circ \mathcal{T}_{\text{VEX}}$	$\mathcal{F}$: $w = 2u^2 + 4uv + 4u + \log 4$
16	$\mathcal{T}_{\text{DIF}}$	$\mathcal{P}$: v
17	$\mathcal{T}_{\text{REC}} \circ \mathcal{T}_{\text{FAC}}\|_{\underline{2u+1}}$	$\mathcal{F}$: $w^2 = u^2 + 2v^2$
18	$\mathcal{T}_{\text{DIF}}$	$\mathcal{P}$: $u + 2v$

Table 5.5: Solutions of Experiments on Randomly Generated Functions Discovered by FFD-II

each box bracketed expression pair refers to a boundary condition corresponding to a differential transformation and each $P(t)$ represents the descriptive expression corresponding to the discovered compositional primitive. As a function form discovery system, FFD-II must be able to provide these descriptive expressions as part of the solutions or a way to find them. Currently, FFD-II does not automatically give the descriptive expressions. Instead, it saves the necessary two-dimensional data for finding the expressions to disk files as the task terminates with a successful discovery.

To generate full functional representation of the given observation, function form discovery from two variables must be carried out upon the saved two-dimensional data. These tasks are non-dominant subtasks (Refer to Section 4.2) which means that they can be separated from the original discovery task in higher dimension and they do not change the structure of the function form description (the transformation sequence and the matching primitive) discovered by FFD-II but only complete the function form description. In other words, whether or not the discovery is successful is decided by the output of FFD-II before the descriptive expressions are figured out[13]. Therefore either traditional numeric tools or two-variable function form discovery systems can be used to handle the task of find the descriptive expressions. The selected method is referred to as "supporting system".

However, since function form discovery systems emphasize the discovery of high quality function form descriptions in terms of the justification, parsimony and transparency, it is better to choose a two-variable function form discovery system to carry out the descriptive expression discovery tasks. The data transformation based function form discovery system FFD is one of the best choice due to its ability to discover a wide variety of one-dimensional functions. The last column in Table 5.6 indicates the results of performing one-dimensional function form discovery upon the recorded two-dimensional data using FFD. Of the seventeen function form descriptions, FFD successfully discovered the accurate descriptive

[13] This is the essential difference between the direct model of FFD-II and a parameter freezing based indirect model. In the latter model, a successful discovery depends on both the successful discovery of low dimension expressions and whether or not the discovered low dimension expressions could be successfully combined into a single function form description to describe the original discovery problem.

Task ID.	Descriptive Expression	Discovered by FFD
1	$[u=0,\ f=v+(2+\arctan(v))^2]$	none
2	$[u=0.45,\ f=v^2e^{3v}]$, $P(t)=t^2e^{3t}$	exact
3	$[u=0,\ f=v(v^2+(1+v)^2)]$	exact
4	$[u=0,\ f=1]$, $[u=0,\ f=2]$	exact
	$P(t)=2/(2t+1)$	exact
5	$[u=0,\ f=(e^{2v}+1)/2]$	exact
	$P=((t-1)e^t+1)/t^2$	none
6	$[u=0,\ f=v^2+1]$, $[u=0,\ f=u+2]$,	exact
	$[u=0,\ f=2\log(u+1)]$	
7	$[u=0,\ f=1+2v+\tan(\frac{\pi}{2}(2v+1))]$,	exact
	$P(t)=\tan(\frac{\pi}{2}(2t+1))$	
8	$[u=0,\ f=2v+v\sqrt{2v+3}]$, $[u=0,\ f=1+v/\sqrt{2v+3}]$	exact
	$P(t)=-1/(2t+3)^{3/2}$	
10	$[u=0,\ f=2v+\log(v+3)]$	exact
11	$[u=0,\ f=(\log 2)u+\arctan(u)]$, $[u=0,\ f=2/(2+u)]$,	exact
	$P(t)=1/(2+t)^2$	
12	$[u=0,\ f=\log 3(1+e^v)]$, $P(t)=e^t/(2+e^t)$	exact
13	$[u=2,\ f=v\log(2(2v+3)^2)]$, $[u=0,\ f=0]$, $P(t)=1/t$	exact
14	$[u=0,\ f=2+1/v]$	exact
15	$[u=0,\ f=2+v]$	exact
16	$[u=0,\ f=\frac{1}{v+1}\log(v+v^2)]$	none
	$P(t)=\log(t+t^2)$	exact
18	$[u=1,\ f=1+2log(2v+2)]$, $P(t)=1+2/(t+1)$	exact

Table 5.6: Descriptive Expressions of Experiment 1

expressions in thirteen cases and failed to find acceptable function forms to describe the recorded sample data in three cases.

When FFD failed to find the description expression, least squares polynomial fitting was used as an alternative. Although polynomial fitting results usually do not have good interpretability, the method is very reliable. In principle, we can increase the fitting accuracy by increasing the degree of the fitting polynomial. Thus, for reserving the accuracy of the discovered function form, the degree of the fitting polynomial, in each case, was selected according to the estimated expected error, the primitive fitting error and the verification error of the original function form description discovered by FFD-II. The lowest degree polynomial were chosen for finding the descriptive expression, such that the polynomial fitting error being smaller than the primitive fitting error, the verified matching error or the estimated expected error of the corresponding discovered solution.

Complete function form descriptions are tabulated in Table 5.7. All the extracted boundary conditions corresponding to the differential transformations are given by the subscripts of each data transformation, and the extracted descriptive expressions corresponding to each of the primitive patterns are transformed into functional formats. For the tasks #1, #5 and #16, polynomial fitting results are used as the descriptive expressions where FFD failed to discover them.

When FFD is used as the supporting one-dimensional function form discovery system, out of the eighteen function form discovery tasks listed in Table 5.3 and 5.4, FFD-II successfully discovered fourteen (78%) accurate functional expressions identical to the original underlying functions[14]. With the help of polynomial fitting, another three approximations were found (Task #1, #5 and #16). For one case, Task #9, FFD-II failed to find an acceptable description of the given observation data.

Traditional numeric analysis tools, such as surface fitting, usually cannot extract the accurate underlying function to describe the given observation data without sufficient knowl-

[14] Verified by symbolically inversion of the data transformations in the discovered transformation sequence starting from the matching primitive. Examples of the inversion have been presented in Section 3.2.1 on page 54 and Section 5.2 on page158.

ID.	Transformation Sequence	Matching Primitive $p(u,v)$ ($w = p(u,v)$)
1	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=4+5v+0.98v^2-1.29v^3-0.52v^4+0.59v^5+0.16v^6-0.16v^7}}$	1
2	$T_{\text{DIF}}\big\|_{\underline{u=0.45,\ f=v^2e^{3v}}} \circ T_{\text{INV}} \circ T_{\text{REC}} \circ T_{\text{INV}}$	v^2e^{3v}
3	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=v(v^2+(1+v)^2)}}$	$32uv + 8v^2$
4	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=1}} \circ T_{\text{VEX}}$	$2/(2u+1)$
5	$T_{\text{FAC}}\big\|_{\underline{v}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=(e^{2v}+1)/2}} \circ T_{\text{FAC}}\big\|_{\underline{u+2v}}$	$10^{-3}\{500 + 335(u+2v) + 125(u+2v)^2 + 31.8(u+2v)^3 + 6.7(u+2v)^4 + 1.55(u+2v)^5 + 0.219(u+2v)^6\}$
6	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2\log(u+1)}} \circ T_{\text{LOG}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=u+2}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=v^2+1}} \circ T_{\text{VEX}}$	$v+1$
7	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=1+2v+\tan(\frac{\pi}{2}(2v+1)}}$	$\tan(\frac{\pi}{2}(2v+1))$
8	$T_{\text{FAC}}\big\|_{\underline{v}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=1+v/\sqrt{2v+3}}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2v+v\sqrt{2v+3}}} \circ T_{\text{VEX}}$	$-1/(2u+2v+3)^{3/2}$
10	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2v+\log(v+3)}}$	2
11	$T_{\text{FAC}}\big\|_{\underline{v}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2/(2+u)}} \circ T_{\text{REC}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=(\log 2)u+\arctan(u)}} \circ T_{\text{VEX}}$	$1/(2+u+v)^2$
12	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=\log 3(1+e^v)}} \circ T_{\text{LOG}}$	$e^u/(2+e^u)$
13	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=0}} \circ T_{\text{VEX}} \circ T_{\text{DIF}}\big\|_{\underline{u=2,\ f=v\log(2(2v+3)^2}} \circ T_{\text{VEX}}$	$1/u$
14	$T_{\text{LOG}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2+1/v}}$	uv
15	$T_{\text{LOG}} \circ T_{\text{FAC}}\big\|_{\underline{u+v+1}} \circ T_{\text{DIF}}\big\|_{\underline{u=0,\ f=2+v}} \circ T_{\text{VEX}}$	$2u^2 + 4uv + 4u + \log 4$
16	$T_{\text{DIF}}\big\|_{\underline{u=0,\ f=-0.84+2.06v-1.14v^2+2.94v^3-0.03v^4}}$	$\log(v+v^2)$
17	$T_{\text{REC}} \circ T_{\text{FAC}}\big\|_{\underline{2u+1}}$	$z^2 = u^2 + 2v^2$
18	$T_{\text{DIF}}\big\|_{\underline{u=1,\ f=1+2log(2v+2)}}$	$1 + 2/(u+2v+1)$

Table 5.7: Discovered Function Form Descriptions of Experiments on Randomly Generated Functions

edge of mathematicians and domain experts. However, in contrast, the test results show that in many cases (in which the known accurate underlying functions are discovered) the discovery system FFD-II has the ability to find compact and meaningful function forms which describe the given data accurately. When an approximation is discovered, however, the superiority of the result of function form discovery is not that explicit. In Section 5.2.2, I have analyzed the approximated form of case 5 and concluded that the proposed function form discovery methodology surpasses the traditional polynomial surface fitting method in that particular case. Now let us compare the proposed function form discovery methodology with polynomial fitting method in dealing with Task #1 and #16. The discovered function forms of those two cases can be transfered to explicit functions through manually inverting the transformation sequences:

Case 1:

$$\begin{aligned} z &= 4 + x + 5y + 0.98y^2 - 1.29y^3 - 0.52y^4 + \\ &\quad 0.59y^5 + 0.16y^6 - 0.16y^7, \end{aligned} \tag{5.7}$$

Case 16:

$$\begin{aligned} z &= -0.84 + x\log(y + y^2) + 2.06y - 1.14y^2 + \\ &\quad 2.94y^3 - 0.03y^4. \end{aligned} \tag{5.8}$$

There are eight fitting parameters in Equation 5.7. Using two-variable polynomial fitting, we can fit the observation data set of Task #1 to the following six-parameter polynomial:

$$\begin{aligned} z &= 4.03 + x + 4.9y + (1.47\text{E-}3)x^2 + (2.05\text{E-}5)xy + \\ &\quad (6.43\text{E-}1)y^2 + (3.79\text{E-}5)x^3 + (4.97\text{E-}5)x^2y + \\ &\quad (4.97\text{E-}5)xy^2 - (7.94\text{E-}2)y^3. \end{aligned} \tag{5.9}$$

Observing Equation 5.9, one might guess that the terms x^2, xy, x^2y, xy^2 and x^3 are zero and the coefficient of the term x is 1 because the corresponding coefficients are very small or close to 1. Therefore, it is reasonable to refit the data to a new polynomial suggested by the observation of Equation 5.9. The result of the new selected fitting scheme might generate the result identical to Equation 5.7. It is true that the traditional polynomial fitting method does similarly well in handling this task. However, unlike surface fitting, FFD-II successfully discovered that the underlying function is of the form $z = x + g(y)$ and correctly set up the subtask for discovering the unknown function $g(y)$ without any human intervention.

Equation 5.8 contains five fitting parameters. We can fit the observation data set to the second order two-variable polynomial which contains eight fitting parameters. Least-squares fitting gives:

$$\begin{aligned} z &= -(6.06\text{E-}2) - (1.42\text{E-}2)x + (5.48\text{E-}1)y + \\ &\quad (4.33\text{E-}5)x^2 + (8.70\text{E-}1)xy - (1.09\text{E-}1)y^2. \end{aligned} \tag{5.10}$$

The comparisons of the two approximations (Equation 5.8 and Equation 5.10) can be summarized as below:

- The root-mean-squared errors of the two representations are 3.7E-2 for polynomial surface fitting and 5.6E-4 for the discovered result. It indicates that the function form discovery result is more accurate than the polynomial surface fitting method.
- The root-mean-squared errors of the first order partial derivatives of the two approximations are 6.2E-2 for polynomial surface fitting and 1.7E-3 for the discovered result. The results indicate that the function form discovery result captures the shape of the given functional image significantly better than the polynomial surface fitting method.
- Equation 5.8 contains one less fitting parameter than Equation 5.10.
- Equation 5.8 is easy to interpret than Equation 5.10. For example, from Equation 5.8, it is easy to tell that:
  - For each fixed y value, the function value changes linearly to the change of variable x.
  - The above changing rate of the dependent variable is related to the value of variable y logarithmically.
  - The underlying function cannot be defined in the range where

$$y + y^2 \leq 0.$$

– When $x \neq 0$, $z \to -\infty$ as $y + y^2 \to +0$.

These pieces of information are consistent with the known underlying function, but they are not revealed by Equation 5.10.

For the analyzed cases, the function form discovery result generated by FFD-II surpasses polynomial surface fitting result with regard to justification, parsimony and transparency.

5.4 Comparison Experiments

5.4.1 The Comparison Discovery System

It has been pointed out that all function form discovery methodologies not in the "Data Transformation" category have a common drawback. They can discover only the function forms in a very small number of function form classes, i.e. either rational functions or a fixed set prototypes. Thus it is not meaningful to compare FFD-II with any method in that group. The comparison should be made between FFD-II and a system that can handle a rich set of three-variable function form discovery tasks. Unfortunately, there is no such a system in existence. However, FFD has an extension that can handle a special type of multi-variable function form discovery tasks, namely families of one-dimensional functions parameterized by a few parameters. It is required that the function value change relatively slowly with the change in the parameter value than with the change in the independent variable.

The underlying discovery strategy for this extension is parameter freezing — a classic indirect technique that has also been used by the BACON system. The parameter freezing approach could be viewed as an indirect approach to three-variable function form discovery. Recall that the current FFD family of functions discovery system finds parametric expressions only by primitive fitting. For conducting the comparison, the simplification assumptions made by FFD family of functions discovery extension (Primitive Union and Simple Descriptive Parameters) are relaxed. In other words, when FFD finds the solutions to the subtasks of one-dimensional function form discovery (generated by putting one of the two independent variable on hold) we assume that:

(1) The discovered solutions to the subtasks are unifiable provided the correct parametric expressions are identified.

(2) For identifying the necessary parametric expressions, regarding to each corresponding parametric expression, the parameter values (the data) could be correctly collected by hand.

(3) FFD could be used to find those parametric expressions upon properly organized data.

The further extended indirect data transformation based three-variable function form discovery method will be called INDIRECT-FFD in the discussion of this part.

I have explained how the indirect system works and discussed some drawbacks of that approach in Section 4.2.1. I have also discussed why a direct approach model may generally perform better than an indirect approach and why a direct model approach is necessary and important. However, since FFD-II employs only a very small transformation set and recognizes only the simplest primitives, it is not guaranteed to discover function forms that are discoverable to INDIRECT-FFD . Due to the rich variety of two-dimensional functions, in certain situation, an indirect method could be the best to solve the discovery problem. In this part, I will focus on studying those cases that require the discovery system to use the "cross effect" information for making a successful discovery.

Three specially designed discovery tasks will be investigated. They are corresponding to the classes "`FFT-Class`", "`FVS-Class`" and "`ICL-Class`" respectively, as named in Section 4.2.1. In each experiment, I will first describe the reasons why INDIRECT-FFD fails to discover the correct solution. And then the discovered results made by FFD-II will follow the explanations.

The reason for not studying the class "`USA-Class`" is that it is closely related to the implementation of the indirect methodology. Designing a case that belongs to the "`USA-Class`"

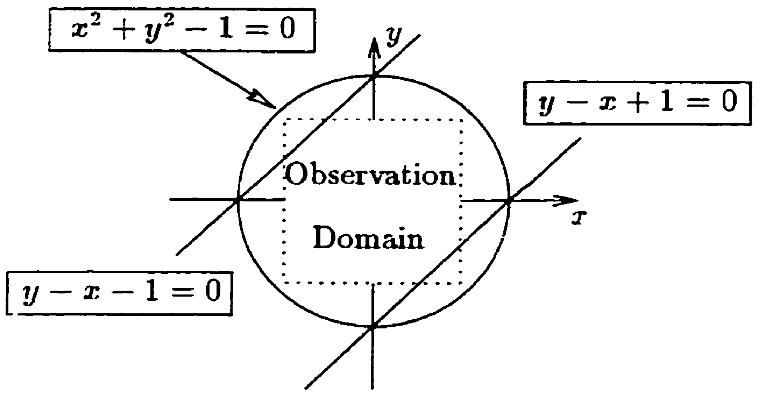


Figure 5.5: Patterns in 0-Contour Plane

of FFD family of functions discovery extension is easy but not very meaningful. To compare with INDIRECT-FFD in this direction is not practical since INDIRECT-FFD is not subjected to any constraints. However, in practice, we cannot always figure out a way to unify one dimensional results of parameter frozen subtasks, especially when the estimated parameter values are not sufficiently accurate. It is indeed the major difficulty for implementing a general purpose indirect multi-variable function form discovery system. In other words, certain types of simplification assumptions are unavoidable for an indirect implementation. Thus without specific implementation, we cannot talk about "USA-Class".

5.4.2 Case Study 1: An FFT-Class Function Form

The first function form to be examined is

$$z = \frac{(y-x)^2-1}{\sqrt{1-x^2-y^2}}. \tag{5.11}$$

This is a second class function form that INDIRECT-FFD will have trouble to deal with. There are two linear factors $y-x-1$ and $y-x+1$ in the underlying function form. And the observation domain is restricted by the circle $1-x^2+y^2=0$. Figure 5.5 shows the circle and two lines. Let us assume that the observation data set is obtained by partitioning the range $x \in [-0.6, 0.6], y \in [-0.6, 0.6]$ into a 101×101 mesh grid. It has been proved (Phan [48]) that the factorization transformation is essential for handling the discovery of rational functions with a data transformation based function form discovery system. To discover the function form of Equation 5.11, there are two factors that must be removed by factorizations. Obviously the two factors cannot be observed from any single sample data set with a fixed x or y value. Since INDIRECT-FFD approaches the discovery problem in an indirect way, each subtask is carried out individually, and it cannot combine the two factors observed from different sample sets together to form a unification transformation. Thus this function form cannot be discovered. The system was tested with the above observation data. It failed to discover any solutions to any sample data set. That means, without removing both factors by factorization transformation, the functional image cannot be simplified into any primitive form by the system.

By taking direct approach, FFD-II can extract hypotheses based on the information gained from all parts of the observation domain. In this study case, the system first finds from the original given observation image a set of planar points where the underlying function has the function value of zero (use interpolation if necessary). In the next and last step, it conjectures the factor functions by fitting the obtained contour points into lines. Two factorizations are successfully performed and the underlying function is discovered. Figure 5.6 is the report card generated by FFD-II system upon the discovery of function form (5.11).

The ability to capture the cross-effects is important in performing high dimension pattern recognition tasks. A direct model achieves this ability as an essential. This study case demonstrates how the system creates hypotheses with cross-reference.

5.4.3 Case Study 2: An FVS-Class Function Form

The second comparison test experimental function is

$$z = \sqrt{x} + \sqrt{x^2+1}\log y - \log^2 y \tag{5.12}$$

and the observation data set is generated within the domain $x \in [0,1]$, $y \in [1,2]$ with a 101×101 mesh grid. This is an FVS-Class function. As usual, we first let INDIRECT-FFD handle the task.

```
TASK:  Comparison-1
TERMINATION STATUS:  Success
INPUT IMAGE:  101 × 101,
        Xmin=-0.6, Xmax=0.6, Ymin=-0.6, Ymax=0.6
ORIGINAL PRECISION:  1.0E-12
REFERENCE IMAGE(S):
         None
NODES:  Total 65, Explored 12, Open 33
RUNNING TIME:  1'12.72"
MEMORY USAGE:  7.11MB
TRANSFORMATIONS:
         1.  Fact(z/(-x+y+1))
         2.  Fact(z/(-x+y-1))
         3.  Reciprocal
MATCHING PRIMITIVE FUNCTION:
         z^2=-x^2-y^2+1  (+)
ERRORS:  Fitting:3.73e-05, Matching:1.5758e-16
```

Figure 5.6: The Report Card for Comparison Test 1

If we put variable y on hold, INDIRECT-FFD first must find a set of one-variable functions

$$z = \sqrt{x} + \phi_1^i \sqrt{x^2+1} + \phi_2^i$$

where i corresponding to the sample data sets indexed by $y_i = 1 + 0.01i$, and ϕ_1^i and ϕ_2^i are descriptive parameters. Unfortunately, testing shows that the system cannot successfully discover any of them. These function forms cannot be easily simplified by the transformations in the system's tool box. Thus freezing y is not an successful choice.

Now let us assume that variable x is held as the control parameter. For those sample data sets with $x^i > 0.96$ (four samples corresponding to the parameter values $x = 0.97, 0.98, 0.99$, and 1.0, INDIRECT-FFD can find a transformation sequence $\Theta \circ \Lambda \circ \Theta$ that transforms the original samples into an uniform primitive form

$$z = y^2 + \phi_1^i y + \phi_2^i$$

and there are no solution found for the remaining samples. The system then tries to verify the obtained similarization transformation sequence with all the samples. Since the transformation Θ, which can only be applied to monotonic sample data set, is not applicable to a majority of the samples as the first transformation, FFD discards the hypothesis and tries to find other ways to get a solution. In the test conducted, there are no more solutions to any samples the system could find. Thus the system terminates without a discovery.

Since the new system performs three dimensional transformations, it can capture more varieties of fundamental features provided by the observation than indirect approaches. This ability is demonstrated in this experiment. Although the underlying function is not monotonic to variable y, a transformed functional image meets the requirement. This ability enabled the system to extract the key transformation sub-sequence $T_{\text{INV}} \circ T_{\text{LOG}} \circ T_{\text{INV}}$ which transforms an original independent variable into its logarithm. Figure 5.7 shows the correct solution found by FFD-II.

The differential transformation extracts the functional pattern of the differences between adjacent sample data sets indexed by x values. In this case, monotonic image is obtained.

```
TASK: Comparison-2
TERMINATION STATUS: Success
INPUT IMAGE: 101 × 101,
      Xmin=0.0, Xmax=1.0, Ymin=1.0, Ymax=2.0
ORIGINAL PRECISION: 1.0E-12
REFERENCE IMAGE(S):
       1. File=./Comparison-2.B1.dat, Type=Boundary
       2. FILE=./Comparison-2.B214.dat, Type=Boundary
       3. File=./Comparison-3.P325.dat,
                         Type=Primitive Pattern Image
NODES: Total 497, Explored 94, Open 197
RUNNING TIME: 26'06.01"
MEMORY USAGE: 7.31MB
TRANSFORMATIONS:
       1. Dif(z,x)
       2. Variable Exchange (x<>y)
       3. Functional Inverse (x<>z)
       4. Log(+z)
       5. Functional Inverse (x<>z)
       6. Dif(z,x)
MATCHING PRIMITIVE PATTERN:
       x
ERRORS: Fitting:4.87e-04, Matching:1.5758e-05
```

Figure 5.7: The Report Card for Comparison Test 2

Three descriptive expressions required for completing the discovery are $\log^2 x$, $1/\sqrt{x}$ and $x/\sqrt{(x^2+1)}$. All can be discovered by the two-variable FFD system.

5.4.4 Case Study 3: An ICL-Class Function Form

The last comparison test is to discover the function

$$z = \log(x) + \sqrt{xy} + \tan(y). \qquad (5.13)$$

It belongs to the fourth function class that INDIRECT-FFD discovery system cannot handle. Observation is made with the mesh grid 101×101 that evenly partitions the observation domain $x \in [0.5, 1.5]$, $y \in [0.5, 1.5]$. To find the underlying function form, INDIRECT-FFD must find expressions of x

$$z = \log(x) + \sqrt{y^i}\sqrt{x} + \tan(y^i)$$

with at least some y^i (y be the chosen control parameter), or find some expressions of y

$$z = \tan(y) + \sqrt{x^i}\sqrt{y} + \log(x^i)$$

with at least some x^i (x be the chosen control parameter). Unfortunately, no such subtasks could be solved by the system in the conducted tests. What happens is that the linear combination of the terms $\log(x)$ and $\sqrt{x}$, or $\tan(y)$ and $\sqrt{y}$ is beyond the system's discovery ability, since none of the transformations defined in FFD 's tool-box can effectively simplifies this combined functional image. In other words, the one-variable function forms are too complicated for the system to handle. We classify this type of three-variable function forms as the ICL-Class function form.

The new system took the advantage of alternatively performing different differential transformations respect to the two independent variables. By doing this (the first three transformation in the solution reported in the system's output card on next page), the original three-variable function form discovery task was split into three easy to handle subtasks: (1) to find a single variable function $\tan(y)$ as the descriptive expression for the first differential transformation, (2) to find a single variable function $1/x$, and (3) to find

```
Task:  Comparison-3
Termination Status:  Success
Input Image:  101 × 101,
        Xmin=0.5, Xmax=1.5, Ymin=0.5, Ymax=1.5
Original Precision:  1.0E-12
Reference Image(s):
        1.  File=./Comparison-3.B1.dat, Type=Boundary
        2.  File=./Comparison-3.B18.dat, Type=Boundary
Nodes:  Total 258, Explored 45, Open 106
Running Time:  12'18.57"
Memory Usage:  4.55MB
Transformations:
        1.  Dif(z,x)
        2.  Variable Exchange (x<>y)
        3.  Dif(z,x)
        4.  Reciprocal
Matching Primitive Function:
        z^2=(1/16)xy
Errors:  Fitting:8.37e-05, Matching:2.97e-05
```

Figure 5.8: The Report Card for Comparison Test 3

the two-variable function $z = 1/\sqrt{xy}$. The first two are easy to handle with the two-variable FFD, and the last can be solved with the discovered transformation T_{Rec} and the primitive fitting $z = \sqrt{xy}/16$. Therefore the original function form discovery problem is solved. Figure 5.8 is the discovery result of FFD-II. The solution is identical to the test function — Equation 5.13.

5.5 Randomly Generated Surfaces

The motivation of function form discovery research is to create a system that can find function forms that represent the given numeric data satisfying the justification, parsimony and transparency criteria. We have seen that when the data is generated explicitly by a compact function form, such as the cases given in Section 5.2 and 5.3, the proposed methodology has a good potential to find the exact form. However, the question remains whether or not the system will perform similarly well when the underlying function forms are not expressible in terms of elementary functions. In this section, we will further examine the system's capability in handling such discovery problems. The objective is to observe whether the system is able to extract information from the observation more effectively than traditional numeric tools.

A set of experiments on discovering function forms from randomly generated smooth surfaces were conducted. Each surface was a ninth order two-variable polynomial over the domain $(x, y) \in [0,\ 1;\ 0,\ 1]$. Their coefficients were randomly generated quantities between -1 and 1. They can be viewed as truncated Taylor series expansions of certain unknown C^{∞} functions.

To carry out the experiments of this part, the fitting thresholds were relaxed to let the system terminate with a relatively rough match. $\epsilon_{\mathcal{P}}$ and $\epsilon_{\mathcal{M}}$ were increased to 100 times of their normal settings, and if the system could not find a match, the thresholds were relaxed by another 100 times of the previous one. All other settings remained the same as described on page 157.

Let us first examine an example. Rounded to four significant figures, an example of random surface generated by a program is:

$$\begin{aligned} z \;=\; & .825 - .682x + .603x^2 + .366x^3 - .487x^4 + .061x^5 - .376x^6 + .021x^7 \\ & -.584x^8 + .270x^9 + .147y - .565xy + .975x^2y + .0148x^3y + .577x^4y \\ & -.951x^5y + .655x^6y - .987x^7y - .495x^8y + .107y^2 + .254xy^2 \\ & +.736x^2y^2 - .604x^3y^2 + .829x^4y^2 - .720x^5y^2 + .219x^6y^2 - .071x^7y^2 \end{aligned}$$

$$+.224y^3 + .065xy^3 + .795x^2y^3 + .443x^3y^3 + .793x^4y^3 - .603x^5y^3$$
$$-.794x^6y^3 + .586y^4 + .715xy^4 - .405x^2y^4 - .729x^3y^4 - .750x^4y^4$$
$$+.780x^5y^4 - .351y^5 + .512xy^5 - .957x^2y^5 - .352x^3y^5 + .195x^4y^5$$
$$-.064y^6 - .068xy^6 - .723x^2y^6 + .492x^3y^6 + .077y^7 - .577xy^7$$
$$+.869x^2y^7 - .305y^8 + .711xy^8 - .116y^9.$$

The function form description discovered by FFD-II in the observation domain $[0, 1;\ 0, 1]$ is:

$$\left\{ \begin{array}{l} T_{\text{Log}} \circ T_{\text{FAC}(0.9822-x)}, \\ \mathcal{F} : z = 2.583x^2 + 1.432xy + 0.7006y^2 - 0.2230x - 0.2847y - 0.0190 \end{array} \right\}.$$

The matching error is 6.893e-02. Transformed to an explicit expression, the function is:

$$z = (0.9822 - x)e^{2.583x^2+1.432xy+0.7006y^2-0.2230x-0.2847y-0.0190}.$$

Since there are seven parameters in the discovered function, we can select the least-squares surface fitting to fit the same random surface to the seven-parameter polynomial[15]:

$$z \;=\; +0.9959 - 2.7292x - 0.5503y + 7.4074x^2 + 0.8975xy + 1.1076y^2 - 5.9783x^3.$$

The root-mean-square error of the fitting is 6.94e-02. The comparisons are listed as the following.

1. The root-mean-square errors are 6.19e-02 Vs. 6.94e-02. FFD-II achieved similar accuracy.

2. The polynomial fitting result cannot be easily interpreted. The discovered description, on the other hand, shows that the function value is around zero when x=0.9822 and the surface stays mainly above the plane of $z = 0$ (positive function) within the observation domain since exponential function is positive and the linear factor is mainly positive.

[15] The complete third order two-variable polynomials contain ten coefficients. To fit to a seven coefficients polynomial, six parameters related to the complete second order polynomial are selected, and only one third order coefficient is non-zero parameter. The fitting with the least mean-square error is picked as the seven parameter polynomial fitting result. This scheme is also used in the followed comparison fittings.

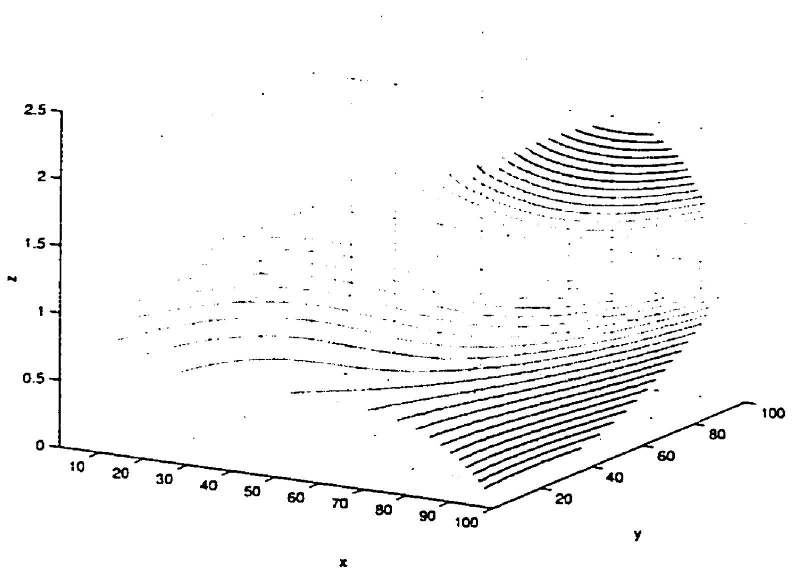


Figure 5.9: The Contour Image of Random Surface 1

3. Figure 5.9, 5.10 and 5.11 show the three dimensional contour image of the original random surface, the discovered function form and the polynomial fitting respectively. Comparing the three images, it is clear that the FFD-II discovered function preserves global features more precisely than the polynomial fitting. For example, the polynomial fitting result exhibits some false oscillatory features that do not appear in the original surface and there is also a *pit* in the polynomial fitting surface that does not appear in the given surface. The single peak and the main trend of the random surface are captured by both approximations.

4. Figure 5.9 shows the derivatives of the original polynomial, the polynomial fitting function and the discovered function. Clearly, the discovered function preserves the shape of the original random surface significantly better than the polynomial fitting result does. The latter representation looses most of the information concerning the derivatives.

Ten other random surfaces were tested. The system successfully found solutions to seven surfaces and failed in three cases even with the further relaxed thresholds. Table 5.8

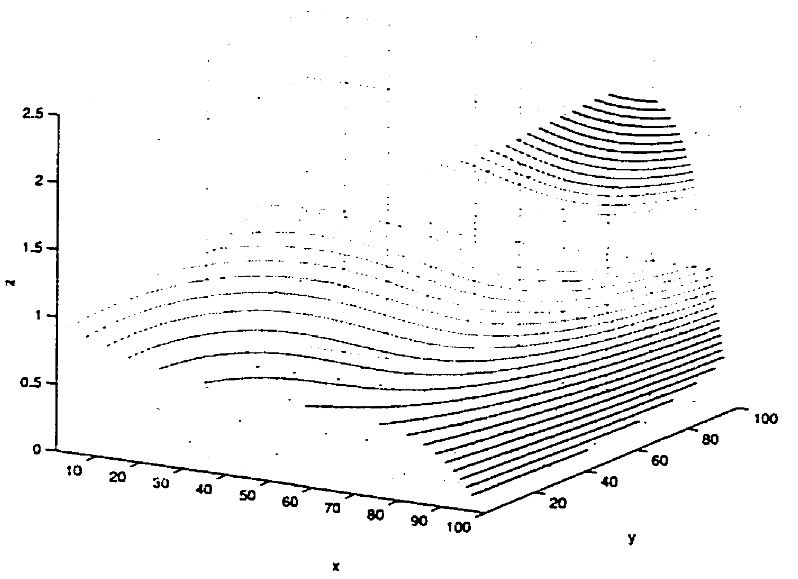


Figure 5.10: The Contour Image of the Discovered Form of Random Surface 1

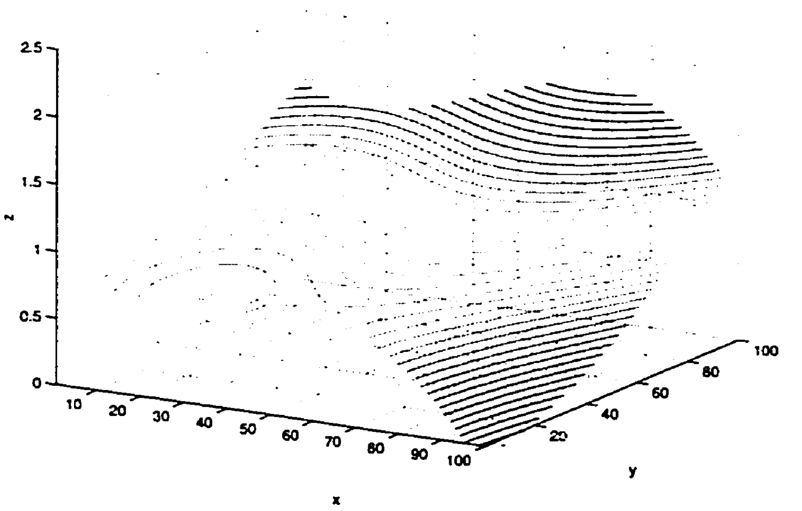


Figure 5.11: The Contour Image of the Polynomial Fitting of Random Surface 1

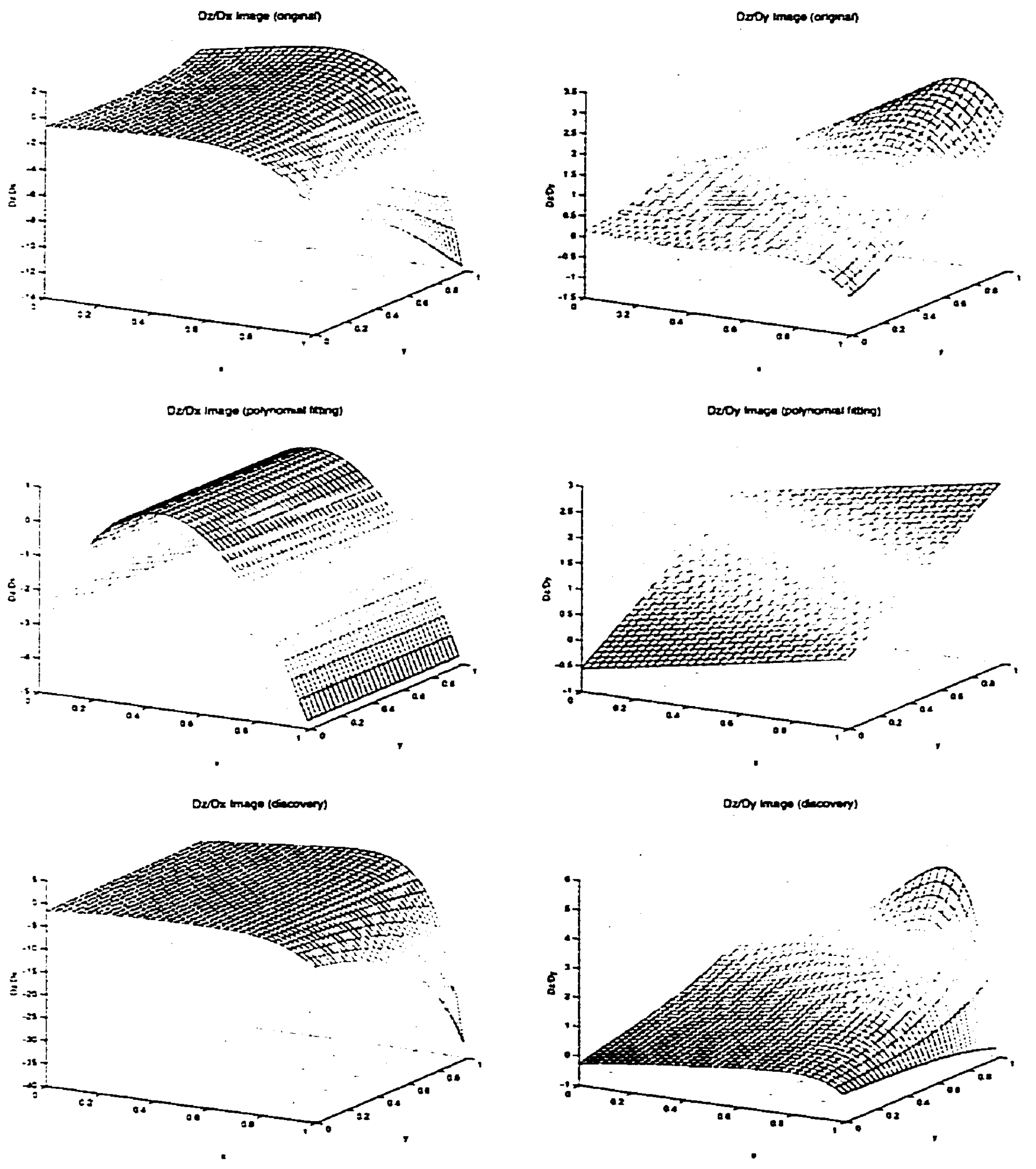


Figure 5.12: Mesh-grid Images of Derivatives of Random Surface 1

lists the results of the experimental cases in which solutions were discovered by the system. In the table, "Matching Primitive" has two formats — a function expression which is a matching primitive function and a two-dimensional linear expression which is a matching primitive pattern. The column "No. of Parameter" is the number of control parameters in the discovered function form including the parameters in the descriptive expressions (if applicable), "$E_{\text{discovery}}$" stands for the computed root-mean-square error of the discovered function, "E_{fitting}" refers to the root-mean-square distances of polynomial surface fitting with the same number of coefficients. For brevity, two significant figures after the decimals are kept. Because the underlying functions are not in short forms of the elementary functions, the descriptive expressions are obtained by fitting the corresponding recorded sample data to one-variable polynomials to a satisfactory precision (close to the primitive fitting error and the verified function form matching error). The fitting results are listed in Table 5.9.

Among the seven discovered functions, five of them are simple. In Task #1, the system found a second order two-dimensional polynomial to express variable x as a function of $1/z$ and y. In this way, the fitting accuracy was improved by about 4 times compared with direct second order polynomial surface fitting. Task #2 and #3 are two other examples of changing to different polynomials in order to improve the fitting accuracy. The accuracies of discovered functions of tasks #4 through #7 are worse than that of the corresponding polynomial surface fitting results. However, the emphases of function form discovery include not only the accuracy but also the meaningfulness. In Task #6, the system discovered that the observation image is approximately a cylindric surface. In Task #7, the system discovered that the underlying function is roughly of the form $z = \left\{\int f\left(u\sqrt{2/5} - v\sqrt{3/5}\right) \mathrm{d}x\right\} + g(y)$, which could also be interpreted as: $\partial z/\partial x$ is approximately a cylindric surface.

Let us examine the results of tasks #4 and #5 in greater detail. The discovered function of Task #4 is

$$z = \int_{0.5}^{x} \left\{\int_{0.5}^{y} \mathrm{e}^{P(x,y)} \mathrm{d}y + P_1(x)\right\} \mathrm{d}x + P_2(y),$$

where,

$$P(x,y) = 3.45x^2 - 2.37xy + 1.72y^2 - 0.26x + 1.08y - 0.46,$$

Task ID.	**Transformation Sequence**	**Matching Primitive**
1	$T_{\text{INV}} \circ T_{\text{REC}}$	$\mathcal{F}: w = 0.41u^2 - 0.19uv + 0.34v^2 + 1.67u - 1.13v + 1.87$
2	T_{INV}	$\mathcal{F}: w = -0.08u^2 + 1.11uv + 1.87v^2 - 1.20u - 1.42v + 0.56$
3	$T_{\text{INV}} \circ T_{\text{VEX}}$	$\mathcal{F}: w = -0.18u^2 + 0.23uv + 0.78v^2 - 1.23u - 0.32v - 0.81$
4	$T_{\text{LOG}} \circ T_{\text{DIF}} \circ T_{\text{VEX}} \circ T_{\text{DIF}}$	$\mathcal{F}: w = 3.45u^2 - 2.37uv + 1.72v^2 - 0.26u + 1.08v - 0.46$
5	$T_{\text{REC}} \circ T_{\text{DIF}} \circ T_{\text{VEX}} \circ T_{\text{DIF}}$	$\mathcal{F}: w = 0.53u^2 - 0.09uv + 0.35v^2 - 0.92u - 0.82v + 0.96$
6	*None*	$\mathcal{P}: u\sqrt{39}/7 + v\sqrt{10}/7$
7	T_{DIF}	$\mathcal{P}: u\sqrt{2/5} - v\sqrt{3/5}$

Task ID.	**Number of Parameter**	$E_{\text{discovery}}$	E_{fitting}
1	6	**6.96e-2**	1.57e-1
2	6	**4.59e-2**	2.38e-1
3	6	**1.02e-1**	1.39e-1
4	15	5.31e-2	**2.06e-2**
5	15	5.45e-2	**2.59e-2**
6	6	3.50e-1	**1.14e-1**
7	10	1.48e-1	**5.97e-2**

Table 5.8: Results of Experiments on Random Surface

No.	Descriptive Expressions
4	$[u = 0.5,\ w = 0.500 + 1.482v - 4.746v^2 + 4.426v^3]$ $[u = 0.5,\ w = 0.260 - 1.911v + 12.505v^2 - 29.062v^3 + 30.872v^4]$
5	$[u = 0.5,\ w = 1.144 - 0.244v + 3.561v^2 - 3.120v^3]$ $[u = 0.5,\ w = 1.119 + 3.404v - 13.319v^2 + 31.688v^3 - 24.003v^4]$
6	$P(t) = 0.302 - 0.456t + 5.254t^2 - 11.682t^3 + 4.829t^4$
7	$[u = 0.5, w = -0.585 + 1.646v - 6.042v^2 + 14.573v^3 - 7.90v^4]$ $P(t) = 0.468 - 0.116t + 0.102t^2 + 0.767t^3$

Table 5.9: Descriptive Expressions for the Experiments on Random Surface

is the matching primitive function, and

$$P_1(x) = 0.260 - 1.911x + 12.505x^2 - 29.062x^3 + 30.872x^4,$$
$$P_2(y) = 0.500 + 1.482y - 4.746y^2 + 4.426y^3,$$

are the corresponding descriptive expressions of boundary conditions. The discovered function of Task #5 is

$$z = \int_{0.5}^{x} \left\{ \int_{0.5}^{y} \frac{1}{P(x,y)} \mathrm{d}y + P_1(x) \right\} \mathrm{d}x + P_2(y),$$

where,

$$P(x,y) = 0.53x^2 - 0.09xy + 0.35y^2 - 0.92x - 0.82y + 0.96,$$

is the matching primitive function, and

$$P_1(x) = 1.119 + 3.404x - 13.319x^2 + 31.688x^3 - 24.003x^4,$$
$$P_2(y) = 1.144 - 0.244y + 3.561y^2 - 3.120y^3,$$

are the corresponding descriptive expressions of boundary conditions. In these two cases, the given observation data set was formulated by first fitting the logarithm or reciprocal of the derivative image $\partial^2 z/\partial x \partial y$ to a second order polynomial and then constructing the functional representation by an integral.

The root-mean-square error of describing the observation data in this way is about double of the root-mean-square error of describing the data by directly fitting the observation data set to the fourth order polynomial which has same number of fitting parameters as tabulated in Table 5.8. However, the discovered function forms capture shape information, i.e. slope and curvature, of the observation images better than polynomial surface fitting. This can be observed by comparing the shapes and patterns of three-dimension contour images (Figure 5.13 through Figure 5.18).

Figure 5.13 shows the plots of the 3D contours of the functional images of Task #4, where (a) is the original random surface, (b) is the polynomial fitting surface and (c) is the surface of the discovered function. The pattern[16] of figure (c) is closer to (a) than figure (b), which implies that the discovered function captures the gradient better than polynomial surface fitting. A similar conclusion can be drawn from Figures 5.14 and 5.15, which plot the first order partial derivatives of the original random surface, the polynomial fitting function and the discovered function. Note that the pattern of partial derivative 3D contour images contains informations concerning the second order curvatures.

Figure 5.16 shows the 3D contours of the functional images of Task #5, where (a) is the original random surface, (b) is the polynomial fitting surface and (c) is the surface of the discovered function. Figure (c) represents the shape and pattern of (a) better than (b). Figure 5.17 and 5.18 are comparisons of 3D contour images of the two representations. Clearly, the discovered function represent the original underlying function significantly better than the polynomial surface fitting result in most part of the observation domain.

Several important conclusions can be drawn from the experiments conducted in this part.

1. As a mathematic formulation tool, FFD-II is able to translate general observation data into a compact and meaningful functional description in many situations. Each discovered form can be interpreted according to the obtained transformation sequence and the primitive.

[16] Note that at a given planar point (x_0, y_0), the direction of the gradient of a scalar field $z = f(x,y)$ is perpendicular to the contour curve $f(x,y) = C$ that crosses (x_0, y_0). Therefore, when the patterns of two sets of contour curves are close to each other, the gradient vectors of the two corresponding fields will be relatively close to each other.

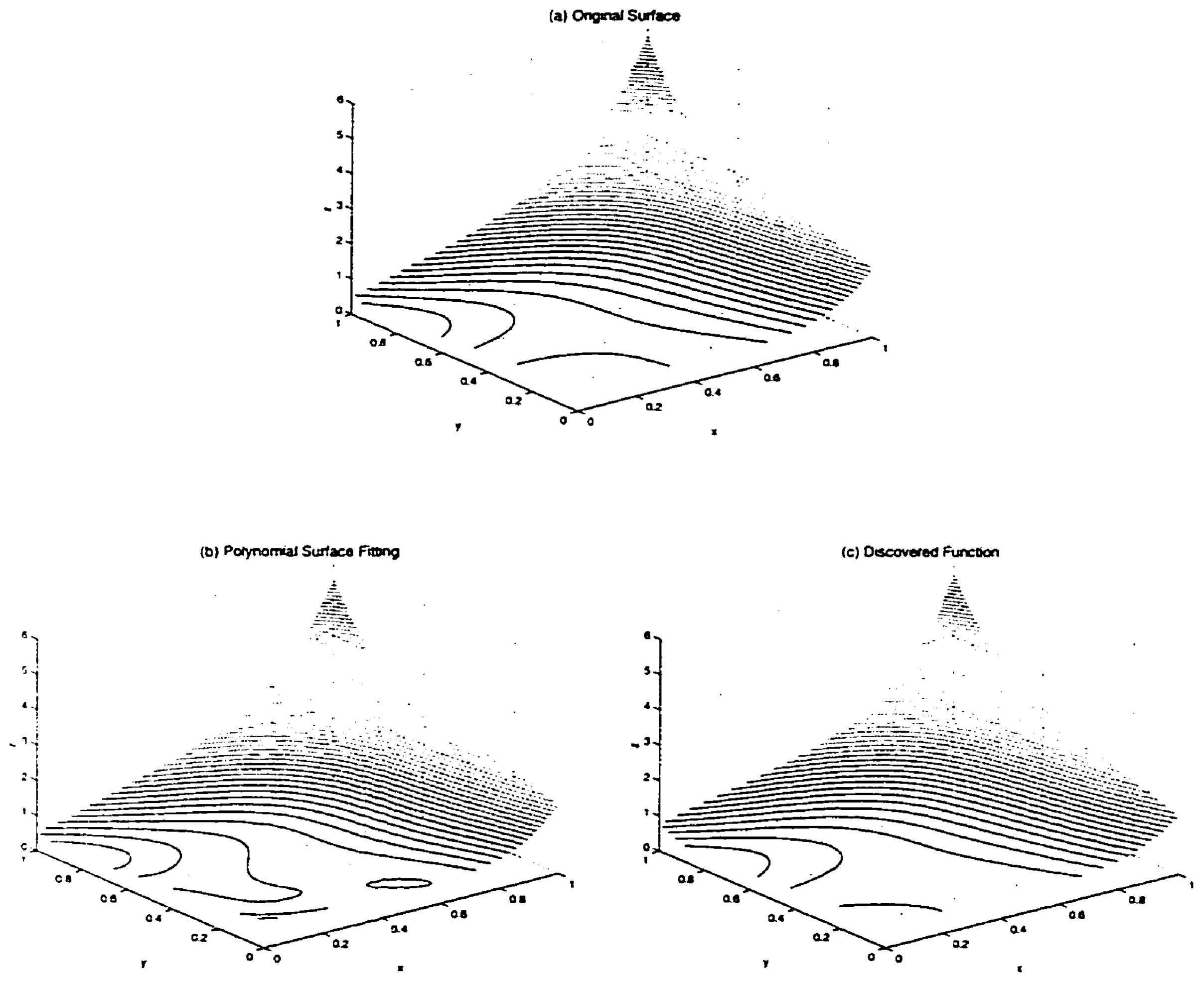


Figure 5.13: 3D Contour Images of the Surfaces in Task #4

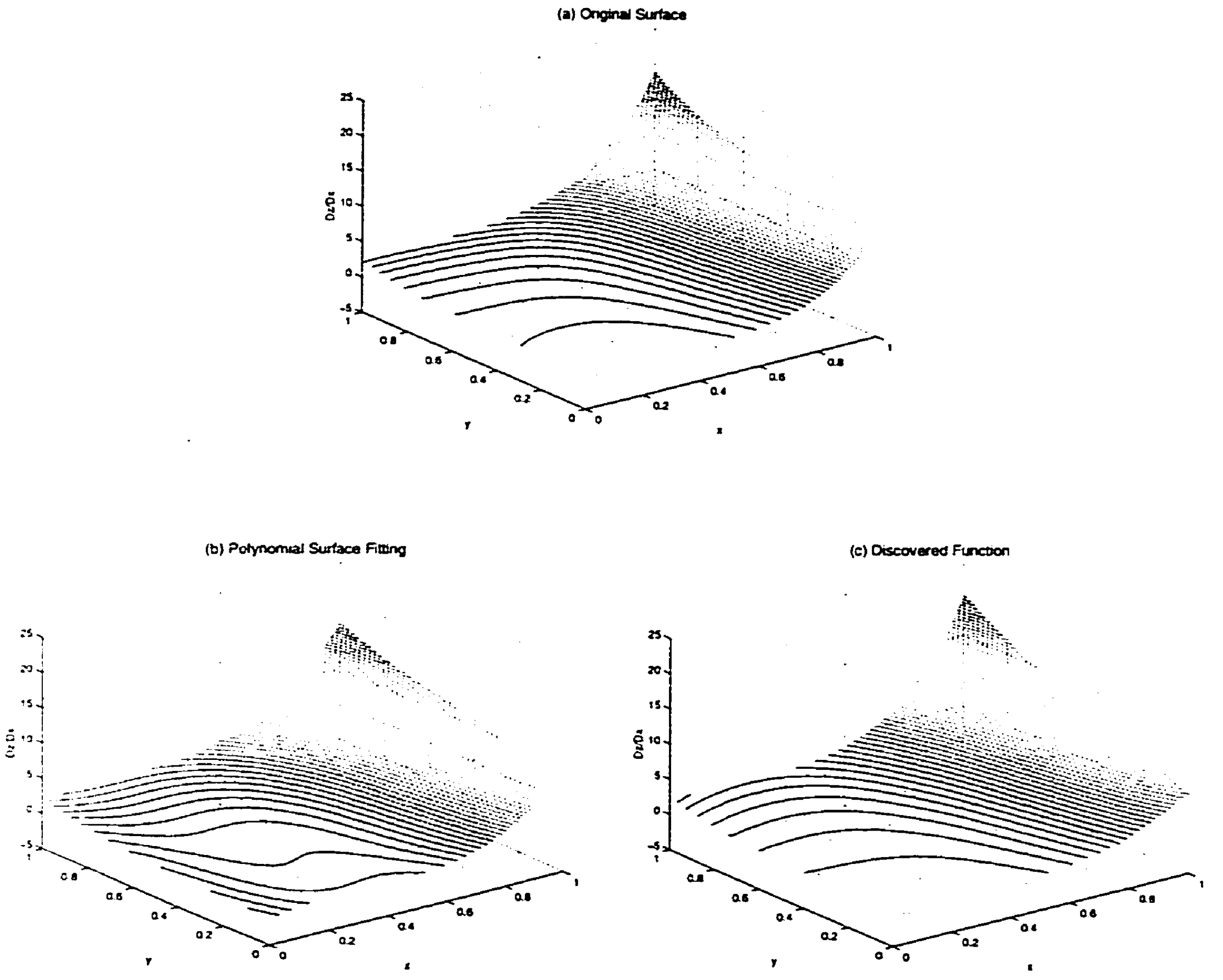


Figure 5.14: 3D Contour Images($\partial/\partial x$) of the Surfaces in Task #4

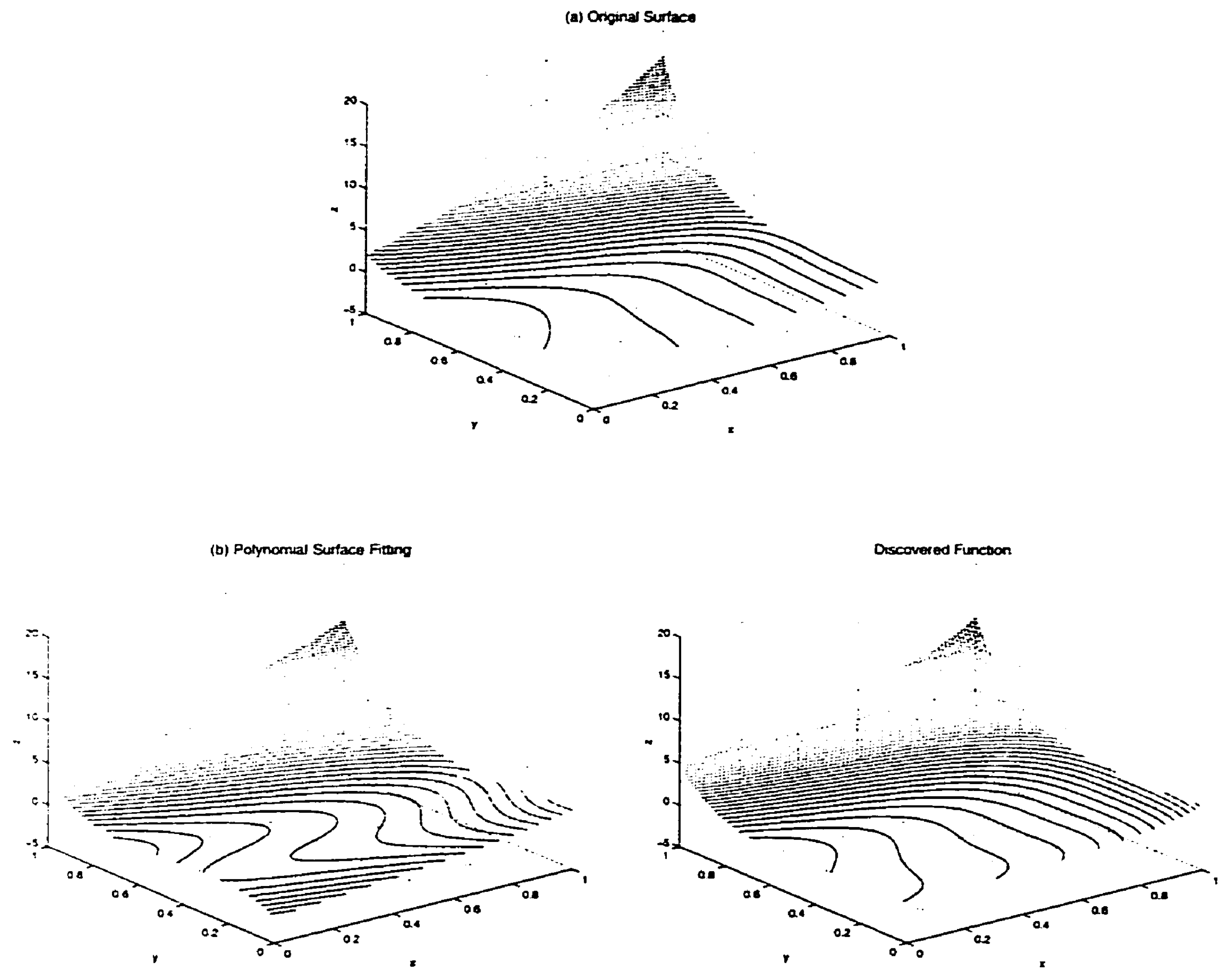


Figure 5.15: 3D Contour Images($\partial/\partial y$) of the Surfaces in Task #4

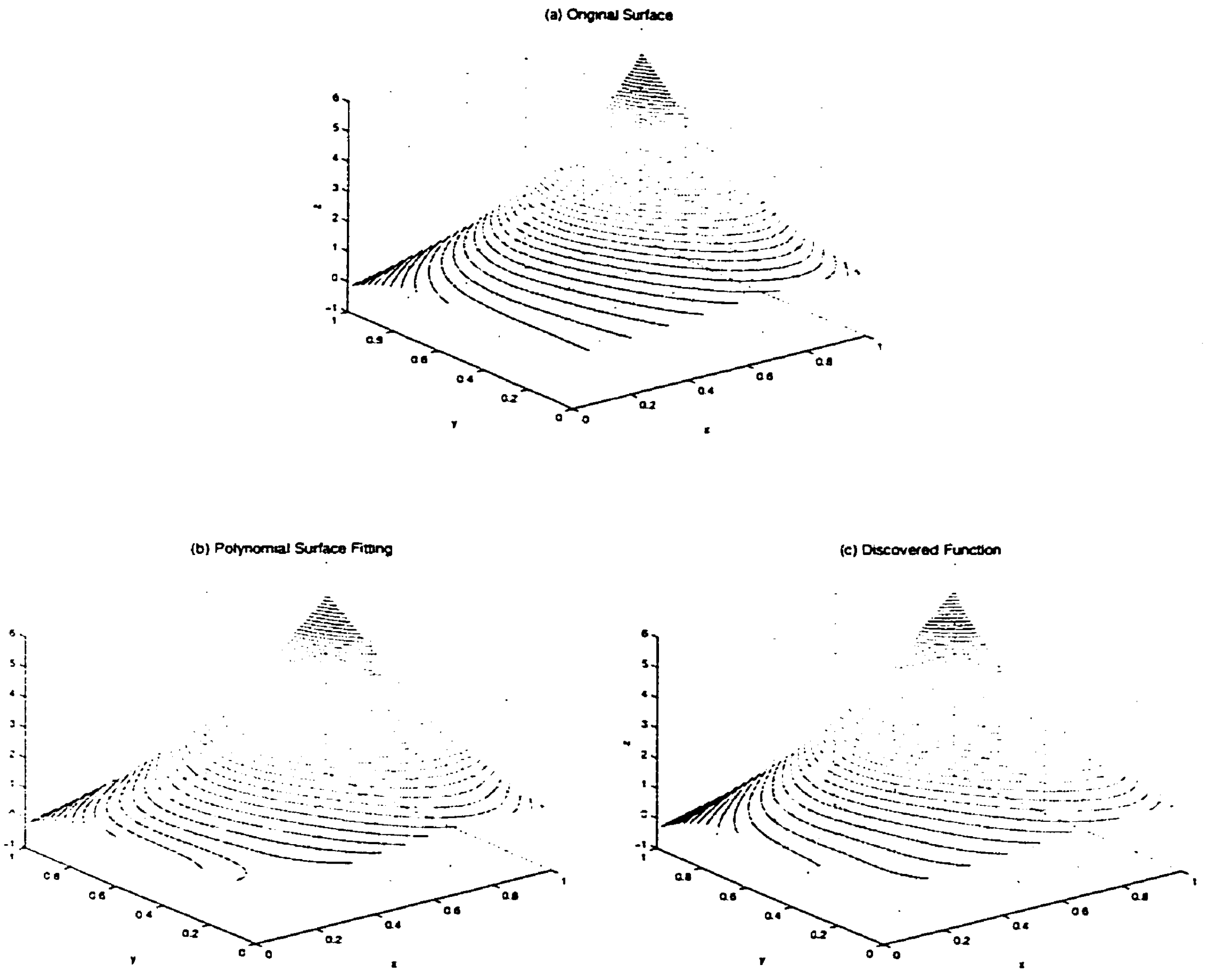


Figure 5.16: 3D Contour Images of the Surfaces in Task #5

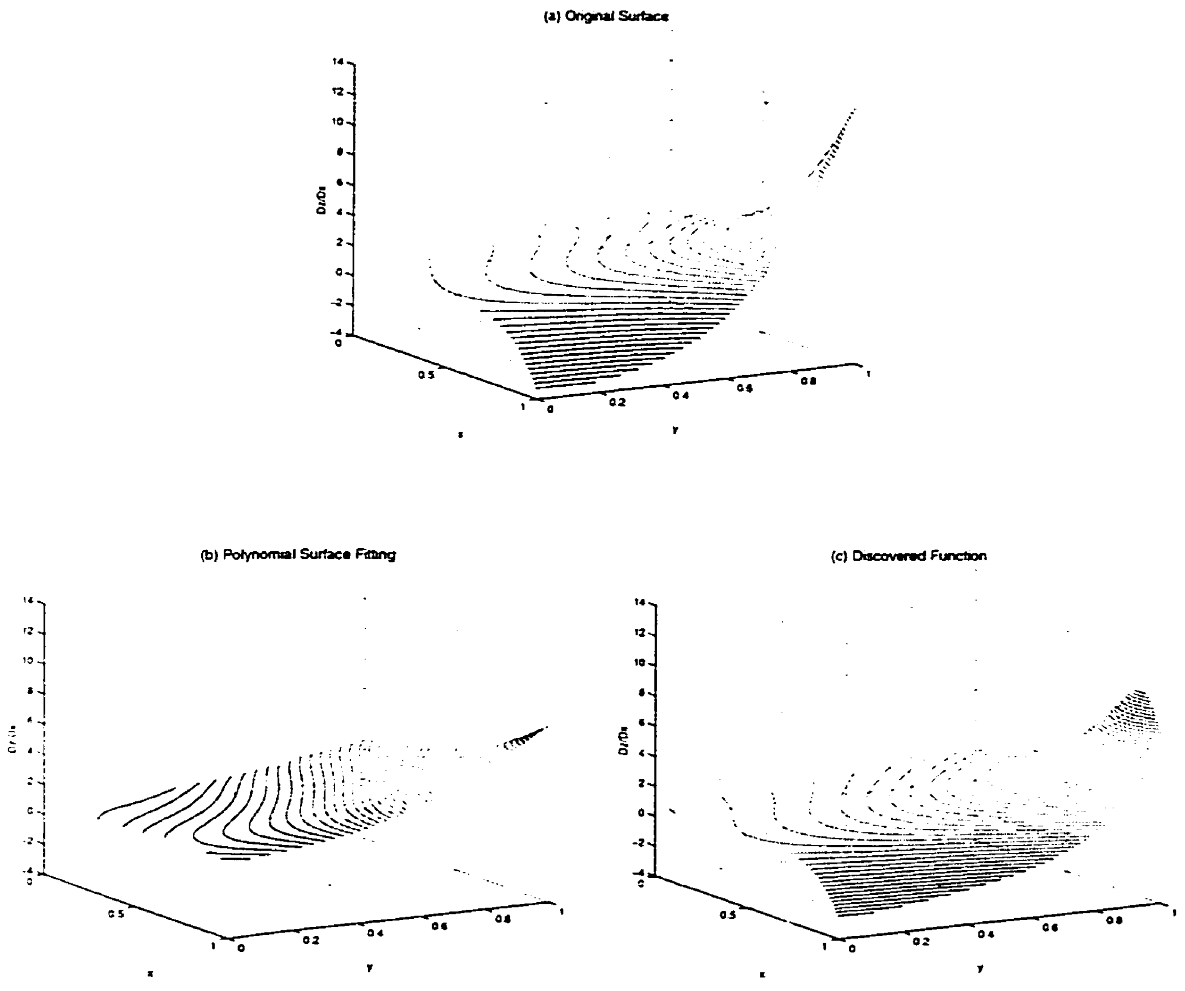


Figure 5.17: 3D Contour Images($\partial/\partial x$) of the Surfaces in Task #5

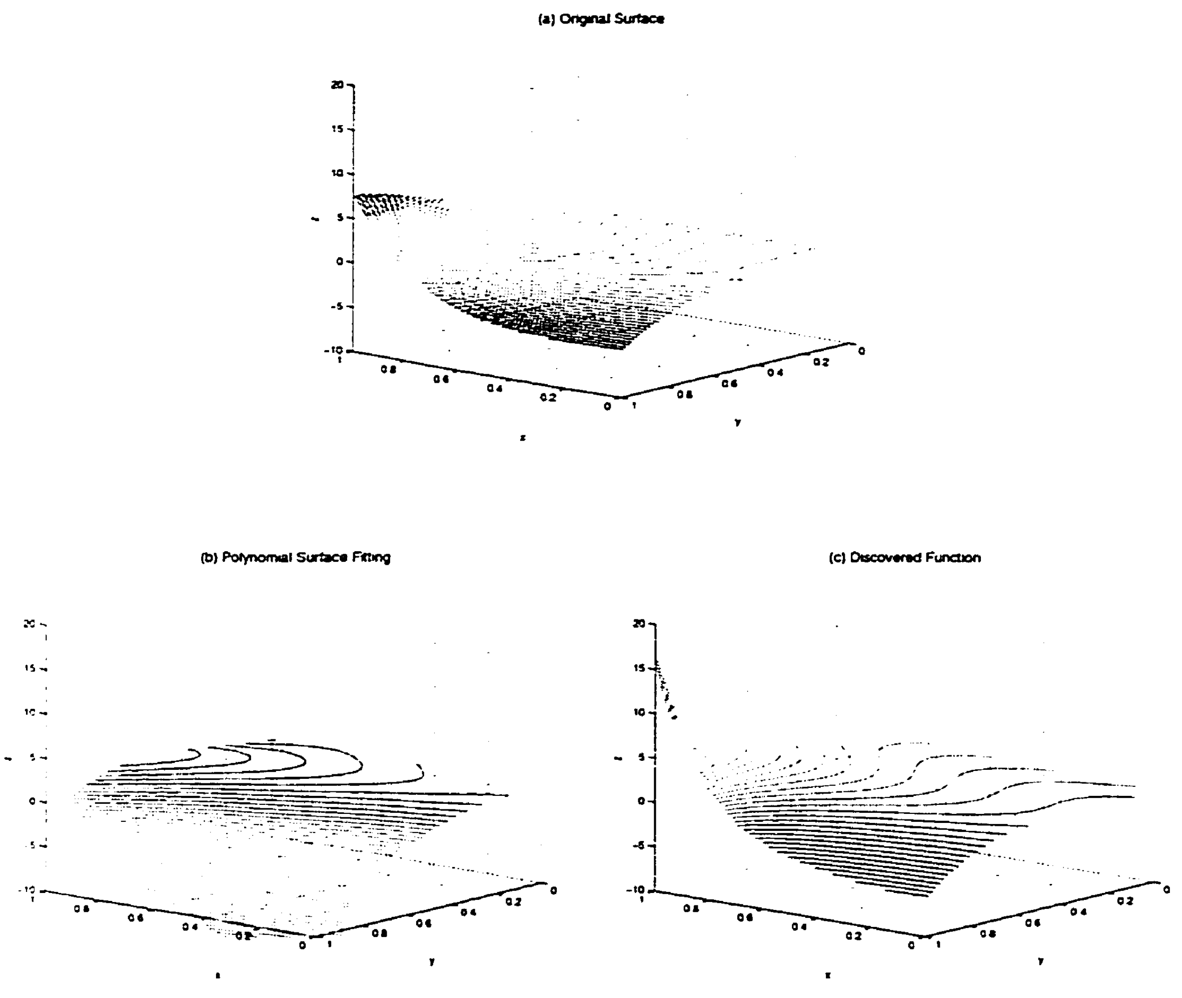


Figure 5.18: 3D Contour Images($\partial/\partial y$) of the Surfaces in Task #5

2. Compared with traditional polynomial surface fitting method, important geometric features are conserved more precisely by the abstracted formula. Such features include surface features represented in the example random surface, gradients and curvatures.

3. FFD-II has the ability to formulate the given numeric observation data using a variety of different functions, and it can achieve better or similar accuracy compared with polynomial surface fitting method.

4. Traditional data modeling tools, such as polynomial fitting, have good reputation in representing given data accurately. When it is necessary we can usually increase the accuracy by simply increasing the number of fitting coefficients. However, FFD-II places more emphasis on the parsimony and transparency. This property has been further demonstrated.

5.6 Experiments on Noisy Data

We have just seen the enhanced ability of the proposed direct model over an indirect model. We also know the challenge associated with solving multi-variable function form discovery problems using direct approach[17]. This section will contribute to the experiments that demonstrate how the new system handles noise using proposed methodology, adaptive error control. I will first describe the noise model and the experiment design. Then the error treatment scheme will be examined from different aspects with experiments on selected function forms.

5.6.1 Noise Model and Experimental Design

To test the noisy input effects and the performance of the proposed noise treatment recipes, a pseudo random number generator is used to generate uniformly distributed random numbers. Let $z = f(x, y)$ be a function form with whom a simulated observation data set will

[17] Refer to the discussions in Section 4.2.2.

be made. The noisy input is the simulated observation data with additive noises:

$$\tilde{z}(x, y) = f(x, y) + \epsilon \qquad (5.14)$$

where ϵ is a uniformly distributed random variable over the interval $[-a,\ a]$ and a is referred to as the *Noise Level* of the simulated observation data set.

It has been pointed out that error propagation is a major challenge for conducting data transformation based direct function form discovery. Computational errors are the errors introduced by digital computations using selected numeric tools with a digital computer. Noises, on the other hand, refer to the inaccuracy of the collected observation data. The essential difference between the effects of computational errors and added noises to the function form discovery system is that the functional image with added noise is usually more uneven than a functional image with computational errors. When certain numeric data transformations are conducted upon an uneven image, large scale of propagated error could be introduced. Examining function $y = x^2$ as an example, let:

$$\begin{cases} x_i &= 0.01i, \quad (i = 0, \cdots, 102), \\ y_i &= x_i^2, \quad (i = 0, \cdots, 102), \\ \tilde{y}'_i &= 2x_i + \epsilon_i, \quad (i = 0, \cdots, 102) \end{cases}$$

where ϵ_i are uniformly distributed random numbers over the interval $[-0.01, 0.01]$ [18], and

$$\begin{cases} \overline{y}'_i &= (y_{i+1} - y_i)/0.01, \quad (i = 0, \cdots, 101) \\ \overline{y}''_i &= (\overline{y}'_{i+1} - \overline{y}'_i)/0.01, \quad (i = 0, \cdots, 100), \\ \tilde{y}''_i &= (\tilde{y}'_{i+1} - \tilde{y}'_i)/0.01, \quad (i = 0, \cdots, 100). \end{cases}$$

Clearly, $\overline{y}'_i$ are the values of numerically computed first order derivative of the function $f = x^2$ (function $f = 2x$), $\overline{y}''_i$ are the numerically computed second order derivative values of function $f = x^2$ (constant 2) based on the values of $\overline{y}'_i$, $\tilde{y}'_i$ are the function values of $f = 2x$ with added noises, and $\tilde{y}''_i$ are the numerically computed first order derivative value of the function $f = 2x$ (constant 2) with added noises. Using digital computer, we

[18] 0.01 is the maximum approximation error of using $\overline{y}'_i$ to approximate the first order derivative of the function $y = x^2$.

can find that the averaged approximation error of $\tilde{y}_i''$ to constant 2 is 0.7538, while the averaged approximation error of $\overline{y}_i''$ to constant 2 is only 3.35e-13.

The simple example demonstrates how the added noises might affect the numeric computation results more significantly than usual computational errors. In FFD-II, differential transformation is numerically implemented. Although the computing scheme is different to the simple first order difference scheme used in the example, similar effect can be observed. Therefore, experiments on noisy observation data set is more useful for justifying the system's ability to handle error propagations.

The following are the general background of the experiment design for the noisy input.

1. The high computing time and memory space intensities of the algorithm decide that we can only choose relatively simple test function forms, which has a solution in a small depth in the search tree, to generate the simulated observation data set.

2. Among the transformations in the system's transformation set, T_{DIF} is the one that is most sensitive to noise. Therefore, the selected test function form must contain at least one differential transformation in the accurate solution.

3. All simulated observation data set were made in the corresponding observation domains that were partitioned by a 511×511 uniformly distributed rectangular mesh grid. Thus each input observation data set contains 511 × 511 double precise real valued observation coordinate triples (x, y, z).

4. To this stage, the system only works with two resolution levels — *Fine* and *Coarse* step functional images[19]. A *Fine Step* functional image refers to the original input functional image or the image transformed from it. A fine step image can be expressed as

$$O_F = \{(u_{ij}, v_{ij}, w_{ij}) \mid i = 1, 2, \cdots, 511; j = 1, 2, \cdots, 511\}.$$

[19] However, the proposed error treatment methodology is able to work with multi-resolution scheme. Considering the available computer resources, only two resolution levels are used to demonstrate and test the proposed methodology.

A *Coarse Step* functional image is a 101×101 observation data set which is either a transformed functional image of a coarse image or a evenly selected subset of a fine step functional image

$$\mathcal{O}_C = \{(u_{ij}, v_{ij}, w_{ij}) \mid i = 6, 11, \cdots, 506; j = 6, 11, \cdots, 506\} \subset \mathcal{O}_F.$$

5. For the purpose of fully observing the performance of the system, a small $\epsilon_{\mathcal{M}}$ and a relatively large $\epsilon_{\mathcal{P}}$ were set. They were 10^{-7} and 10^{-1} respectively. The maximum searching depth were set to be 5. The relaxed $\epsilon_{\mathcal{P}}$ setting enables the system to propose more function form hypotheses for verifications and the tight $\epsilon_{\mathcal{M}}$ setting keeps the search goes on. Altogether, they can force the system to test more hypotheses in a single discovery task.

6. To analyze the results, the discovery system was slightly modified so that the full discovery processes with all necessary information, such as the fitting and matching error of all the abstracted hypotheses, could be recorded.

7. To each simulated observation data set, two rounds discovery were run, one with error treatment switch turned "ON" and the other with it turned "OFF". Since the error treatment can only be conducted with multi-resolution observation data set, it could be disabled by specifying that there is only one available resolution level. The performance of the proposed methodology can be evaluated based on the comparison of the corresponding results.

Generally speaking, the discovered function form may not be in the exact form of the given underlying function that was used to generate the simulated observation data set. The system discovers the functional representation of the given data within a tolerable error level. However, the selected function forms in this section are all expressible by the function form description language $\tilde{\mathcal{L}}$. Thus the purpose of the experiments is to observe how well the proposed error treatment scheme will reduce the effects of noises and propagated errors and help to extract the exact underlying function forms.

5.6.2 Multi-Solution

The first function form to be tested is

$$z = e^{x-y} + xy. \qquad (I)$$

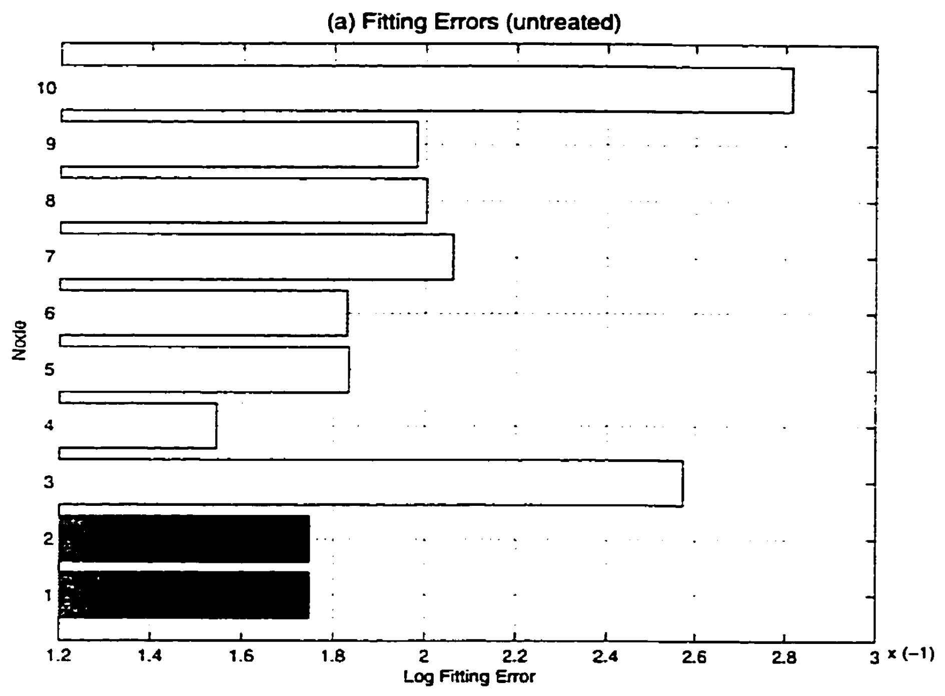
There are four solutions to the problem. They are listed in Table 5.10.

Solution	Transformation	Matching Primitive	Treated	Plain
1	$T_{\text{DIF}} \circ T_{\text{VEX}} \circ T_{\text{DIF}}$	$\mathcal{P}$: $x - y$	$\surd$	$\surd$
2	$T_{\text{DIF}} \circ T_{\text{VEX}} \circ T_{\text{DIF}} \circ T_{\text{VEX}}$	$\mathcal{P}$: $x - y$	$\surd$	$\surd$
3	$T_{\text{DIF}} \circ T_{\text{DIF}}$	$\mathcal{P}$: $x - y$	$\surd$	$\times$
4	$T_{\text{DIF}} \circ T_{\text{DIF}} \circ T_{\text{VEX}}$	$\mathcal{P}$: $x - y$	$\surd$	$\times$

Table 5.10: Four Solutions of Form I

The observation domain was chosen to be $(x, y) \in [-0.5, 0.5; -0.5, 0.5]$ and the noise level is 10^{-2}. The last two columns in Table 5.10 show whether or not a correct solution was extracted in the experiment. The column titled by 'Treated' means that the error treatment was used in the test, and 'Plain' means that error treatment was not used. "$\surd$" denotes that the corresponding exact function form was correctly abstracted and "$\times$" denotes that the corresponding exact function form was not correctly abstracted. All four accurate solutions were abstracted when proposed error treatment was used. However, there are two out of the four accurate solutions that were not abstracted without error treatment.

Let us summarize ten best matches recorded during the system's discovery process, including four accurate solution matches and six other verified matching hypotheses. Figure 5.19 shows the primitive fitting errors at each of the nodes where function form hypotheses were abstracted. In the figure, (a) shows the results of conducting the discovery without applying error treatment and (b) shows the results with error treatment applied. Node 1 to 4 are the four nodes which may be associated with an accurate solution (depending on whether or not the corresponding correct primitive could be found). Each of the shaded bars im-



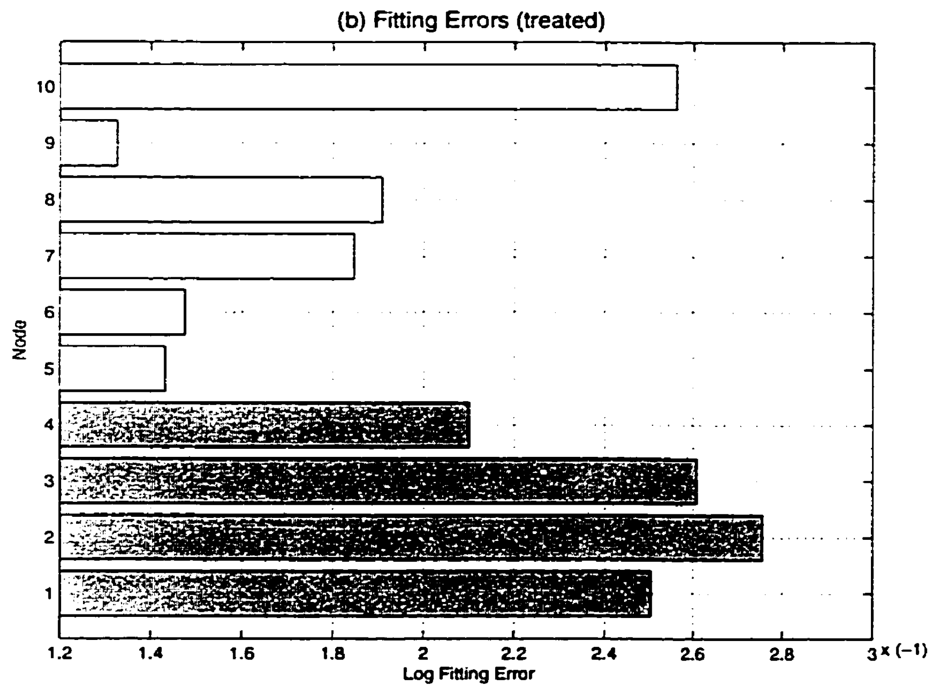


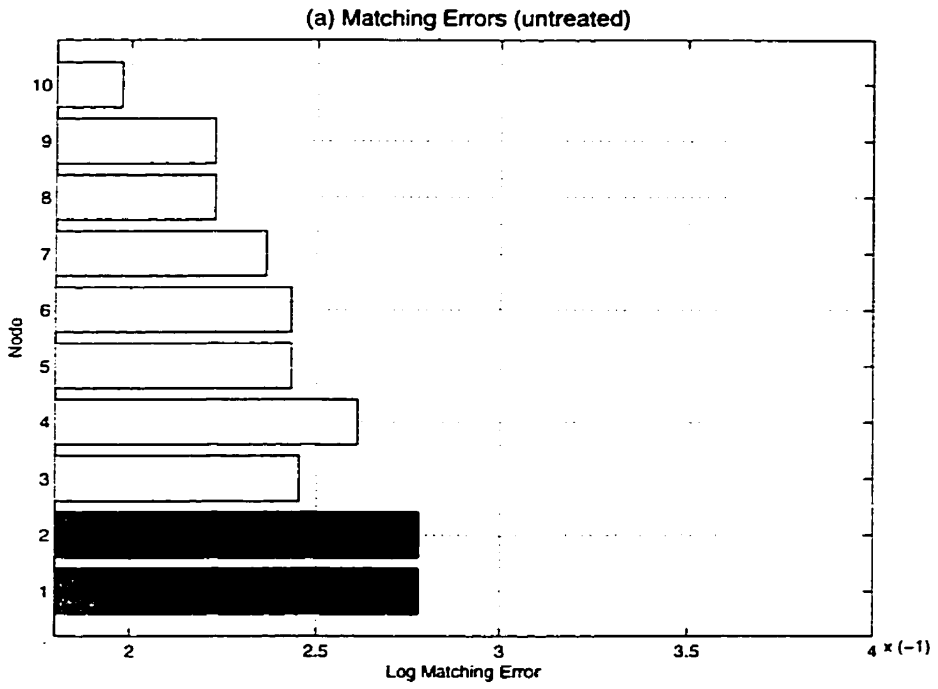
Figure 5.19: The Fitting Errors At Corresponding Hypothesis Nodes of the Form I

plies that the associated abstracted hypothesis is an accurate functional form description of Function (I), each of the unshaded bars implies that the associated abstracted hypothesis is not a description of Function (I). The axis "Log Error" is the values of $\log_{10}(\epsilon_p)$ where ϵ_p is the computed fitting error at each corresponding node. From the figure we can clearly observe that the proposed error treatment scheme significantly improves the primitive fitting test[20]:

- Without treating the error, the system fails to abstract two of the four accurate descriptions due to the noises associated with the input observation data. However, by treating the error with the proposed scheme, the system successfully abstracted four hypothesis corresponding to the four descriptions of Function (I).
- Without treating the error, Figure 5.19(a), the fitting errors of the two hypotheses associated with node 1 and 2 respectively, which are accurate descriptions of Function (I), are not the smallest. There are totally seven other hypotheses whose primitive fitting errors are smaller than the fitting errors of node 1 and 2.
- By treating the error using the proposed scheme, Figure 5.19(b), the fitting errors at nodes 2 and 3 are the smallest among the ten recorded hypotheses, and the error at nodes 1 and 4 are only greater than the error of one of the six other recorded hypotheses, (node 10), which are not description of Function (I).

Figure 5.20 depicts the matching errors in similar layout as Figure 5.19. In Figure 5.20(a), the two hypotheses corresponding to the accurate descriptions of the underlying function have the smallest verified matching errors, Figure 5.20(b) shows that the four hypotheses corresponding to the accurate descriptions of the underlying function have the smallest verified matching errors. It implies that the system has the potential to discover 1) two out of the four accurate descriptions of the underlying function form without treating the error, and 2) all four descriptions when the proposed error treatment scheme is employed.

[20] Since the system forms abstracted function form hypotheses based on test of whether or not the computed primitive fitting error is smaller than a threshold, it is desired that the nodes associated with accurate function form descriptions have the smallest primitive fitting errors.



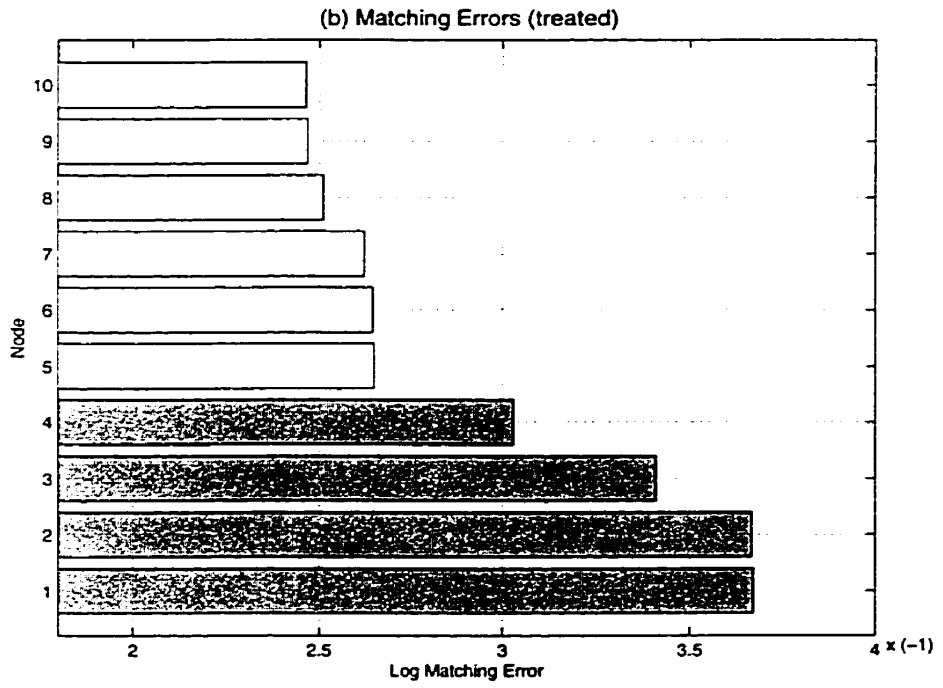


Figure 5.20: The Matching Errors At Corresponding Hypothesis Nodes of Form I

However, the proposed error treatment recipes not only allow the system to find two more accurate descriptions, but also make the accurate descriptions more distinguishable from other hypotheses. In Figure 5.20(a), the matching errors of the two hypotheses corresponding to the accurate descriptions (node 1 and 2) are not significantly smaller than the matching error at node 4 which is not associated with an accurate description of the underlying function. They are 1.66e-03 and 2.45e-03 respectively. In Figure 5.20(b), in contrast, the matching errors of the hypotheses corresponding to accurate descriptions are significantly smaller than the matching errors of the hypotheses which do not correspond to an accurate description of the underlying function. For example, the matching error at node 1 is 2.13e-04 and the matching error at node 5 is 4.19e-02. Thus we have stronger evidence to believe that the discovered functional form captured the significant underlying functional pattern of the given observation data.

Concerning the descriptive parameter fitting error, Figure 5.21 is a comparison of the two test results. By treating the error utilizing the proposed error treatment scheme, the accuracy of the fitting parameters are significantly increased.

5.6.3 Variation of Noise Level

We have seen the improvements made by the proposed error-treatment methodology through studying a multi-solution case. In this section, I will justify the methodology by observing the performances through variating the noise level on a single solution case. To evaluate the proposed methodology, a new term — Discovery Ratio — is introduced.

Definition 24 *Let ϵ_m be the matching error of the hypothesis witch is an accurate function form description of the known underlying function, ϵ_N be the matching error of a function form hypothesis that has the smallest value among all the hypotheses that are not function form description of the known underlying function. The* Discovery Ratio *is the ration of ϵ_m and ϵ_N*

$$\mathcal{D}_R = \frac{\epsilon_m}{\epsilon_N}.$$

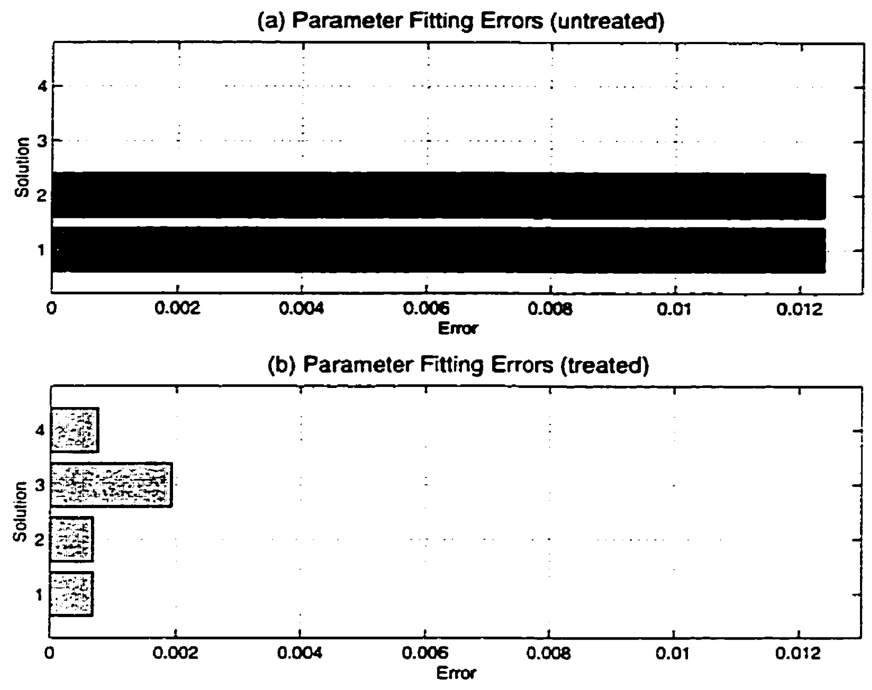


Figure 5.21: The Parameter Fitting Errors At Corresponding Solution Nodes of Form I

Since the system accepts a hypothesis based on testing whether or not the corresponding matching error is smaller than a preset threshold, a small $\mathcal{D}_R$ implies a better chance for the system to discover the accurate underlying function form. For example, if $\epsilon_m = 10^{-4}$ and $\epsilon_N = 0.1$, any preset threshold value in the range $(10^{-4}, 0.1)$ will enable the system to terminate with a successful discovery of the accurate underlying function, and in contrast, if $\epsilon_m = 0.1$ and $\epsilon_N = 10^{-4}$, it is generally impossible for the system to terminate with a successful discovery of the accurate underlying function. The discovery ratio values for these two situations are 10^{-5} and 10^5 respectively.

The second function form to be tested is

$$z = \log(x + \sqrt{x^2 + y^2}). \qquad \text{(II)}$$

This function form has one solution described by the proposed function form description language $\tilde{\mathcal{L}}$:

$$\{T_{\text{REC}} \circ T_{\text{DIF}}, \mathcal{F} : z = \sqrt{x^2 + y^2}\}.$$

Totally fourteen test runs were conducted to discover the above description with both the error-treatment switched on and off. The observation domains of the experiments carried out in this section were all $(x, y) \in [0.2, 2, 2; 0.2, 2.2]$. However, the input noise level variated from 10^{-4} to 0.5×10^{-2} (Refer to the table below).

Test	1	2	3	4	5
Noise	10^{-4}	2×10^{-4}	3×10^{-4}	4×10^{-4}	5×10^{-4}
N-Solution	$\surd$	$\surd$	$\surd$	$\surd$	$\times$
T-Solution	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Test	6	7	8	9	10
Noise	6×10^{-4}	7×10^{-4}	8×10^{-4}	9×10^{-4}	10^{-3}
N-Solution	$\times$	$\times$	$\times$	$\times$	$\times$
T-Solution	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Test	11	12	13	14	
Noise	2×10^{-3}	3×10^{-3}	4×10^{-3}	5×10^{-3}	
N-Solution	$\times$	$\times$	$\times$	$\times$	
T-Solution	$\surd$	$\surd$	$\surd$	$\times$	

In the above table, "$\surd$" denotes that the accurate function form was discovered, and "$\times$" means that the accurate function form was not discovered. The rows 'N-Solution' shows the results of the tests without employing the proposed error treatment scheme and the raws labeled by 'T-Solution' show the results of the tests with proposed error treatment scheme employed.

Without employing the error treatment, the system failed to discover the underlying function form when input noise level increased to 5×10^{-4}. With employing the error treatment, the noise tolerance increased to 4×10^{-3}, that is about ten times of the error tolerance without employing the proposed error treatment scheme. Remember, this result was obtained by using observation data with only two-resolution levels. Intuitively, a better improvement could be expected when multi-resolution image is used since the accuracy of the averaging based smoothing scheme increases with the increase of the number of usable sample points.

Figure 5.22 plots the discovery ratios. Observing the plot carefully, we can find that no matter the error treatment is employed or not, the results to the first four test samples are very close. This phenomenon is due to the adaptive manner of the treatment scheme. When the roughness of a node does not exceed the threshold, the smoothing process will not be triggered. That is the situation here. However, the parameter matching errors are still improved since the final functional images had been smoothed. Figure 5.23 compares the parameter fitting errors of the two test types.

This test proves again that the proposed error-treatment scheme significantly increases the chance for the system to discover the accurate underlying function, and improves the accuracy of the discovered results. In other words, the noise tolerance level is increased[21].

5.6.4 Experiments on More Transformations

Up to now, the transformations T_{LOG}, T_{REC} and T_{FAC} are absent from the presented noisy input experiments. To complete our experiments, the third and fourth experiments are designed to include these transformations into our examinations.

The third function form to be tested is

$$z = \frac{e^{x^2+y^2} + e^{x^2-y^2}}{2}. \tag{III}$$

The simplest description of this function using the proposed function form description language is

$$\{T_{\mathrm{DIF}} \circ T_{\mathrm{LOG}},\ \mathcal{F} : z = 2x\}.$$

[21] Note that since the numeric tools employed by the system's basic discovery process are based on numeric fitting. Thus the system has basic capability to tolerate noises to a certain level. This comment also applies to the FFD system.

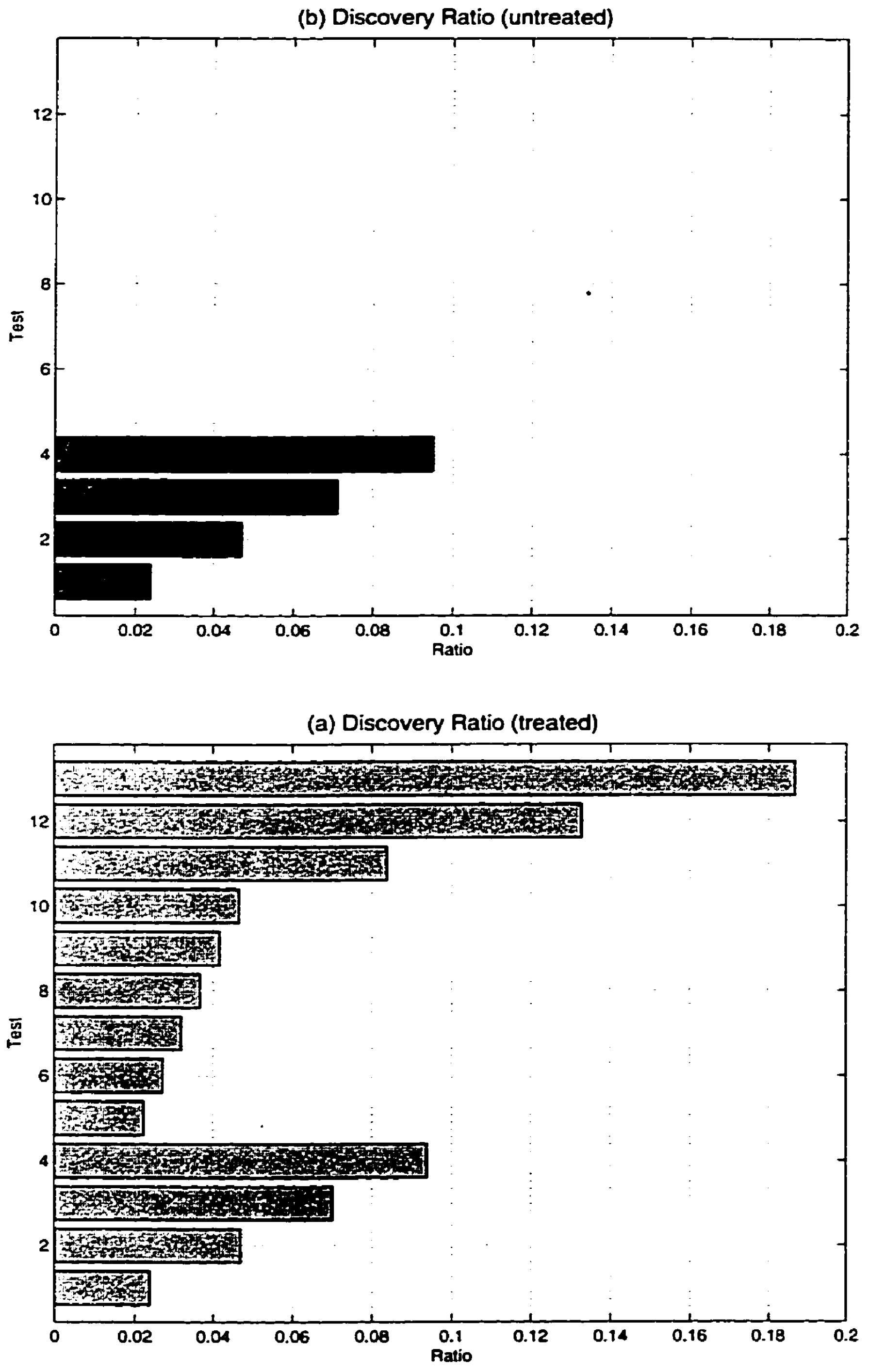
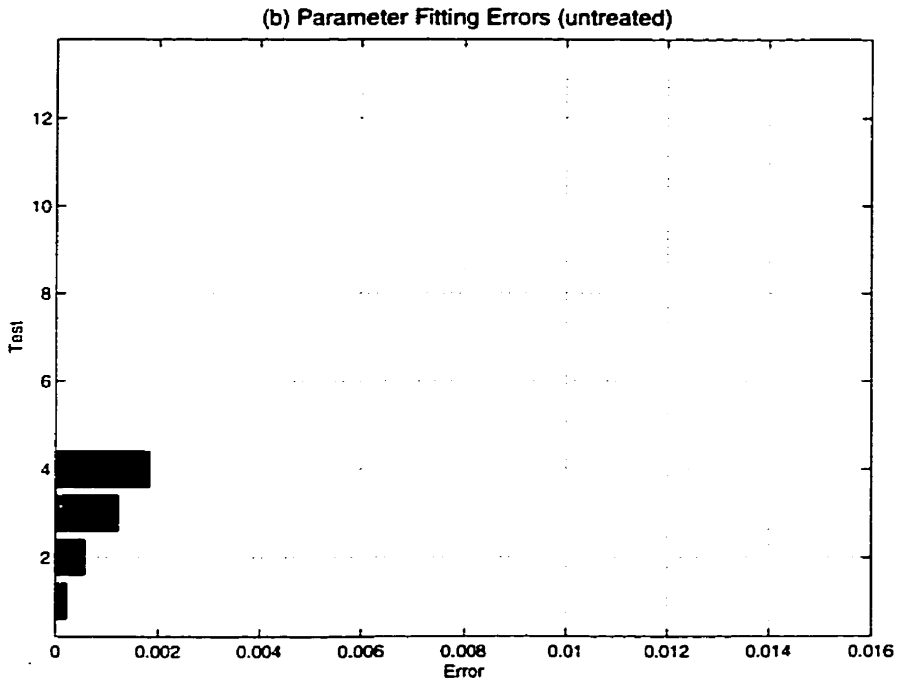


Figure 5.22: The Comparison of the Discovery Ratios of Form II



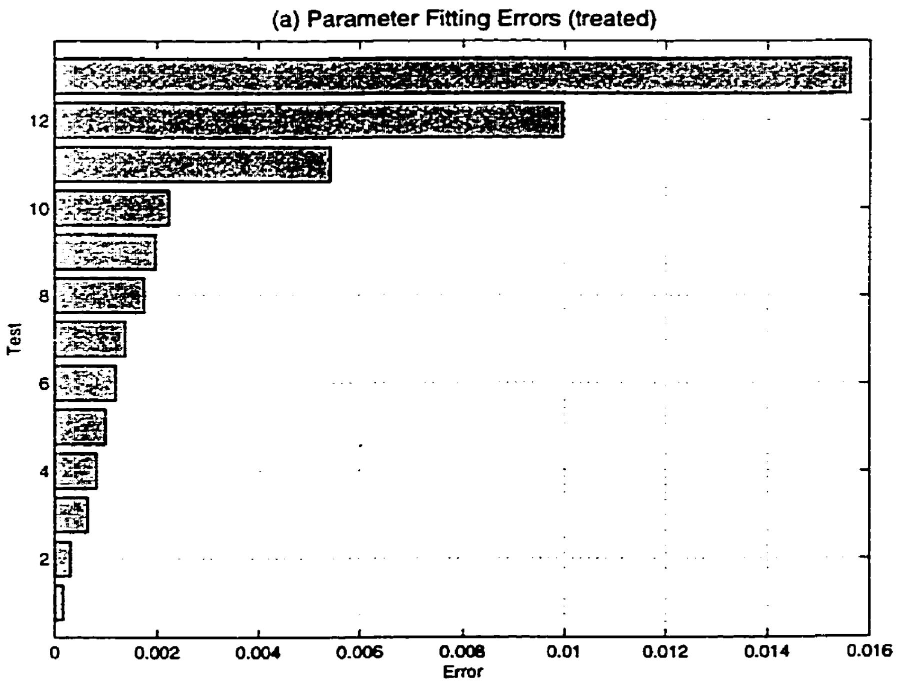
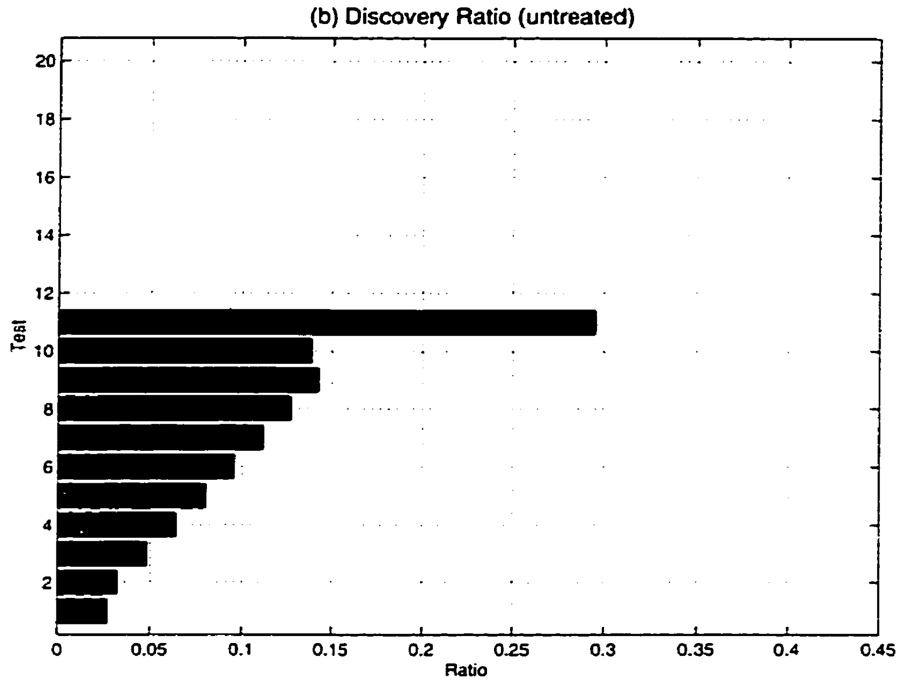


Figure 5.23: The Parameter Fitting Errors of Form II

Twenty one pairs of experiments were conducted to discover this description upon various input noise levels (See the table below. The symbols used here are the same as they appeared in the preceding sections). The observation domains were fixed to $(x, y) \in [-1, 1; -1, 1]$.

Test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Noise	10^{-3}	2×10^{-3}	3×10^{-3}	4×10^{-3}	5×10^{-3}	6×10^{-3}	7×10^{-3}	8×10^{-3}	9×10^{-3}	10^{-2}	2×10^{-2}	3×10^{-2}	4×10^{-2}	5×10^{-2}	6×10^{-2}	7×10^{-2}	8×10^{-2}	9×10^{-2}	10^{-1}	2×10^{-1}	3×10^{-1}
N–Solution	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
T–Solution	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$

Figure 5.24 and 5.25 are the comparisons of discovery ratios and descriptive parameter fitting errors. The error tolerance increases by 10 times when the proposed error-treatment scheme is employed. The other thing worth to mention is that the tolerable input noise of this experiment is much higher than that of the rest experiments. The reason is that the input noises are additive, and they are largely compressed by the transformation T_{LOG}. However, if multiplicative noise are used in the simulation of the noisy input, the noise tolerance level should be close.



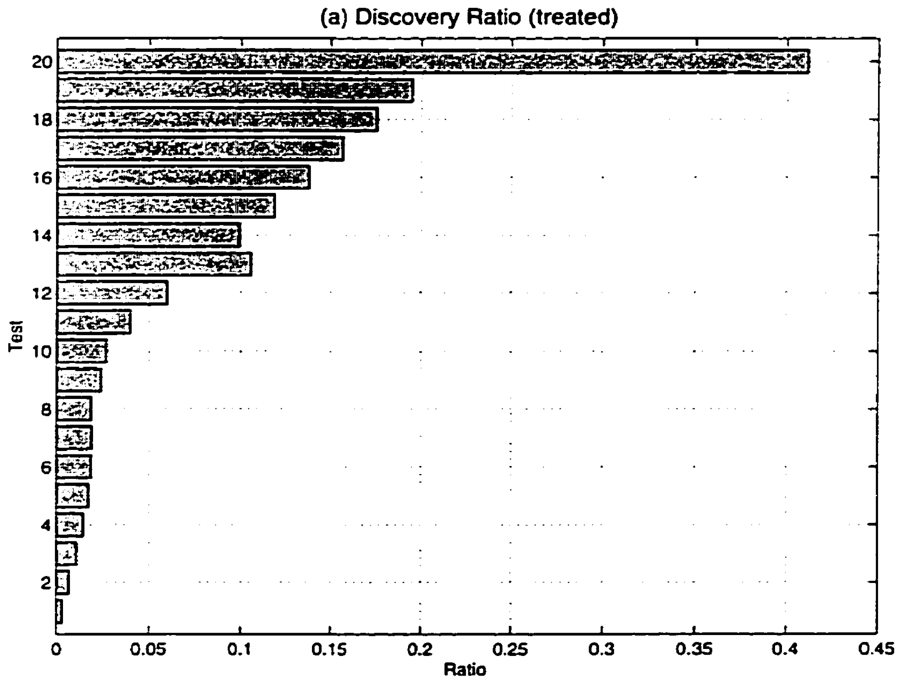
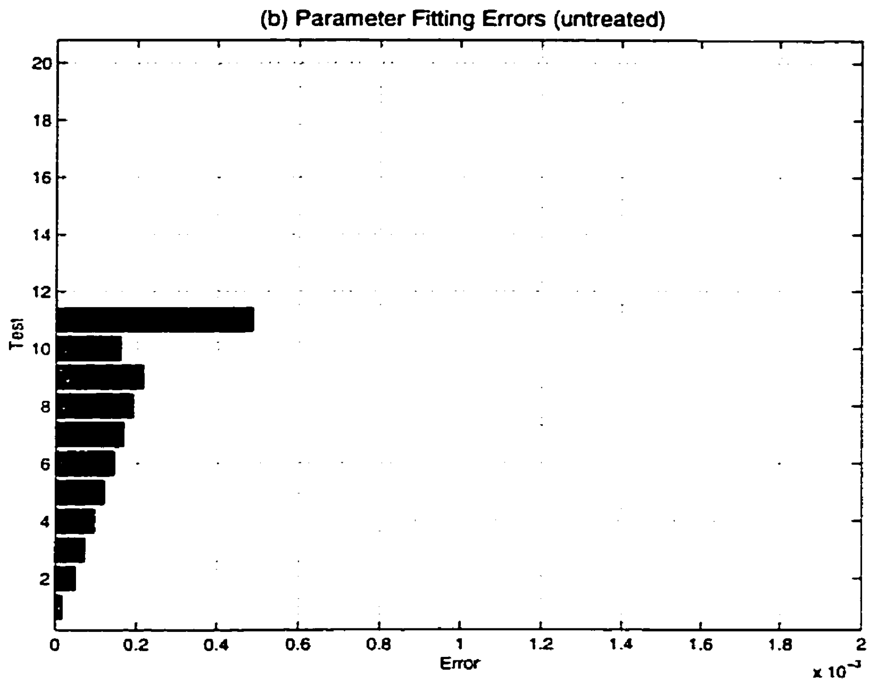


Figure 5.24: The Comparison of the Discovery Ratios of Form III



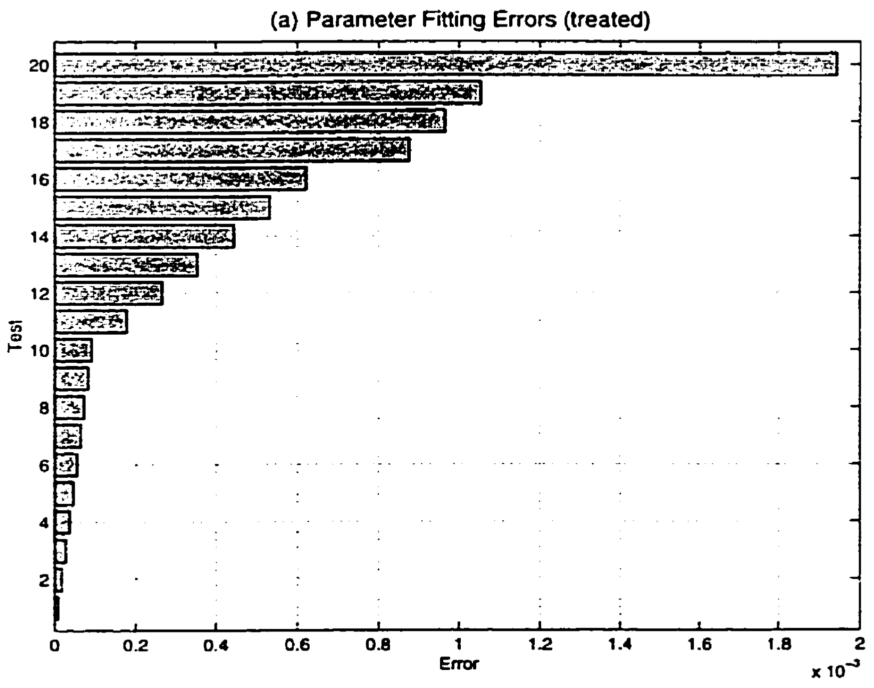


Figure 5.25: The Parameter Fitting Errors of Form III

The fourth function form to be discovered from noisy input is

$$z = \arctan(y/x) + y. \qquad \text{(IV)}$$

The solution of this form contains the only unused transformation T_{FAC}:

$$\{T_{\mathrm{REC}} \circ T_{\mathrm{FAC}}|\underline{y} \circ T_{\mathrm{DIF}},\ z = -x^2 - y^2\},$$

or similarly,

$$\{T_{\mathrm{REC}} \circ T_{\mathrm{DIF}} \circ T_{\mathrm{FAC}}|\underline{y},\ z = -x^2 - y^2\}.$$

Seven pairs of tests were conducted for different input noise levels. The observation domains were fixed to $(x, y) \in [0.5, 2.5;\ -1, 1]$ for all tests. The input noise levels and the test results are tabulated below.

Test	1	2	3	4	5	6	7
Noise($\times 10^{-4}$)	1.0	2.0	3.0	4.0	5.0	6.0	7.0
N–Solution	$\surd$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
T–Solution	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$

The results are plotted in Figure 5.26 and 5.27. Comparing with the results of the preceding noisy input experiments, we can clearly see that the improvement is not as good as before — only improving the noise tolerance by six times. Through carefully analyzing the discovery processes, the reasons are found to be

1. Numerically conducting factorization introduced very large computational errors near the zero points of the extracted factor in current implementation.

2. The differential transformation that follows the factorization further increases the error significantly at the points where large scale errors have been introduced by T_{FAC}.

3. The reciprocal transformation requires that the functional image has constant signs and the requirement is not satisfied (as it should be) due to the propagated errors introduced by the transformations T_{FAC} and T_{DIF}.

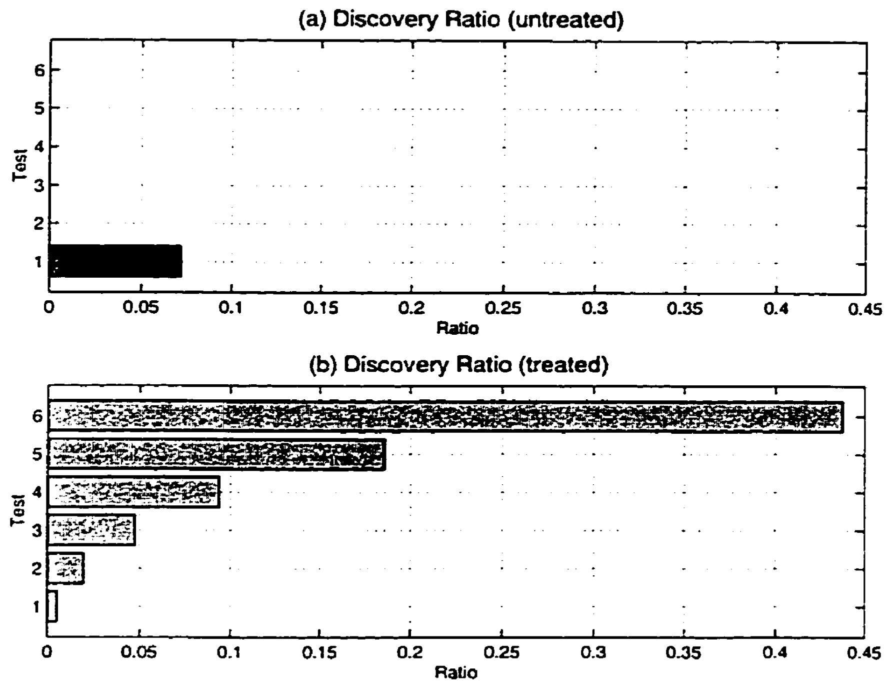


Figure 5.26: The Comparison of the Discovery Ratios of Form IV

When the propagated errors increase to the level at which the transformed functional image does not have constant signs as it should be in the observation domain, reciprocal transformation will not be applicable to the corresponding transformed image. That is the situation of this experiment. It prevents the system from making a successful discovery by applying the necessary transformation T_{REC}. This observation suggests future improvements in three directions. First, improve the accuracy of the computations of factorization transformation around the zero points of the factor. Second, improve the scheme of transformation validation check to enable it to handle noised exceptional points with large errors. Last, improve the smoothing method to handle sharp peak pattern, i.e. the points where the function values are significantly larger (or smaller) than the function values of their surroundings.

5.6.5 The Role of Adaptive Strategy

If we consider the size of a functional image, the smoothing processes is very expensive. Adaptive rules have been designed to avoid unnecessary image smoothing. In the current

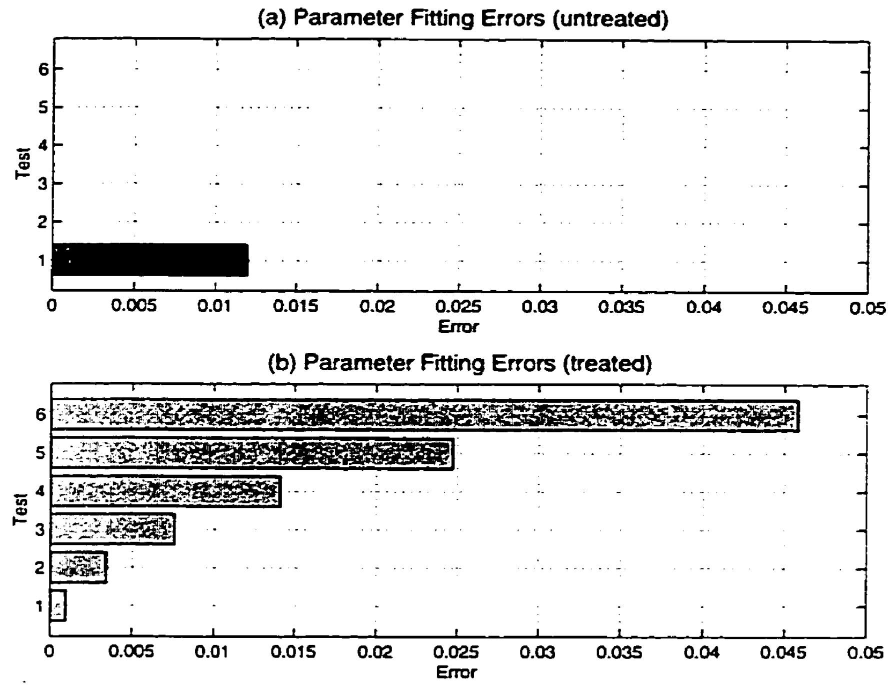


Figure 5.27: The Parameter Fitting Errors of Form IV

implementation, a functional image is only smoothed when

1. the image is not a smooth image measured by the roughness value of the image (Refer to 4.5.2), and

2. the image fits to a primitive well enough measured by the primitive fitting errors.

A smoothing call is viewed as a *Redundant* call if its associated search node is not on the path to a solution. Otherwise it is viewed as an *Effective* call. The ratio of the number of effective calls to the number of total smoothing calls reflects the efficiency of the designed adaptive strategies. We call this ratio the *Observed Adaptivity Efficiency Rate* (OAE-Rate):

$$\text{OAE–Rate} = \frac{\text{Number of Effective Smoothing Calls}}{\text{Number of Total Smoothing Calls}}.$$

Only OAE–Rate is not enough for measuring the effectiveness of the methodology. The ratio of the number of redundant calls to the total number of nodes the discovery system investigated is also an important measurement. We call this value the *Observed Adaptivity Redundancy Rate* (OAR–Rate):

$$\text{OAR–Rate} = \frac{\text{Number of Redundant Smoothing Calls}}{\text{Number of Total Explored Nodes}}.$$

Clearly, a large OAE–Rate and small OAR–Rate supports the adaptive error control scheme as an effective methodology. In the computations of the rates, only the accurate descriptions of the underlying function are counted as solutions to the discovery problems[22].

It is important to note that the two introduced measurements are prefixed with the term "Observed". This is because

1. In the carried out experiments, we limited the searching depth to 5. As such, the related numbers only reflect the situation of the specific threshold setting.
2. It is not practical to prove that a node is not on any solution path since we usually do not know whether or not the underlying function form can be expressed in some other different form. In other words, we can only claim that a node is not on the solution path of any known solutions within a limited depth.

Table 5.11 summarizes, in terms of OAE–Rate and OAR–Rate, seven discovery tasks that are conducted and reported in the preceding sections.

The averaged OAE-Rate is 62.1%. It means that more than half of the smoothing efforts were contributed to the discovery of accurate function form descriptions. The averaged OAR-Rate is 12.5%. It means that there were only 12.5% nodes that distracted the proposed adaptive strategy to do smoothing unnecessarily. The results justify the performances of the proposed adaptive strategies.

[22] Note that the observation data set are simulated images of known underlying functions. Although in principle an approximation with sufficiently small matching error should be viewed as a solution, we do not count them as solutions since the designed test functions have simple descriptions in $\tilde{\mathcal{L}}$ which have the smallest matching errors within the depth limit.

No.	Function Form[†]	Observation Domain	Noise Level	OAE-Rate (%)	OAR-Rate (%)
1	(I)	$[-0.5, 0.5; -0.5, 0.5]$	10^{-2}	100	0
2	(II)	$[0.2, 2.2; 0.2, 2.2]$	10^{-4}	66.6	19.0
3	(II)	$[0.2, 2.2; 0.2, 2.2]$	3×10^{-3}	42.9	12.9
4	(III)	$[-1, 1; -1, 1]$	10^{-3}	75.0	9.1
5	(III)	$[-1, 1; -1, 1]$	10^{-2}	60.0	22.2
6	(IV)	$[0.5, 2.5; -1, 1]$	10^{-4}	50.0	4.1
7	(IV)	$[0.5, 2.5; -1, 1]$	6×10^{-4}	40.0	20.0
[†]: Refer to the numbering of the test function forms in Section 5.6.					

Table 5.11: The Observed Efficiency of the Proposed Adaptive Error Control Strategies

5.7 Conclusions

The performances of the new function form discovery system have been justified from four different aspects. The experimental results shows that

1. Compared with previous discovery methodologies, which handle the discovery tasks only within a very limited function classes, the new system performs significantly well in discovering the underlying functional relationship among three relevant variables in a larger variety of function forms. The discovered form are accurate, compact and meaningful in terms of each description component representing an important significance of the observation.

2. As a direct model, the system provides the flexibilities in handling complex three-variable function form discovery problems. It extends the function form coverage of its previous system in the same category (data transformation approach) with a very compact and more powerful data transformation set.

3. The system has a better potential in performing informative data modeling than traditional numeric tools. Less human knowledge are needed for expressing the numeric observation data with compact and meaningful mathematic formulas.

4. The designed adaptive error control technique is effective and efficient in reduce the effects of the noises introduced by the source (additive noises) and the errors introduced by the numeric computations.

However, we had encountered with one elementary function form that is not discoverable by the system — Form #9 in Table 5.3. Examining the form carefully and doing manual simplification using the transformations defined in $\tilde{\mathcal{L}}$, we can find that T_{DIF} and T_{INV} make the form more massive, and T_{VEX}, T_{REC}, T_{FAC} and T_{LOG} can not help much. In fact, the trouble comes from the structure of the function form. The form is composed by a linear combination that links several functions. When differential transformation does not yield nice results directly, the system may suffer fatal problems in simplifying the form into a primitive. FFD-II shows its weakness in dealing with such forms that are linear combinations of functional operations '$\sqrt{\ }$', 'log', 'exp', 'tan' or 'arctan'. For example, Form #9 involves linear combination of tangent and logarithm functions. As has been pointed out, the powerfulness of a transformation based function form discovery system relies on the embedded data transformation set and primitive set. The current system is implemented in a preliminary fashion. The variety of three-variable function forms are too large to be handled with the function form description language defined for the current system. The experimental forms in the case studies in Section 5.4 and the reported failure cases in FFD documentation shows the same weakness of FFD in one dimensional situations. To identify more powerful transformations and to identify what function form cannot be easily simplified using certain set of data transformations are two equally important subjects for the research in this area.

Chapter 6

Conclusions and Future Research

The goal of function form machine discovery is to develop an autonomous system that can find symbolic descriptions that capture the underlying regularities hidden in the numeric data. In scientific and engineering studies, numeric data collected from the environment represent first-hand information. Therefore, a function form discovery system is an important part in an integrated machine intelligence system. It processes the given data and provides succeeding discovery or reasoning components with high level knowledge in the form of compact and easy to interpret symbolic mathematic formulas.

Function form discovery by data transformation was first introduced by Wong with the implementation of FFD in 1991. The work reported in this thesis is the first attempt to adopt this method in solving three-variable function form discovery problems. The FFD-II system, as reported here, was successful in solving a variety of such problems.

6.1 Research Contributions

The contributions of the research can be summarized as below.

1. A compact function form description language has been developed and used in the implementation of a direct data transformation based three-variable function form

discovery system called FFD-II. Formal definitions and analyses of the expressiveness and redundancy of the function form description language have been presented as a part of the theoretical results of this research. The analysis methods and results could be beneficial to further development of new systems for specifying new language components and identifying the redundancy of the language. The specified function form description language, the analyses of the data transformation method as a general function form discovery model, and the implemented system with extensive experimentation have furthered our theoretical understanding of quantitative discovery.

2. The implemented system demonstrated the flexibility of data transformation model in tackling function form discovery problems. The direct characteristic of the discovery mechanism of FFD-II allows the system to discover a significantly wider variety of function forms than its predecessors. The cumulative enhancement methodology, which has been demonstrated by the development of FFD-II (see Section 3.1.4), could be used to develop new enhanced data transformation based function form discovery system that adopts more sophisticated numeric analysis tools, application domain knowledge and new enhanced low dimension function form discovery implementations. The methodology could also be applied to the development of higher dimension system based on implemented systems. Such developments will contrast with *ad hoc* customizations like those being used to construct numeric analysis based function form discovery system. Mathematics analysis in the demonstrated way could be carried out to help the construction of new description languages and removing redundancy to achieve better efficiency.

3. The quantified measurement of the smoothness of curves and surfaces defined in this thesis is simple and has proven effective. Based on the measurements of image simplicity, the rank value of the transformation sequence and the quality of the image, searching heuristics has been defined for carrying out the best-first search. It has been demonstrated to be a simple and effective way to guide the system to find function form description that matches the given numeric data. Furthermore, the system

employs simple numeric primitive recognition and hypothesis verification algorithms. From the experimental results show that they can effectively distinguish "goodness" of the matches.

4. Special purposed numeric methods have been developed to conduct differentiation, functional pattern recognition, and surface smoothing. These algorithms are simple and effective. The methods could be used in the development of new high dimension systems or for other numeric analysis purposes.

5. The theoretical analysis of propagated errors corresponding to each numerically implemented data transformation not only provides valuable results but also demonstrates a general way to carry out such analyses for new data transformations. The designed error control strategies, including image refinement, smoothing and the heuristics for triggering the processes, establish an example for handling noisy input and monitoring the discontinuity of the transformed images.

6. The superiorities of the implemented system over its predecessors (most of them can only discover function forms within a very limited number of function form classes) have been proven by extensive experiments. *Firstly*, the experimental results on random selected functions show the great expressiveness of the designed function form description language. They also demonstrate the discovery system's great ability of generating accurate, compact and meaningful mathematic formulas to describe the given numeric data. This ability usually could not be achieved by using traditional numeric tools. *Secondly*, comparison experiments show the superiority of the direct multi-dimension function form discovery model over an indirect model from the expressiveness point of view. The direct model is also superior to an indirect model for its flexibilities to be extended. Domain knowledge, new advanced language components and new achievement in the field of function form discovery (including theoretical analysis results and improved working systems) could be incorporated more easily than an indirect system. *Thirdly*, the experiments on random surfaces suggest that, comparing with polynomial surface fitting, the discovery system may formulate the

given data in a flexible way to achieve better interpretability and to capture high order functional pattern more precisely. *Fourthly*, detail oriented experiment results on noisy input not only justify the effectiveness of the designed error control strategies, but also provide us with a chance to observe how errors affect each transformation, for example, the application of a certain transformation could be turned down by relatively low level noises or propagated errors. This observation suggests that it is necessary to verify the applicability of a transformation considering also the possible error effects. *Lastly*, the incompleteness of the description language is revealed by the case in which FFD-II failed to find the description. The identified special function structures, which may not be expressible in the function form description language $\tilde{\mathcal{L}}$, suggest future research directions to complete the language.

6.2 Future Research

Data transformation methods are a promising researching direction for automated function form discovery. Research with the developed systems in this category, e.g. FFD, LINUS, and FFD-II, are only beginnings. Many research issues remain open for mathematicians, computer scientists and application domain specialists to work together in this rich field. To conclude this thesis, I will briefly describe some possible directions of future research.

First of all, the experimentation shows that the function form description languages used by the existing systems, including FFD, LINUS, and FFD-II, have a major incompleteness (see Section 5.7). To carry out more experiments to identify more such incompletenesses and to conduct theoretical studies to identify and incorporate new language components that can help to handle identified incompleteness would highly enhance the expressivenesses of the existing systems.

Secondly, FFD-II is implemented by choosing simple numerical tools to carry out the numeric analysis tasks. More sophisticated methods could be employed to improve the accuracies of the numeric computations and the speed and memory efficiency of the system. More sophisticated supporting low dimension discovery systems could be used to improve the discovery of possible descriptive expressions.

Thirdly, in the current implementation, the transformation sequence associated with a possible functional hypothesis is not considered in the processes of primitive recognition and descriptive expression determination. Obviously, this may cause problems under certain circumstances, especially to those images distorted by noisy input or propagated errors. For example, when the last applied transformation is reciprocal transformation and the functional primitive fitting result is not a polynomial with constant sign in the corresponding observation domain, the verification process will immediately turn the hypothesis down. That makes the system unnecessarily sensitive to the observation domain. Developing new primitive recognition and descriptive expression extraction algorithms by taking into account the data transformations will enhance the performances of the current system, especially to the situations of noisy input and disccvering complex function forms that require lengthy simplification (i.e. transformation) steps to reach a recognizable primitive.

Fourthly, the composed search heuristics takes into account only a few basic facts. Possibly, new heuristics could be constructed based on the consideration of domain knowledge and the discovery experience of the system.

Fifthly, the error control strategies could be improved. Currently, the expected error analyses are not precise enough and the estimation is "globally". More accurate analyses results of the error propagations, probably point-wise, will help to improve the system's performance. Concerning surface smoothing we could improve the efficiency of the current system by selecting points with the highest discontinuity and smooth only the selected points under certain circumstances. And at the same time, new noise removing algorithm could be considered. However, handling noises is an important subject in engineering design and mathematics study. Many different methodologies have been developed to handle different types of noises. Developing noise tolerant function form discovery systems to handle "real-world" problems should be a very interesting research subject. Current implementation is only the first trial and it demonstrates only a possible way to handle noises. New research in this direction could start with a more thorough experimentation on the system with different function forms and different noise models. Comparing the performances of different techniques with extensive experiments is necessary for the development of a noise tolerant model.

Sixthly, theoretical investigations on intrinsic relationship between a given function form description language and the expressiveness of the language are of practical interests. Based on the results of **FFD** and **FFD-II**, one can improve the efficiency of the existing data transformation based function form discovery systems by identifying new redundant transformations or design new function form description language to acquire new discovery power. It is also an interesting direction to develop special function form description language that incorporates domain knowledge and solves the function form discovery problems in a particular application domain. The challenge involved in the theoretical investigations may require new abstract mathematic notions.

Seventh, to conduct new experiments on advanced function forms is an important research direction. In this research, the system has only been tested with analytic functions. Other function forms that can be used to verify the data transformation function form discovery model include ordinary differential equations, partial differential equations and integral equations. They might be of greater practical interests than analytic functions. Moreover, since the variables may be "coupled" more tightly in these types of function forms than in analytic function forms, experiments on the advanced function forms may provide us with a better way to understand the discovery model.

Other possible research directions include solving general multi-dimensional problems, integrating with a symbolic algebra system to provide symbolic solution verification, and integrating with qualitative reasoning systems to perform automatic interpretation of the investigated problem.

Bibliography

[1] Aho AV, Hopcroft JE and Ullman JD: *The design and analysis of computer algorthms.* Addson-Wesley, Menlo Park, CA, 1974.

[2] Barenblatt GI: *Dimensional analysis.* Gordon and Breach Science Publishers, New York 1987.

[3] Bradshaw G, Langley P, Simon HA: BACON.4: The discovery of intrinsic properties. *Proceedings of the Third National Conference of the Canadian Society for Computational Studies of Intelligence.* 1980.

[4] Brooks RA: Intelligence without representation. *Computation & Intelligence.* Luger GF (ed), AAAI Press, Menlo Park, CA 1995.

[5] Brooks RA & Connell JH: Asynchronous distributed control system for a mobile robot. *Proceedings SPIE.* Cambridge, MA. 1986.

[6] Brychkov YA, Glaeske HJ, Prudnikov AP and Tuan VK: *Multidimensional integral transformation.* Gordon and Breach Science Publishers, Philadelphia, 1992.

[7] Carbonell GJ, Michalski RS and Mitchell TM: An overview of machine learning. *Machine Learning: An Artificial Intelligence Approach.* Michalski RS, Carbonell JG and Mitchell TM, (eds), Pittsburgh, Pa., 1983.

[8] Collins JS: A regression analysis program incorporating heurisic term: selection. *Machine Intelligence 2.* Dale E and Michie D, (eds), American Elsevier, New York, 1968.

[9] DeBoor C: *Bicubic spline interpolation.* J. Math. and Phys. 41, 1962.

[10] Falkenhainer BC and Michalski RS: Integrating quantitative and qualitative discovery: ABACUS system. *Machine Learning.* 1, Kluwer Academic Publishers, Boston. 1990.

[11] Falkenhainer BC and Michalski RS: Integrating quantitative and qualitative discovery in the ABACUS system. *Machine Learning: An Artificial Intelligence Approach Vol.III.* Kodratoff Y and Michalski R (eds), Morgan Kaufmann Publishers Inc. California, 1990.

[12] Fisher D: Knowledge acquisition via incremental conceptual clustering. *Machine Learning.* 2, 1987.

[13] Gerwin DG: Information processing, data inferences, and scientific generalization. *Behavioral Science.* 19, 1974.

[14] Gleich J: *Chaos: Making a new science.* Viking, New York, 1987.

[15] Goetz A: *Introduction to differential geometry.* Addison Wesley Publishing Company, California, 1970.

[16] Gregory HG: *The ABACUS.2 system for quantitative discovery: Using dependencies to discover non-linear terms.* Technical Re-port MLI 88-17, George Mason University, Machine Learning and Inference Laboratory, June 1988.

[17] Hämmerlin G and Hoffmann KH: *Numerical Mathematics* Spriner-Verlag, New York, 1991.

[18] Hamming RW: *Digital Filters.* 3rd ed. Englewood Cliffs, NJ:Prentice Hall, 1989.

[19] Jones R: Generating predictions to aid the scientific discovery process. *Proceedings of the Fifth National Conference on Artificial Intelligence.* 1986.

[20] Koehn BW, and Zytkow JW: Experimenting and theorizing in theory formation. *Proceedings of the ACM Sigart International Symposium on Methodologies for Intelligent Systems.* Knoxville, 1986.

[21] Kokar MM: Discovering functional formulas through changing representation base. *Proceedings of the Fifth National Conference on Artificial Intelligence.* 1986.

[22] Kokar MM: Determining Arguments of Invariant Functional Descriptions. *Machine Learning.* 1, 1968.

[23] Kulkarni D & Simon H: The Processes of Scientific Discovery: The Strategy of Experimentation. *Cognitive Science.* 12, 1988.

[24] Lancaster P, Salkauskas K: *A survey of curve and surface fitting.* University of Calgary, 1977.

[25] Langhaar HL: *Dimensional analysis and theory of models.* John Wiley and Sons, New York. 1951.

[26] Langley P: BACON.1: A general discovery system. *Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence.* 1978.

[27] Langley P: *Descriptive Discovery Processes: Experiments in Baconian Science.* Ph.D. Th., Department of Psychology, Carnegie–Mellon University, 1979.

[28] Langley P: Data-driven discovery of physical laws. *Cognitive Science.* 5, 1981.

[29] Langley P: Strategy acquisition governed by experimentation. *Proceedings of the European Conference on Artificial Intelligence.* Paris, 1982.

[30] Langley P: Learning search strategies through discrimination. *International Journal of Man-Machine Studies.* 18, 1986.

[31] Langley P: Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science.* 9, 1986.

[32] Langley P, Bradshaw G, Simon HA: BACON.5: The discovery of conservation laws. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* 1981.

[33] Langley P, Bradshaw G, Simon HA: Data-driven and expectation-driven discovery of empirical laws. *Proceedings of the Fourth National Conference of the Canadian Society for Computational Studies of Intelligence.* 1982.

[34] Langley P, Bradshaw G, Simon HA: Rediscovering chemistry with the BACON system. *Machine Learning: An Artificial Intelligence Approach.* Michalski RS, Carbonell JG, Mitchell TM, (eds), Tioga Press, Polo Alto, CA, 1983.

[35] Langley P, Bradshaw G, Zytkow J, Simon HA: Three facets of scientific discovery. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* 1983.

[36] Langley P, Ohlsson S, Thibadeau R, Walter R: Cognitive architectures and principles of behavior. *Proceedings of the Sixth Conference of the Cognitive Science Society.* Boulder, Colorado, 1984.

[37] Langley P, Zytkow J, Simon HA, Bradshaw GL: The search for regularity: four aspects of scientific discovery. *Machine Learning: An Intelligence Approach Vol. II.* Michalski RS, Carbonell JG, Mitchell TM, (eds), Tioga Press, Palo Alto, CA, 1986

[38] Langley P, Simon HA, Bradshaw GL, and Jan MZ. *Scientific Discovery: Computational Explorations of the Creative Processes.* MIT press, 1987.

[39] Langley P, Simon HA, Bradshaw GL: Heuristics for Empirical Discovery. *Computational Models of Learning.* Leonard Bolc (ed), 1987.

[40] Langley P, Simon HA, Bradshaw GL & Zytkow JM: *Scientific explorations of the creative process.* Cambridge, MA: MIT Press. 1987.

[41] Langley P and Zytkow J: Data-driven approaches to empirical discovery. *Artificial Intelligence.* 40, 1989.

[42] Lebowitz M: Experiments with incremental concept formation: UNIMEM. *Machine Learning.* 2, 1987.

[43] Lenat DB: Automated theory formation in mathematics. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.* 1977.

[44] Michalski RS: Understganding the nature of learning: issues and research directions. *In Machine Learning, An Artificial Intelligence Approach Vol II.* Michalski RS, Carbonell JG, Mitchell TM (eds.). M. Kaufmann Publishers, 1986.

[45] Michalski RS: Learning Strategies and Automated Knowledge Acquisition. *Computational Models of Learning.* Leonard Bolc (ed), 1987.

[46] Nordhausen B, Langley P: robust approach to numeric discovery. *Proceedings of the 1990 International Conference on Machine Learning.* Morgan Kaufmann Publishers Inc., California, 1990.

[47] Nordhausen B, Langley P: An integrated approach to empirical discovery. *A Computational Models of Scientific Discovery and Theory Formation.* Shrager J and Langley P (eds.), Morgan Kaufmann Publishers, Inc., 1990

[48] Phan TH: *Function discovery using data transformation.* Ph.D Th., Department of Computer Science, University of Calgary, Alberta, Can. 1994.

[49] Rao RB & Lu SCY: KEDS: a knowledge–based equation discovery system for engineering problems. *A Proceedings of the Eighth Conference on Artificial Intelligence for Applications.* Monterey, California. IEEE Press, CA, 1992.

[50] Samuel AL: Some studies in machine learning using the game of checkers. *Computer and Thought.* Feigenbaum EA and Feldman J (eds), McGraw-Hill NY, 1963.

[51] Schaffer C: Bacon, data analysis and artificial intelligence. *Proceedings of the Sixth International Workshop on Machine Learning.* 1989.

[52] Schaffer C: An environment/classification shceme for evaluation of domain independent function-finding programs. *Proceedings of the IJCAI Workshop on Knowledge Discovery in Databases.* 1989.

[53] Schaffer C: *Domain-Independent Scientific Function Finding.* Ph.D. Th., Rutgers University, 1990.

[54] Schrager J and Langley P: Computational Approaches to Scientific Discovery. *Computational Models of Scientific Discovery and Theory Formation.* Shrager J and Langley P (eds), Morgan Kaufmann Publishers, Inc., 1990.

[55] Simon HA: Why should machines learn? *Machine Learning: An Artificial Intelligence Approach.* Michalski RS, Carbonell JG, Mitchell TM, (eds), Pittsburgh, Pa., 1983.

[56] Simon HA: Machine as Mind. *Computation & Intelligence, Collected Readings.* Luger GF (ed.), AAAI Press, CA 1995.

[57] Simon HA and Lea G: Problem solving and rule induction: a unified view. *Knowledge and Cognition.* Gregg L (ed.), Lawrence Erlbaum Associated, Hiils dale, New Jersey, 1975.

[58] Sneddon IN: *Fourier Transformations.* McGraw-Hill, New York, 1951.

[59] Sneddon IN: *The Use of Integral Transforms.* McGraw-Hill, New York, 1972.

[60] Spath H: *Spline interpolation of degree three.* The Computer Journal, Vol. 12, 1969.

[61] Stepp RE: *Conjunctive conceptual clustering: a methodology and experimentation.* Ph.D. Th., Department of Computer Science, University of Illinois, Urbana, IL, 1984.

[62] Thagard P: *Computational Philosophy of Science.* Cambridge, Massachusetts: MIT Press, 1988.

[63] Thagard P and Nowak G: The Conceptual Structure of the Geological Revolution. *Computational Models of Scientific Discovery and Theory Formation.* Shrager J and Langley P (eds), Morgan Kaufmann Publishers, Inc., 1990.

[64] Valdés-Pérez RE: *Machine Discovery of Chemical Reaction Pathways.* Ph.D. Th. Department of Computer Science, Carnegie Mellon University, 1990.

[65] Wong P: *Mahcine Discovery of Function Forms.* Ph.D. Th., Department of Systems Design Eng., University of Waterloo, Ontario, Can. 1991.

[66] Wu YH: Reduction: a practical mechanism of searching for regularity in data. *Proceedings of the Fifth International Conference on Machine Learning.* Morgan Kaufmann Pub., 1988.

[67] Wu YH and Wang SL: Discovering knowledge from observational data. *Proceedings of the IJCAI Workshop on Knowledge Discovery in Databases.* 1989.

[68] Zagoruiko NG: Empirical prediction algorthms. *Computer Oriented Learning Process.* Simon JC(ed), Noordhoff, Leyden.

[69] Zytkow JM: Combining many searches in the FAHRENHEIT discovery system. *Proceedings of the Fourth International Machine Learning Workshop.* Irvine, CA., 1987.

[70] Zytkow JM and Simon HA: A theory of historical discovery: the construction of componential models. *Mahcine Learning.* 1. Kluwer Academic Publishers, Boston. 1986.

[71] Zytkow JM: Deriving laws through analysis of processes and equations. *Computational Models of Scientific Discovery and Theory Formation.* Shrager J and Langley P (eds), Morgan Kaufmann Publishers, Inc., 1990.

[72] Zytkow JM, Zhu J and Zembowicz R: Operational definition refinement: a discovery process. *Proceedings of the 1991 National Conference on Artificial Intelligence.* MIT Press, Massachusetts. 1991.