THE ABACUS.2 SYSTEM FOR QUANTITATIVE DISCOVERY:
Using Dependencies to Discover Non-Linear Terms

Gregory H. Greene

MLI 88-17
TR-11-88

A publication of *Machine Learning & Inference Laboratory*
Artificial Intelligence Center
George Mason University
Fairfax, VA 22030 USA
(703) 764-6259

Editor: R. S. Michalski
Assistant Editor: J. Zhang

MLI Reports replaces *ISG Reports* published until December, 1987 by the Artificial Intelligence Laboratory Department of Computer Science, University of Illinois Urbana, IL 61801.

THE ABACUS.2 SYSTEM FOR QUANTITATIVE DISCOVERY:
Using Dependencies to Discover Non-linear Terms

Gregory H. Greene

Martin Marietta Corp.
Denver, CO 80222

MLI 88-17
TR-11-88

June 1988

This report is based on the author's Master Thesis completed in the ISG group of the Department of Computer Science of the University of Illinois at Urbana-Champaign. The thesis is done under the guidance of Professor R. S. Michalski.

THE ABACUS.2 SYSTEM FOR QUANTITATIVE DISCOVERY:

Using Dependencies to Discover Non-linear Terms

ABSTRACT

Research on inductive learning addresses the problem of formulating general concepts from specific training events. The result is a system of concepts that can be used by the learner to handle new events. In most cases, the new knowledge is more compact and concise. There are many different methods for generalizing a set of examples. Most of these methods can be viewed as a heuristically-guided search through a space of possible concepts. Examples of such heuristics include the *closing interval* rule, the *climbing generalization* tree rule, and the *dropping condition* rule. These rules work well as long as the domain for the learning is primarily symbolic. That is, as long as the description space for the examples consists primarily of attributes whose values are either nominal or structure. But if the domain contains much numerical data, as is the case in physics and chemistry, then these rules are not applicable. Inductive learning within a quantitative domain can still be viewed as a heuristic search, but the heuristics and the operators must be different than those in symbolic domains. The resulting concepts are equations or empirical laws. Such learning is called *quantitative discovery*.

This report presents a methodology for quantitative discovery and describes an implementation called ABACUS.2, which is an extension of the program ABACUS. The main results of this research are the extension of the power of ABUCUS to allow the discovery of non-linear equations, the addition of curve fitting techniques, and a clustering algorithm.

ACKNOWLEDGEMENTS

I wish to thank Professor R. S. Michalski for his guidance and support. His suggestions throughout the research were most helpful. I would also like to thank Heedong Ko for his friendship, discussions in all areas of AI, proof reading, and contributing good ideas to this report. I would also like to thank Bob Stepp for helping us maintain the equipment.

This research was supported in part by the Defence Advanced Research Projects Agency under grant, administered by the Office of Naval Research, No. N00014-87-K-0874, in part by the Office of Naval Research under grant No. N00014-88-K-0226, and in part by the Office of Naval Research under grant No. N00014-88-K-0397.

1. INTRODUCTION

Machine learning is a field in Artificial Intelligence which attempts to develop computational models of learning. There are many definitions of learning. Minsky states that *learning is making useful changes in our minds* [Minsky, 1985], while Simon relates learning to efficacy: *learning denotes 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 effectively the next time* [Simon, 1983]. While Simon's definition is fairly functional, it does not seem to capture all of the aspects of learning, particularly those cases where knowledge is acquired that does not improve performance. Minsky's definition is not very useful from a computational standpoint – it's hard to characterise useful changes. A definition that is more functional yet still general is: *learning is constructing or modifying representations of what is being experienced* [Michalski, 1986].

Given this definition, the task for scientists in machine learning is to develop models and theories which can be used to build learning systems. Nobody knows what a "complete" learning system will look like but it will probably be composed of many learning procedures, each one employing a different strategy depending on the type of knowledge to be learned, the information available, the representation of this information, the task, and so on. For example, if the learning system knows that all men are mortal and that Socrates is a man, and if the task is to decide if Socrates is mortal, then a deductive procedure is sufficient to carry out the task. If, on the other hand, the system knows only that Socrates is a man and that Tom, Sam and Bill are all men and are mortal, then a procedure for induction is needed. Other possible strategies include rote learning, learning from instruction, and learning via analogy [Michalski, 1986]. This paper addresses one aspect of learning via induction.

Research on inductive learning addresses the problem of formulating general concepts from specific training events. The result is a system of concepts that can be used by the learner to

handle new events. In most cases, the new knowledge is more compact and concise. There are many different methods for generalizing a set of examples. Most of these methods can be viewed as a heuristically-guided search through a space of possible concepts[1]. Examples of such heuristics include the *closing interval rule*, the *climbing generalization tree rule*, and the *dropping condition rule*. These rules work well as long as the domain for the learning system is primarily symbolic. That is, as long as the description space for the examples consists primarily of attributes whose values are either nominal or structured. But if the domain contains much numerical data, as is the case in physics and chemistry, then these rules are not applicable. Inductive learning within a quantitative domain can still be viewed as a heuristic search, but the heuristics and the operators must be different than those in symbolic domains. The resulting concepts are equations or empirical laws. Such learning is called *quantitative discovery*.

To see the usefulness of quantitative discovery, consider the data in figure 1. The task for the learner is to derive equations which fit[2] the data. In this case, the equation which fits the data is $f = \frac{mv^2}{r}\cos(\theta)$. This formula describes the centrifugal force parallel to the road resulting from a vehicle traveling around a curve, an important factor in the design of highways. This example shows many of the difficulties which can arise in quantitative discovery, including noisy data and irrelevant variables: e.g., *length* was excluded from the final equation. Finding such unimportant variables is a key issue. Additionally, integrating symbolic information and finding multiple equations need to be addressed. These will be discussed in more detail in following chapters.

---

[1]Exceptions to this are those method which use *Explanation-based Generalization* or *Explanation-based Learning* [Mitchell, T.M. et al., 1986; DeJong, G. and Mooney, R., 1986]

[2]The term "fit", in a strict sense, means that the equation evaluates to a constant for all given events. However, this definition is relaxed later to allow for uncertainty.

f	r	m	v	θ	length
3221.6787	30	50	75	1.22	3.42
8121.7944	30	50	75	0.523	4.53
-4909.809	30	50	75	2.122	3.53
-3986.427	30	50	75	2.01	5.9
-2885.0676	20	46	52	4.23	4.51
1883.0659	20	46	52	5.02	2.97
6549.0776	23	61	50	6.44	5.6
3927.5881	23	61	50	7.22	3.12
2774.2974	20	30	44	0.3	4.29
3699.0632	20	40	44	0.3	5.33
-345.31192	27	48	53	1.64	4.27
300	20	3.955	40	0.324	6.1
500	493.6	50	75	0.5	3.98

Figure 1. Table of Data for Centrifugal Force Equation.

This paper presents a methodology for quantitative discovery and describes an implementation called ABACUS.2, which is an extension of the program ABACUS [Falkenhainer, 1985; Falkenhainer and Michalski, 1986]. The main results of this research are the extension of the power of ABACUS to allow the discovery of non-linear equations, the addition of curve fitting techniques, and a clustering algorithm. The next chapter discusses the important issues that arise in quantitative discovery. Chapter 3 describes related work, chapter 4 outlines the methodology used by ABACUS.2, and chapters 5, 6, and 7 describe the specific implementation. Chapter 8 presents some experimental results and chapter 9 concludes the paper and discusses potential future work.

2. ISSUES AND GOALS

There are many issues that affect the overall design of quantitative discovery systems. The assumptions made regarding them can affect the performance in real-world applications. This chapter discusses these issues and briefly describes how ABACUS.2 deals with them.

2.1. Method of Discovery

An important consideration is the method used to construct the equations. There are a variety of methods which have been explored, including curve fitting, dimensional analysis, and regression analysis. Both curve fitting and dimensional analysis require extensive domain knowledge which is not always available, especially in new domains. Another method, called the "two-dimensional" approach, has gained some support, particularly in AI. This technique forms new terms from two previous terms by looking at the projection of events onto a plane. It can be used in a heuristic search to find the equation. This is the basis for ABACUS.2's heuristic search algorithm.

2.2. Irrelevant Variables

Any learning system must be able to disregard some information while focusing on the more important information. In many applications of quantitative discovery, it is not yet known which variables are important for the final equation and which ones are not. For systems to be of any use in the real world, they must be able to make such distinctions.

2.3. Noisy Data

Data cannot be expected to be 100% accurate. There may be events which, for some reason or other, may have incorrect values for certain variables. This is especially true with data drawn from real experiments. This inaccuracy can be due to human error, intermittently faulty

equipment, or coarse measurements. Although noisy data is a problem for all types of learning, it is particularly bad in numerical domains.

2.4. Multiple Equations

According to Lagrange's theorem, there exists a (single) polynomial of arbitrary precision that fits the data. However, there are some cases in which it is better to use two or more simple equations, each of which fit a unique subset of the data, than to use one complex equation. For example, consider figure 2. The polynomial that fits the data is quite complex. A better solution is to split the data into two parts, one for those events with values of x in the range [0..5.0] and one for those with values in the range [5.1..30.0], and to fit each subset. In this case, the equation $y = 5x$ fits the first subset of events and the equation $x + y = 30$ fits the second subset.

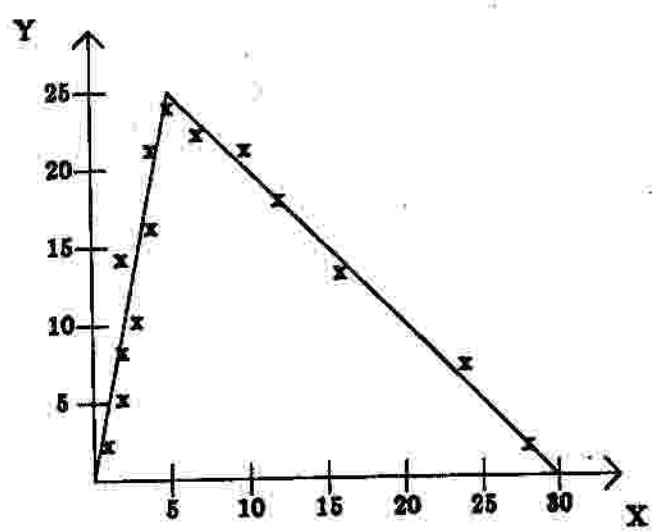

Figure 2.

To find multiple equations, ABACUS.2 uses an approach which divides the original set of events into subsets and attempts to find one equation for each subset. It uses information about the behavior of the values of a variable, relative to the values of all other variables, to hypothesize a clustering of the events.

2.5. Integrating Information From Symbolic Data

Systems for quantitative discovery should be able to use information from symbolic as well as numerical data. Most of the previous discovery systems, including those employing traditional statistical methods, ignored symbolic data (or did not allow it) (see chapter 3). An exception to this is ABACUS which used the symbolic data to form preconditions for each of the equations it generated. ABACUS.2 uses this same approach. After generating the set of equations, ABACUS.2 calls the inductive learning algorithm A^t [Becker, 1985] to generate a disjunctive cover for each equation. These covers indicate the conditions necessary for the corresponding equation to be applied. In addition to using symbolic information to build preconditions, ABACUS.2 uses it to help cluster the events.

2.6. Efficiency

Efficiency is a function of the background knowledge, the heuristics, and the discovery algorithm. The purpose of the heuristics is to increase the efficiency of the algorithm. An uniformed search will eventually find an equation that fits (unless it is a depth-first search) if one exists but is of no help for non-trivial equations. ABACUS.2 uses a set of heuristics which look at how values of one variable change with respect to other variables. Since heuristics are not guaranteed to work, ABACUS.2 employs a user-defined depth limit to constrain the maximum amount of search.

2.7. Background Knowledge

Another issue is just how much information is to be supplied by the user prior to discovering equations. For example, techniques using linear regression can "discover" general equations but they require extensive background knowledge – the general form of the equation must be given a priori. Algorithms using an uninformed search require no background knowledge at all but such algorithms are too inefficient for serious consideration. Discovery systems should become more efficient as more relevant information is provided[3]. ABACUS.2 allows the user to include certain optional information as background knowledge, such as the units of each variable, candidate equation forms, and possible terms for the final equation. Future work on ABACUS.2 could allow the use of other optional information such as "possibly" important and unimportant variables.

[3]Since efficiency is not a monotonically increasing function of the background knowledge, this will not always be the case.

3. RELATED WORK

Some of the first approaches to the problem of constructing equations was that of linear regression, regression analysis, and dimensional analysis [Edwards, 1976; Daniel and Wood, 1980; Chatterjee and Price, 1977; Langhaar, 1951; Huntley, 1952]. The basic approach of linear regression is to find values for the coefficients of an equation form that "best" matches observed values. A popular approach to determine the values is the least-squares method. Regression analysis uses linear regression to form the equation and then analyses the fit. It can suggest the addition of new variables or the removal of irrelevant ones. Dimensional analysis uses information regarding the dimensions of physical quantities to derive equations. All of these approaches require a fairly well-understood domain to work, but if little is known about the domain, they may produce erroneous results or simply not work very well. This is especially true in dimensional analysis if variables are omitted that influence the phenomenon. Irrelevant variables may appear in the final equation in some cases. Also, these methods assume that one equation fits the data which can result in an overly complex equation. Noisy data can also present problems as well.

The BACON series [Langley, Bradshaw, and Simon, 1985; Langley, Zytkow, Simon, and Bradshaw, 1986], with the exception of BACON.6, builds equations incrementally by using a heuristic search to form new terms. By looking for trends between two variables or terms, BACON can postulate new terms. This allows it to build and test the equation incrementally. BACON.4 was able to state conditions on symbolic variables and BACON.5 was able to learn from symmetry. BACON.6 differs from the approaches in the earlier versions in that the equation form must be known a priori.

One of the main problems with BACON is that it is limited in the type of equations it can discover (with the exception of BACON.6 but its power comes from the fact that it is given the

form a priori). It cannot discover non–linear terms. Additionally, it cannot find multiple equations which fit different subsets of the data.

Two approaches similar to dimensional analysis and linear regression are those used by Kokar and El–Shafei. El–Shafei's system HOTEP [El–Shafei, 1986] uses regression analysis to form the desired equation. He also uses dimensional analysis to help ensure dimensional homogeneity. Kokar's work on COPER [Kokar, 1986] attempted to address some of the problems with dimensional analysis. His system can detect missing variables by checking a condition of *complete relevance* and if it is violated, then new descriptions are generated. It can also find and discard irrelevant variables.

Neither of these approaches allow for multiple equations. They will derive one equation, no matter how complex it is. Also, they do not use information from symbolic data which can be important in many experiments.

ABACUS uses an approach for discovering equations similar to the one used by BACON. In addition, it is able to discovery multiple equations and use symbolic data to derive preconditions for each of the equations. Many of the problems that ABACUS faces are the same ones faced by BACON. The type of equations it can discover are those of the form $f(x)$=*constant*, where $f(x)$ is composed of variables combined using the operations multiplication, division, addition, and subtraction. The only way it can discover non–linear terms is in those cases where it can find a trend between two variables. For example, to generate $y=x^2$, most of the values for x must be either ≥ 0 or ≤ 0, but not *both*. Also, in order to discover multiple equations, ABACUS relies on symbolic data. Each time it generates a new term, it tests the term to see if it fits events which belong to the same *nominal subgroup*. A nominal subgroup is a set of events which have identical values for all nominal (symbolic) attributes. If so, the events, associated with the corresponding equation, were removed from further consideration. The problem with this approach is that it

relies on symbolic data which is not always available. Additionally, it divides the events *during* the search. If this was done before the search, curve fitting techniques could be used to help find multiple equations.

4. ABACUS.2 METHODOLOGY

There are two techniques used by ABACUS.2 to construct the equations. If the user includes in the data a set of equation forms, then it uses the least-squares method to determine the coefficient values for each form and then measures the resulting equations. If one exceeds a user-defined parameter, then the process stops. If there are no equation forms, or if the curve fitting approach fails, then ABACUS.2 uses an approach similar to that used by ABACUS and BACON with the exception that it has more powerful heuristics. This approach is to repeatedly create new terms based on trends between pairs of variables or terms generated from previous iterations until one of the new terms evaluates to a constant for a user-defined percentage of the events. Consider the simple graph in figure 3. Given data corresponding to the graph, ABACUS.2 would find that since the values of x tend to increase as the values of y decrease, the term

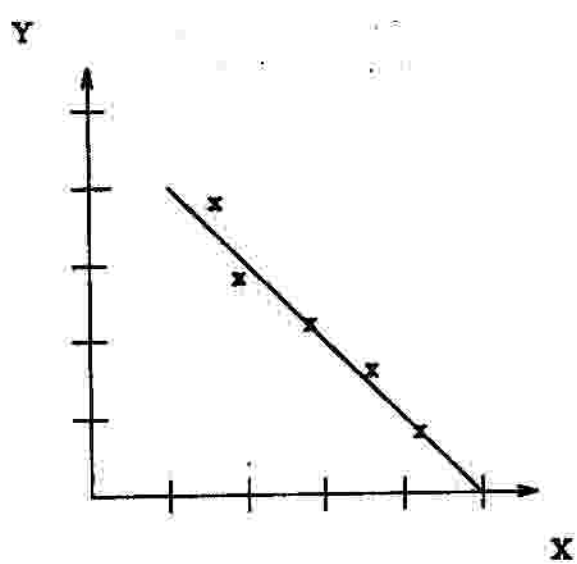

Figure 3.

$xy=constant$ or $x+y=constant$ might be important in the final equation. In this case, $x+y$ evaluates to a constant for most of the events so this equation would be outputted. More complex equations are built up over many iterations. Given data for the equation $\frac{x}{y}z=constant$, ABACUS.2 might first create the term $\frac{x}{y}$ after noticing that the values of z tend to increase as the values of y increase. The next iteration, it would find that the values of $\frac{x}{y}$ tend to decrease as the values for x increase and assert the term $\frac{x}{y}z$. Since this term evaluates to a constant, the algorithm halts.

This technique of searching for trends between pairs of variables works well as long as such trends actually exist. But in cases where there are non-linear terms, it is not possible (in most cases) to check to see whether one variable's values increase or decrease whenever another variable's values increase (or decrease). For example, in the equation $y=x^2$, the values for y tend to decrease as the values for x increase when $x<0$ but the y values increase when $x>0$. In this case, nothing can be concluded regarding the overall trend. ABACUS.2 solves this problem by approximating the *derivative* of the trend. In the example $y=x^2$, it would see that there are two different trends, one when $x<0$ and one when $x>0$. It uses this observation along with other information to hypothesize the existence of the term x^2 in the final desired equation. Chapter 6 discusses this new approach in more detail.

4.1. Monotonic Dependencies

The power of ABACUS.2's ability to discover equations comes from the use of *monotonic dependencies*, a term which describes the behavior of the values of one variable relative to the behavior of the values of another variable. Consider the examples in figure 4. The first example shows that the values of z increase as the values of y increase for most of the events. In this

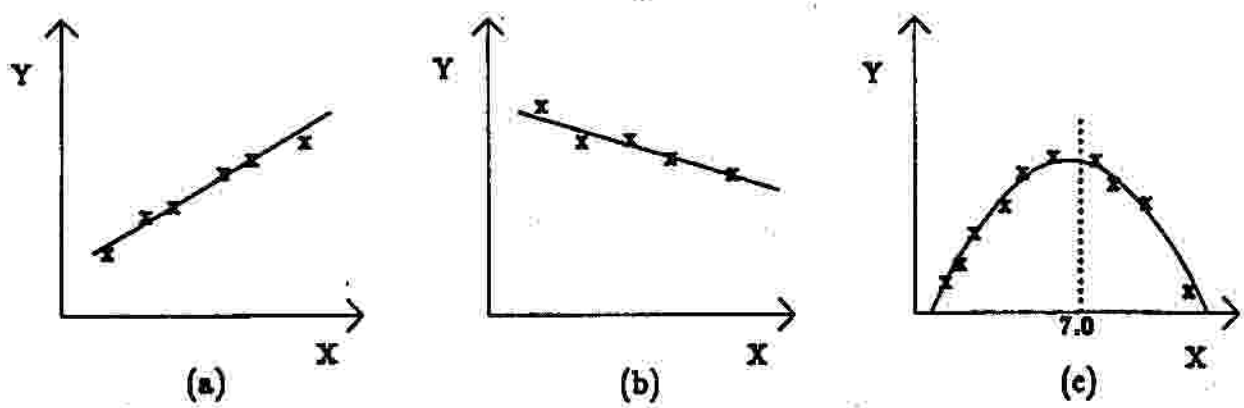

Figure 4. Examples of different monotonic dependencies.

case, we say that x is *approximately positively monotonic* to y. In the second example, the values of x tend decrease as the values of y increase implying that x is *approximately negatively monotonic* to y. More formally, x is approximately positively monotonic to y if, for a certain percentage of the events (determined by the user), the values of x increase (decrease) when the values of y increase (decrease). Similarly, x is approximately negatively monotonic to y if the values of x decrease (increase) when the values of y increase (decrease) for a certain percentage of the events. This paper uses the terms $amon^+(x,y)$ and $amon^-(x,y)$ to refer to approximately positively and negatively monotonic, respectively. Note that $amon^+(x,y) \leftrightarrow amon^+(y,x)$ and $amon^-(x,y) \leftrightarrow amon^-(y,x)$.

Referring to the example in figure 4c, since it appears that for about the first half of the events the dependency is $amon^+(x,y)$ and the second half it is $amon^-(x,y)$, what can be said about the dependency for all of the events? Note that there seems to be two types of dependencies: the *local* dependency and the overall or *global* dependency. As the terms suggest, the global dependency refers to the dependency for all of the events while the local dependency applies to a

subset of the events. Looking at figure 4c, it is easy to see that in terms of the local dependencies, for $x = [0..7]$ we have $amon^{+}(x,y)$ and for $x>7$ we have $amon^{-}(x,y)$. The global dependency is neither positive or negative. The term $\neg amon(x,y)$ is used to refer to this situation. The distinction between global and local dependencies becomes important later on.

When there are more than 2 variables involved, the task of finding the dependencies is slightly more difficult due to the fact that one variable can be affected by more than one other variable. Suppose that there are 3 variables involved, x, y, and z. To calculate the dependency between x and y, the influence of z must be taken into account. ABACUS.2 solves this interdependence problem by considering only groups of events in which the values of z are constant for each group. For example, consider the following data for variables x, y, and z:

event	x	y	z
1	3.4	0.9	12.3
2	4.8	1.2	12.3
3	1.2	8.0	2.9
4	0.8	6.5	1.7
5	0.1	21.3	16.7
6	12.0	26.7	16.7

If we want to find the dependency between x and y, we would place the events in groups in which z is constant. There are two such groups: (event1, event2) and (event5, event6). The events 3 and 4 cannot be placed in any group since z is not constant. From these two groups, we can see that the dependency is $amon^{+}(x,y)$. If no group exists, then $amon^{?}(x,y)$ is asserted indicating that nothing can be said about the dependency.

In the general case, ABACUS.2 must hold all variables constant except for those in the *exclusion set* when calculating the dependency for any two variables. The exclusion set consists of the two variables whose dependency is being calculated and the subcomponents of these two variables. The subcomponents of a variable are the user-defined variables which compose it. It

would not make sense to require that the subcomponents be held constant. Additionally, the user can specify a set of variables which must be held constant. There are cases where holding some user-defined variables constant will prevent ABACUS.2 from finding the desired equation. This set is added to the exclusion set. As evident in the above example, ABACUS.2 must sort the events before it can calculate the dependency. To do this, it sorts using a multiple key defined as follows: let x and y be the pair of variables whose dependency we desire, *EXSET* be the exclusion set, and *TVARS* be the set of all variables. Then the key = $\{TVARS-EXSET, x, y\}$. Sorting the events by this key places those events which have constant values for all variables except those in the exclusion set next to each other.

4.2. System Overview

There are 3 main components of ABACUS.2: clustering, equation discovery, and the covering algorithms. The first one is the clustering algorithm and its job is to divide the events into subsets depending on the dependencies and the symbolic data. As stated before, the goal of the clustering algorithm is to group the events into subsets such that each subset corresponds to

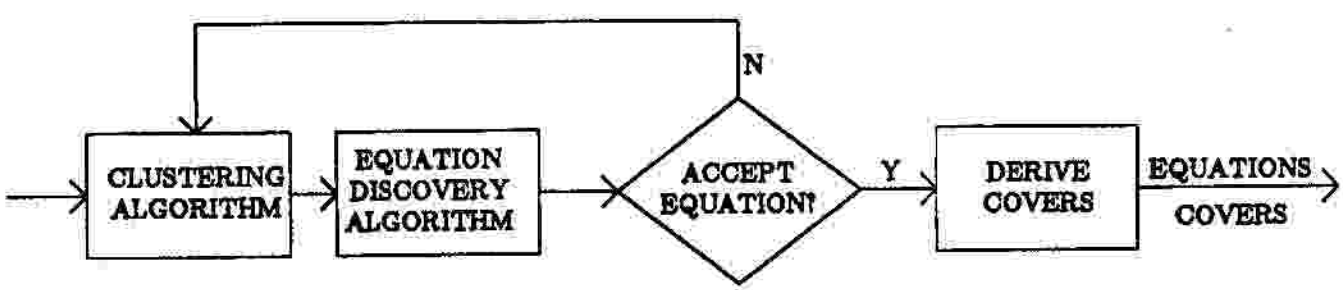

Figure 5. System Overview.

one equation and that the set of equations resulting are more comprehensive than the equation formed if no clustering was performed. A mutually exclusive and exhaustive grouping of the events is called a *cluster*. Each subset is passed on to the second component: the equation discovery algorithm. This algorithm outputs a list of equations, possibly empty, which fit the events in each subset (at most one equation per subset). If the list is of size 2 or more, the inductive learning algorithm A^t is called to develop the covers for the equations and the equations along with their preconditions are returned. However, if the list of equations is empty, then the process is repeated using a new cluster. Figure 5 shows the overview of the system. The user has several options which affect the flow of control, including "turning off" the clustering component and calling the discovery algorithm on each of the remaining clusters. These options are discussed in a later section.

5. CLUSTERING ALGORITHM

The clustering algorithm is used to split the original event set into subsets in such a way that there exists one "comprehensible" equation for each subset. It generates a set of possible clusterings and tests each one. To illustrate the approach used by the clustering algorithm, consider the graph in figure 2 again. In this example, the local dependency for the points $x \leq 5$ is $amon^{+}(x,y)$ but for the points $5 < x \leq 30$, it is $amon^{-}(x,y)$. This seems to indicate that it might be better to attempt to fit one equation for the events in which $5 < x \leq 30$ and another equation for the other events. ABACUS.2 looks for changes in the dependency to help it decide how to split the events.

The main approach of the clustering algorithm is to form subsets of events for each pair of variables (x,y) such that the events in each subset have the same dependency between x and y. Besides using these clusterings as candidate subsets, the algorithm also generates another clustering which uses information from the previous ones. The result is a list of candidate clusterings. The clustering algorithm is described in figure 6.

5.1. Forming the Initial Clusters

ABACUS.2 builds a list of candidate clusters by first splitting the events into groups for each pair of variables based on changes in the local dependencies. For each pair of variables (x,y), ABACUS.2 sorts the events according to the multiple key sort described previously ("projecting" the events) and then it splits the events into groups in such a way that the events in each group have the same values for all variables not in the exclusion set. Splitting the events in this manner results in a set of groups in which the only changes are those of x and y, thus removing any effect of the other variables. Then it finds the dependency for each group, stores it with the group, and builds a structure called a *projection* which contains all the groups

1. For each pair of variables (x,y), do:
   1.1 "Project" the events onto a 2–dimensional plane with x and y axes.
   1.2 Split the events into groups, where each group contains events which have the same values for all variables not in the exclusion set. Calculate the dependency of each group and store the groups in a structure called a projection.

2. For each projection, do:
   2.1 Merge as many groups together as possible.
   2.2 Call A^t to find a cover for each group.
   2.3 Add to each group any "unclaimed" event which satisfies its cover.

3. Build the group support graph.

4. Use a form of the greedy algorithm to choose the best groups based on the group support graph. Add these groups to a new projection.

5. Return an ordered list of the projections.

Figure 6. Clustering Algorithm.

corresponding to (x,y). Thus, a projection might look like:

```
(PROJECTION :NAME (X Y)
            :GROUPS ((GROUP :NAME G2
                            :AMON amon+(x,y)
                            :EVENTS (E1 E6 ...))
                     .
                     .
                     .
))
```

Each group is stored in a structure which includes the name, the dependency, and the events. The projection name is simply the list of the two variables involved in the projection. Because it is not always possible to place all the events into a group such that all the appropriate variables

are held constant, there is usually a fair number of events which are not included in any group for a given projection. This problem is addressed later.

5.2. Merging the Groups

After building projections for all pairs of variables, ABACUS.2 merges the groups in each projection. The purpose of the merging is to avoid the situation where there are two adjacent groups which have the same dependency. Two types of merge algorithms were tested: *conservative merge* and *liberal merge*. Conservative merge merges two groups if and only they have the same dependency and there are no "unclaimed" events in between them (by "unclaimed" we mean an event which has not been placed in a group). Liberal merge requires only that the two groups have the same dependency. As its name suggests, liberal merge will combine more groups than the conservative merge and empirical results indicate that the former is the better of the two. It may result in incorrect merges due to the lack of information about the dependency of the events *in between* the two groups which have been merged, but this is an acceptable inductive leap – adding more events can remedy this.

At this point, each projection can be interpreted as a cluster of events, where each cluster indicates a change in the global dependency. However, the projections were formed strictly on the basis of numerical information. Many times it is helpful to complement this approach with information gained from the symbolic data as well. ABACUS.2 uses information from the symbolic data to help add more events to the groups by passing each projection to A^t which forms covers for each group in the given projection. Each cover is formed by considering one group as the positive events and all other groups as the negative events. ABACUS.2 uses these covers to determine which group the "unused" events belong to. Chapter 7 discusses the A^t algorithm and its use in more detail.

5.3. Creating the Global Projection

ABACUS.2 could simply use the current list of projections as the possible clusterings. However, each projection uses information only from the behavior of the dependency between 2 variables. To use information from all of the variables, ABACUS.2 forms a *group support graph* which allows it to use a form of the greedy algorithm to create a "global" projection. A group support graph shows the "support" that a particular group from a projection has from the other projections. The nodes are the groups and the links indicate the amount of support one group has from another group. For each group G_i^k in the kth projection, ABACUS.2 finds the "best" group G' in each of the other projections and creates a directed link from G_i^k to G'. The "best" group contains the greatest number of events in common with G_i^k. Each link is labeled with a number indicating the percentage of intersection. A high percentage implies positive support. There is also a filter which prevents any links from being created if they are under a certain user–defined percentage. Therefore, if the percentage of intersection between the current group and the corresponding "best" group in another projection is below a threshold, then no link is created. This filter has a large influence on the resulting global projection as will be shown. Figure 7 gives an example of a group support graph for 3 projections.

At this point, each group has $\leq p-1$ links, where p is the number of projections. The average of all the percentages of the links from a group is an indication of the global support. That is, the average indicates how much support exists from the other projections for the hypothesis that the group should be included in the final clustering. Therefore, the next step is to sort all of the groups in the support graph in decreasing order of support, that is, average link percentage. Since there will probably be several groups which have events in common with each other, ABACUS.2 starts at the beginning of the list and removes all groups which have any events in common with one of the groups in the list before it. This assures a set of mutually

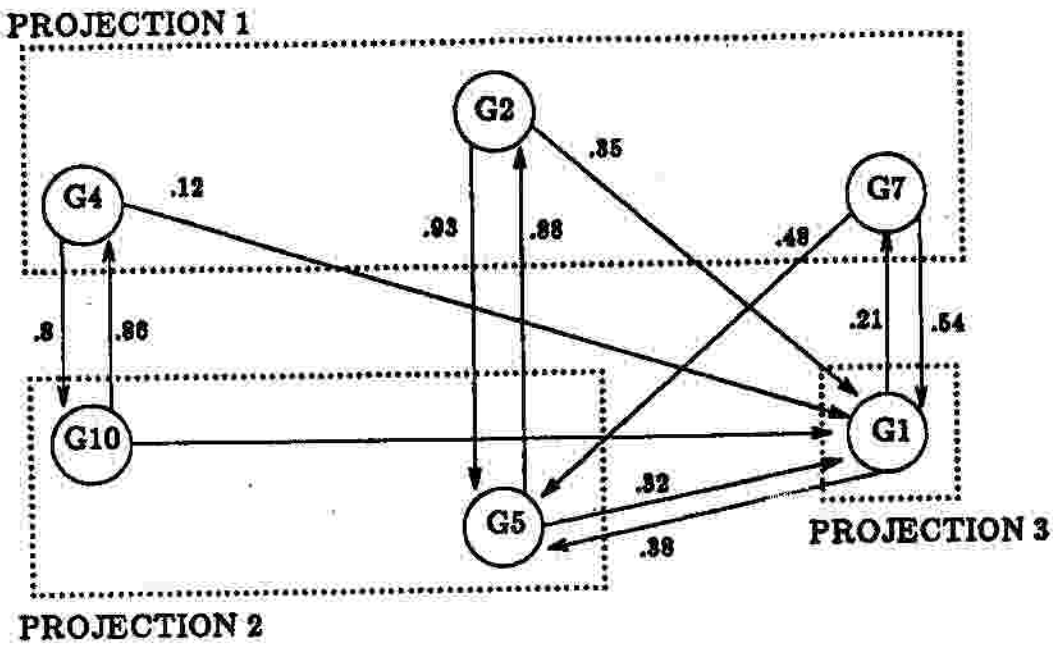

Figure 7. Example of a Support Graph.

exclusive groups of events.

Referring back to figure 7, we can see that there seem to be strong links between groups G4/G10 and G2/G5 and weak links between all others. Intuitively, this implies that the global projection should have 2 groups, either (G4 G2), (G4 G5), (G10 G5), or (G10 G2). However, this depends on the strength of the filter. Consider the support for each group when the filter strength is 0% (no filtering) and compare that to a strength of 30% (links whose percent intersection is $\leq$ 30% are not allowed):

	filter=0%	filter=30%
SUPPORT(G1):	.295	.295
SUPPORT(G2):	.640	.640
SUPPORT(G4):	.460	.800
SUPPORT(G5):	.550	.550

SUPPORT(G7):	.510	.510
SUPPORT(G10):	.500	.860

SUPPORT(G) is a function indicating the average of all links leaving node G. The main difference between the two filter levels is that the support for G4 and G10 increase drastically when the filter is changed from 0% to 30%. The order of decreasing support for the nodes in the first case is G2, G5, G7, G10, G4, and G1. However, with the higher filter, the order for the nodes would be G10, G4, G2, G5, G7, and G1. This illustrates the necessity for the filter and shows how different levels can alter the final "global" projection. Currently, the filter level is set to 50%, which appears to be sufficient based on empirical results.

ABACUS.2 now has a list of projections of length p+1: one from each pair of variables and one from the support group graph. One important point needs to be brought out. The clustering algorithm builds clusters by looking for changes in the dependencies. However, the heuristics for generating non-linear terms look at these changes to decide if such terms do exist. If the events are split up such that the dependencies do not change, then these heuristics will not work. ABACUS.2 creates one more projection which contains *all* of the events and includes that in the list which is passed to the equation discovery component. This allows for the discovery of non-linear terms. Future research should address this problem. One possible solution is discussed in chapter 9. The final step is to order them. This ordering depends on the values of some of the user-definable parameters but the default order is the global projection first, then the projection which contains all of the events, followed by all other projections, in the order that they were created. Once they are ordered, the projection list is passed on to the equation discovery component.

8. EQUATION DISCOVERY

ABACUS.2 relies on two methods to discover equations: curve fitting and heuristic search. The curve fitting method employs the least–squares algorithm and is used whenever the user inputs a list of candidate equation forms. The heuristic search is used if no such forms are provided or if the curve fitting algorithm fails.

ABACUS.2 takes the first clustering (projection) created by the clustering algorithm and attempts to fit one equation for each group (subset) of events. If the user includes in the data a list of equation forms, ABACUS.2 uses the least–squares method on each form with the current group. It then calculates the *constancy*, the percentage of events in which the equation evaluates to a constant. If this value exceeds the user–defined amount *CONSTANCY*, then the equation is "linked" to the group and the next group is tried in the same fashion. If none of equations result in a constancy which exceeds this amount, then ABACUS.2 stores the equation with the highest constancy and uses the heuristic search algorithm on the current group. If this method generates an equation whose constancy is greater than *CONSTANCY*, then the next group is processed using the procedure above. Otherwise, the equation with the highest constancy is outputted.

Note that since ABACUS.2 assumes that one equation is supposed to fit each subset, recognising the goal is trivial. This assumption allows the use of curve fitting techniques – something which ABACUS was not able to allow. Both the curve fitting and the search method are discussed in more detail in following sections.

8.1. Domain Independent Constraints

ABACUS.2 employs 3 domain independent constraints to increase the efficiency of the search (the same three used by ABACUS): units compatibility, formula redundancy, and numerical tautologies. These provide simple constraints which drastically reduce the search

space yet are implemented in such a way that they cost very little in terms of time.

6.1.1. Units Compatibility

This constraint requires that two variables' units be "compatible" before they are used in the creation of a new variable. For example, x seconds can not be subtracted from y meters. In general, when two variables are involved in addition or subtraction, their units must be equal. ABACUS.2 can sometimes coerce the units of two variables to be equal. For example, one of its heuristics generates the term $x^m + x^n$. Given two variables x^i and x^j, $i \neq j$, if there exists values for m and n such that $im = jn$, then the term $x^{im} + x^{jn}$ can be asserted (see the section of "Search" for more details on the heuristics).

6.1.2. Formula Redundancy

Another constraint is that a term must be unique before it can be generated. There are situations where a newly generated term can be syntactically unique but semantically identical to another term. For example, $\frac{x}{yz}$ is semantically identical to $\left(\frac{x}{y}\right)\left(\frac{x}{xz}\right)$. To avoid this problem, a canonical representation is used such that semantically identical terms are also syntactically equal. Since ABACUS.2 stores each term in a hash table, checking for redundancy is a one-step operation involving an access operation. The canonical form is given by the grammar in figure 8 and a term represented by the grammar is called a *form*. Generic forms are used to represent user-defined variables. Regular forms are used to represent multiplicative or additive relations. If the type of a regular form is "/", then all of the variables in each side are multiplied together. If the type is "-", then each side is added together. Trig forms represent trigonometric forms and only 1 side is used. Here are a few examples illustrating the interpretation of forms:

FORM ::= REGULAR-FORM | GENERIC-FORM | TRIG-FORM
GENERIC-FORM ::= (NIL (ID) NIL)
REGULAR-FORM ::= (REGULAR-TYPE LEFT-FORM-SIDE RIGHT-FORM-SIDE)
TRIG-FORM ::= (TRIG-TYPE LEFT-FORM-SIDE NIL)
REGULAR-TYPE ::= "/" | "-"
TRIG-TYPE ::= "SIN" | "COS"
LEFT-FORM-SIDE ::= (FORM-LIST) | NIL
RIGHT-FORM-SIDE ::= (FORM-LIST) | NIL
FORM-LIST ::= ID FORM-LIST | ID
ID ::= variable name | form name

Figure 8. Form Grammar.

$$(/\ (x\ y\ z)\ (a\ b)) \leftrightarrow \frac{xyz}{ab}$$

$$(-\ (x\ y\ z)\ (a\ b)) \leftrightarrow (x+y+z)-(a+b)$$

$$(\sin\ (\mathrm{NIL}\ (x)\ \mathrm{NIL})\ \mathrm{NIL}) \leftrightarrow \sin(x)$$

$$(\sin\ (-\ (xy)\ \mathrm{NIL})\ \mathrm{NIL}) \leftrightarrow \sin(xy)$$

To insure the detection of redundancy, ABACUS.2 requires all forms to be in the sums-of-products format.

6.1.3. Numerical Cancellations

When combining terms to create new terms, there is the possibility of creating a new term which either is a tautology or contains a partial cancellation. For example, multiplying $\frac{xy}{z}$ by $\frac{z}{xy}$ results in $\frac{xyz}{xyz}$ which is, of course, equal to 1. Such a result does not contribute much to the discovery of scientific laws. Also, multiplying z by $\frac{z}{yz}$ results in a partial cancellation and causes inefficiencies in the search.

form1 * form2:

1.) TYPE(form1) ∈ {/, NIL} & TYPE(form2) ∈ {/, NIL} &
[LEFT(form1) ∩ RIGHT(form2) ∨ RIGHT(form1) ∩ LEFT(form2)]

2.) [TYPE(form1) = "-" ∨ TYPE(form2) = "-"] & TYPE(form1) ≠ TYPE(form2) &
[(∀ item ∈ sum-form (LEFT(product-form) ∩ RIGHT(iterm))) ∨
(∀ item ∈ sum-form (LEFT(item) ∩ RIGHT(product-form) ∨
item ∈ RIGHT(product-form)))]

form1 /form2:

1.) TYPE(form1) ∈ {/, NIL} & TYPE(form2) ∈ {/, NIL} &
[LEFT(form1) ∩ LEFT(form2) ∨ RIGHT(form1) ∩ RIGHT(form2)]

2.) [TYPE(form1) = "-" ∨ TYPE(form2) = "-"] & TYPE(form1) ≠ TYPE(form2) &
[(∀ item ∈ sum-form (RIGHT(product-form) ∩ RIGHT(iterm))) ∨
(∀ item ∈ sum-form (LEFT(item) ∩ LEFT(product-form) ∨
item ∈ LEFT(product-form)))]

form1 + form2:

1.) TYPE(form1) ∈ {-, NIL} & TYPE(form2) ∈ {-, NIL} &
[LEFT(form1) ∩ RIGHT(form2) ∨ RIGHT(form1) ∩ LEFT(form2)]

2.) [TYPE(form1) = "/" ∨ TYPE(form2) = "/"] & TYPE(form1) ≠ TYPE(form2) &
product-form ∈ RIGHT(sum-form)

form1 - form2:

1.) TYPE(form1) ∈ {-, NIL} & TYPE(form2) ∈ {-, NIL} &
[LEFT(form1) ∩ LEFT(form2) ∨ RIGHT(form1) ∩ RIGHT(form2)]

2.) [TYPE(form1) = "/" ∨ TYPE(form2) = "/"] & TYPE(form1) ≠ TYPE(form2) &
product-form ∈ LEFT(sum-form)

where sum-form = form whose TYPE = "-" and product-form = form whose TYPE = "/"

Figure 9. Tests for Cancellations.

ABACUS.2, like its predecessor, does not allow the creation of terms which result in one of more cancellations. Since all forms are represented in sums-of-products format using the form grammar described previously, checking for such situations involve a simple logical check,

depending on the type of operation performed. Figure 9 shows the tests. If any are true, then the operation is not allowed. Note that trig forms do not require special checks for cancellation.

6.2. Heuristic Search Method

This method is basically a search for constancy. The algorithm simply creates new terms based on a set of heuristics until the resulting constancy exceeds *CONSTANCY*. There are actually two different types of search algorithms, the *dependency graph search*[4] and the *suspension search*. These are described in more detail later.

6.2.1. Heuristics

ABACUS.2 uses the following 4 heuristics. The first two are the same ones used by ABACUS and the last two form the heart of the creation of non-linear terms during the search process:

If the global dependency is $amon^{+}(x,y)$, then
create a variable consisting of the quotient of x and y (6.1)
create variables consisting of the difference of x and y

If the global dependency is $amon^{-}(x,y)$, then
create a variable consisting of the product of x and y (6.2)
create variables consisting of the sum of x and y

If $\#AMON_{local}(x,y) = 2$, then
if $amon^{+}(x^2,y)$ or $amon^{-}(x^2,y)$, create x^2 (6.3)
if $amon^{+}(y^2,x)$ or $amon^{-}(y^2,x)$, create y^2

If $\#AMON_{local}(x,y) \geq 3$ then
if $amon^{+}(\cos(x),y)$ or $amon^{-}(\cos(x),y)$, create $\cos(x)$ (6.4)
if $amon^{+}(\cos(y),x)$ or $amon^{-}(\cos(y),x)$, create $\cos(y)$
if $amon^{+}(\sin(x),y)$ or $amon^{-}(\sin(x),y)$, create $\sin(x)$
if $amon^{+}(\sin(y),x)$ or $amon^{-}(\sin(y),x)$, create $\sin(y)$

---

[4]This algorithm is also called *proportionality graph search* in [Falkenhainer, 1985]. The reason for the change is that Falkenhainer used the term *proportionality* instead of dependency. We feel that dependency is a more appropriate term. Similarly, we use the term dependency graph instead of proportionality graph.

The first two rules create terms such as $x + y$, $x^2 + y^2$, and $x^n - y^m$, for $n,m<4$. In rules (6.3) and (6.4), $\#AMON_{local}(x,y)$ is the number of different local dependencies (after merging) for the (x,y) projection. This is a measure of how many times the local dependency changed. The interpretation of (6.3) is that if the number of different local dependencies is 2, then the terms x^2 and y^2 might be important terms in the final equation However, ABACUS.2 makes one more test before asserting these terms. It checks to see if it can establish a positive dependency between either x^2 and y or y^2 and x. If so, it asserts the corresponding term. A similar interpretation exists for (6.4).

To get a better idea of how these rules work, consider the graph in figure 10. ABACUS.2 has found that the dependency changes 3 times. According to (6.4), it then checks to see if there

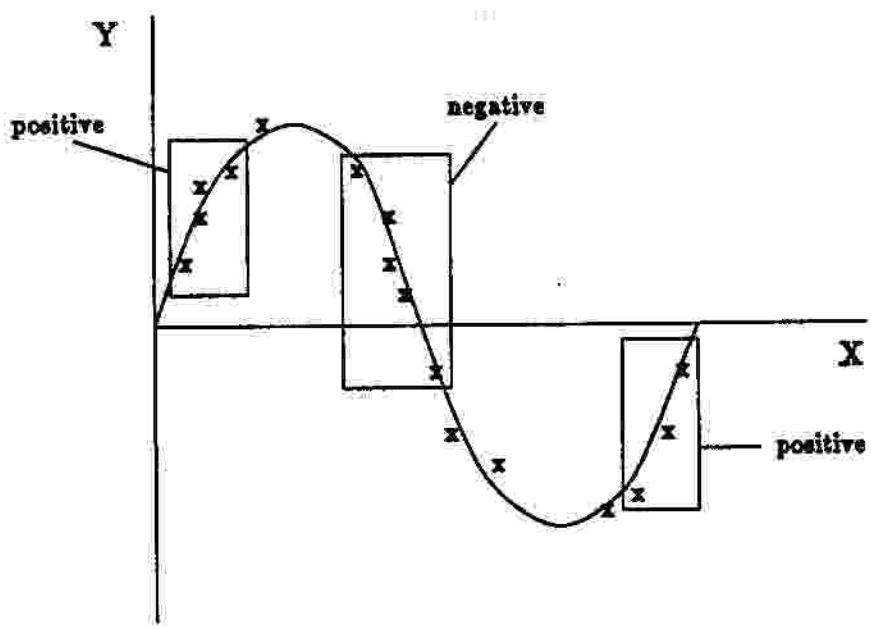

Figure 10. Checking for the Term sin(x).

is a positive dependency between y and $\sin(x)$, which there is. This allows it to assert $\sin(x)$.

The last two heuristics take the two-dimensional analysis of ABACUS and BACON one step further: instead of looking only at the global dependency which provides a very coarse view of the real behavior between two variables, look at the changes of the local dependencies which provides a more fine measurement of this behavior. An analogy can be made between the use of local dependencies and the graphical analysis used by mathematicians to assist in the discovery of equations. In graphical analysis, a graph is created (usually two-dimensional) which allows one to note the potential existence of certain terms. Rules (6.3) and (6.4) are approximations to this method. Chapter 9 discusses possible extensions to this method.

6.2.2. Dependency Graph Search

The first search method that ABACUS.2 uses is the dependency graph search. This method is useful when the desired equation is composed solely of multiplication and division. As the name implies, it uses a graph called a dependency graph to direct the search. A dependency graph G is an ordered pair (V,E), where V is the set of variables and E is the set of edges. The pair (x,y) is in E if and only if the global dependency is $amon^{+}(x,y)$ or $amon^{-}(x,y)$. The main purpose of creating such a graph is to find a list of maximal cycles. A maximal cycle is a cycle which is not a subset of some other cycle. Each cycle represents a set of variables which are probably strongly interrelated. Thus, irrelevant variables tend to be excluded from the cycles.

For example, consider the equation $\frac{xys}{ab} = constant$ Holding any two of the variables constant and varying the third causes the fourth variable to change in a systematic way. This is true regardless of which variables are held constant. One of the cycles in the graph would be $\{x,y,s,a,b\}$. If there were irrelevant variables, they would form other cycles as well. Consider one possible dependency graph when the irrelevant variables c and d are included, shown in

figure 11. The set of maximal cycles sorted in decreasing order of size is $\{ \{x,y,z,a,b\}, \{c,b\}, \{y,d\} \}$. This example shows how irrelevant variables tend to not be included in the large cycles.

The dependency graph search uses the graph to extract the maximal cycles. Once these cycles are obtained, it sorts them in decreasing order of size since the larger cycles are the most promising. For each cycle, ABACUS.2 performs a depth-first search with backtracking. If no equation can be found after trying all of the cycles, then a new dependency graph is constructed where V consists of all the current variables, including user-defined variables and those generated by the program, and the new maximal cycles are extracted again and the process repeats (up to a maximum of four times). Since this search is depth first, in the example above, one possible order of creation is xy, $\frac{xy}{a}$, $\frac{xyz}{z}$, and $\frac{xyz}{ab}$. Note that the maximum depth of the search is the size of the cycle.

If, when ABACUS.2 is building the dependency graph, it finds that one of the rules (6.3) or (6.4) are applicable, it will add any new terms created by these rules to the graph. For example,

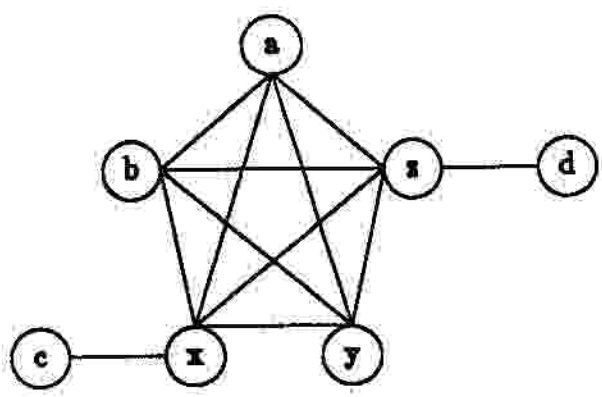

Figure 11. Example Dependency Graph.

when it was given the data corresponding to the equation $f = \frac{mv^2}{r}\cos(\theta)$, it found that rule (6.4) generated the term $\cos(\theta)$ and (6.3) created the term v^2. It then added $\cos(\theta)$ and v^2 to the graph. After building the graph, the equation was formed by first generating mv^2, then $\frac{mv^2}{r}$, $\frac{mv^2}{r}\cos(\theta)$, and finally $\frac{mv^2}{rf}\cos(\theta)$.

As noted above, the dependency graph search takes advantage of the assumption that the desired equation is composed only of multiplication and division since these types of equations naturally fit into the concept of the dependency graph. However, if this assumption does not hold, then this search is not well-suited to discovering equations. The main problem stems from the fact that as new graphs are formed from the current variables, the resulting cycles tend to be large since there are many terms which differ only by one or two variables. This causes increasingly larger interactions and thus larger cycles (that is why there is a limit of four graphs).

6.2.3. Suspension Search

Due to the problems described above, a new search is used if the dependency graph search fails to generate the desired equation. This search is a modification to the suspension search algorithm used in ABACUS. The suspension search algorithm is basically a beam search with backtracking. At each level of the search, the nodes are divided into two groups, one consisting of *active nodes* and one of *suspended nodes*. The active nodes are those whose constancy exceeds a threshold. When entering the next level, the suspension search includes only those nodes which are active (which includes the active nodes from all the previous levels). The suspended nodes are stored in an *environment* for backtracking later, if necessary. If no equation has been found in which the constancy exceeds *CONSTANCY* by the time the depth limit has been reached, then the search backtracks to the previous level and includes the suspended nodes for that level, along

with all the current nodes, in the search. If the search fails again, then it backtracks again to the next previous level, and so on. A filter is used which prevents the use of suspended nodes after a user–defined level. The filter depth must be less than the depth limit.

ABACUS.2 uses a modified suspension search algorithm. If a non–linear term is created on an arbitrary level, it calls the dependency graph search algorithm on the new non–linear term along with the original user–defined variables. Any new nodes are added into the current level's nodes and the suspension search continues. The reason for this modification is that non–linearity can make it hard to find trends between some of the user–defined variables. So, whenever a new non–linear term is created, it helps to check the trends between the user–defined variables and the new non–linear variables. This was the case when running the data for the "instantaneous current" example (see chapter 8). Figure 12 shows the basic algorithm for the suspension search.

6.3. Curve Fitting

ABACUS.2 uses the method of least–squares whenever the user inputs a set of equation forms. It tries each form on the current set of events until one is found which meets the criteria. The specific algorithm is based on Sedgewick's [Sedgewick, 1983] algorithm. In this method, given an equation form $f(x)=c_1f_1(x)+c_2f_2(x)+\cdots+c_mf_m(x)$, the goal is to find the "best" choice for the coefficients $c_1,\ldots,c_m$ by using the least–squares criterion. Let $E=\sum_{j=1}^{N}(f(x_j)-y_j)^2$, where N is the total number of events and y_j is the observable value. E is the sum of the squared errors and the goal is to choose the parameter values that minimize E. In the case where $M=2$ and $N=3$,

$$\begin{aligned}E=&(c_1f_1(x_1)+c_2f_2(x_1)-y_1)^2\\&+(c_1f_1(x_2)+c_2f_2(x_2)-y_2)^2\\&+(c_1f_1(x_3)+c_2f_2(x_3)-y_3)^2\end{aligned}$$

- If the search limit has been reached, then if the best constancy exceeds the user-defined parameter CONSTANCY, return true else return false.

- Generate new active or suspended nodes. If one of the new nodes is a non-linear term, then call the dependency graph search and combine the nodes generated from it with the new active and suspended nodes. If the constancy of the new nodes exceeds CONSTANCY, return true else return true if a call to the suspension search returns true.

- If the filter limit has been reached, then save the environment and return false.

- Generate new active or suspended nodes from the current list of suspended nodes. If one of the new nodes is a non-linear term, then call the dependency graph search and combine the nodes generated from it with the new active and suspended nodes. If the constancy of the new nodes exceeds CONSTANCY, return true else return true if a call to the suspension search returns true.

- return false.

Figure 12. Suspension Search Algorithm.

In the general case, the first step is to calculate the function component vectors

$$F_1=(f_1(x_1),f_1(x_2),\ldots,f_1(x_N)),$$
$$F_2=(f_2(x_1),f_2(x_2),\ldots,f_2(x_N)),$$
$$\vdots$$
$$F_M=(f_M(x_1),f_M(x_2),\ldots,f_M(x_N)).$$

Then Gaussian elimination is used on an M-by-M linear system of equations $Ac=b$, with $a_{ij}=F_i \cdot F_j$ and $b_j=F_j \cdot Y$, $Y=(y_1,y_2,\ldots,y_N)$. Refer to [Sedgewick, 1983] for more details.

The general procedure for processing equation forms is to try each form on the current subset until one is found in which the constancy exceeds the threshold. If one such form exists, then the next subset is tried in a similar fashion. If, on the other hand, no "acceptable" equation exists, then the equation with the highest constancy is saved and the heuristic search algorithm is

called. If it produces an equation whose constancy exceeds the threshold, then the search for the current subset is finished. Otherwise, the equation with the highest constancy is returned. This process allows a smooth integration of standard linear regression techniques and the heuristic search method.

7. DERIVING THE PRECONDITIONS

As mentioned earlier, ABACUS.2 uses A^t for two purposes. One is to help build the clusters and the other is to generate a set of preconditions, one for each equation, when multiple equations exist. These preconditions specify the applicability of the equations. The objective for both cases is to generate a set of class descriptions called *covers*. The problem of generating a set of covers is called the *set covering problem* [Michalski, 1975]. The general problem can be stated as:

- Given a set of events which have been divided into classes,
- Find a description for each class such that the description includes the events for the corresponding class and distinguishes it from all other classes.

The set of such descriptions is called a *discriminating description* since each cover discriminates its class from all other classes.

7.1. A^t Algorithm

There are two main algorithms in A^t. The first one, called *Cover*, generates a cover given a set of positive instances and negative instances. The second, called *Star*, generates a list of maximally general *complexes* which do not cover any negative events. A complex is a conjunction of attribute values such as *color=red & size=big & shape=round*. The function Cover works by selecting an event called the *seed* and calling Star with the seed and the negative events to generate the list of complexes. Then it picks the complex from this list which has the highest preference, according to the preference criterion LEF. It removes from consideration all positive events covered by this complex and repeats the process again until there are no more positive events remaining. The result of Cover is a disjunction of complexes which cover all the positive events and none of the negative events.

The Star algorithm is the heart of A^t. Given a seed event and the negative events, it generates a star $G(e|NEG,m)$. e is the seed, NEG is the set of negative events, and m is an upper bound on the number of complexes allowed during the search. To generate the star, it performs a beam search through the space of alternative generalisations. A generalisation operator, called ExtendAgainst, is used to help create the complexes. ExtendAgainst(p,n) works by comparing an attribute value p from a positive event with the corresponding attribute value n from the negative event and generalising p as much as possible without covering n. The actual generalisation depends on the type of attribute. For nominal attributes, ExtendAgainst(p,n) extends p by adding all values except those in n. For linear and structured attributes, the process is more complicated but it is based on the same idea as that used for nominals (see [Becker, 1985] for more details).

Using the ExtendAgainst operator, Star first randomly selects a negative instance and extends the seed against the negative instance. This returns a list of candidate complexes. This list is appended to (multiplied) the complexes in the current star to prevent any of them from covering the current negative event. The ExtendAgainst operator is called for each negative event. During this phase, if $G(e|NEG,m)$ ever exceeds m, then the complexes are ordered (using LEF) and the best m are kept. When all of the negative events have been processed, $G(e|NEG,m)$ is returned.

7.2. Using A^t to Build Preconditions

ABACUS.2 uses A^t to generate preconditions by creating a set of classes where each class consists of the events for a particular equation. For example, one of the experiments used in ABACUS.2 involved Stoke's law which describes the velocity of objects falling through different media (see chapter 8). Three equations were generated: $v=9.8t$, $rv=.9m$, and $rv=.7m$. The events for each equation were placed into individual classes and passed to A^t. The resulting

covers or preconditions were:

- If [substance = vacuum] then v=0.8t
- If [substance = glycerol] then v=.9m
- If [substance = casteroil] then v=.7m

Although this is a simple example, it does illustrate how symbolic information can be integrated into the discovery process. In addition to symbolic information, A^t can also include numerical information in the covers.

8. EXPERIMENTS

One important aspect with any work in machine learning is the experimental results. This chapter describes the experiments used and relates them to the various abilities of ABACUS.2. The experiments performed are:

- Centrifugal force resulting from curves
- Snell's law of reflection and refraction
- Instantaneous current
- Angular conservation of energy
- Stoke's law of falling bodies

8.1. Centrifugal Force

When a vehicle encounters a curve in the road, it experiences centrifugal force which tends to push the vehicle away from the direction of the curve. Friction between the tires and the road is generally sufficient to keep the vehicle on the road but as the velocity increases, it becomes necessary to bank the curve a certain number of degrees. In designing highways, several equations come into consideration. One of these, $f = \frac{mv^2}{r}\cos(\theta)$, describes the component of centrifugal force which is parallel to the road. One reason for banking the road is to control this parameter. θ is the degree of banking, m is the mass, v is the velocity, and r is the radius of the curve (this equation makes the assumption that the curve is a circle). Figure 13 shows the various components.

ABACUS.2 was able to discover this equation by using the dependency graph only. As it built the graph, it found that 2 non-linear terms (v^2 and $\cos(\theta)$) existed and added them to the graph as well.

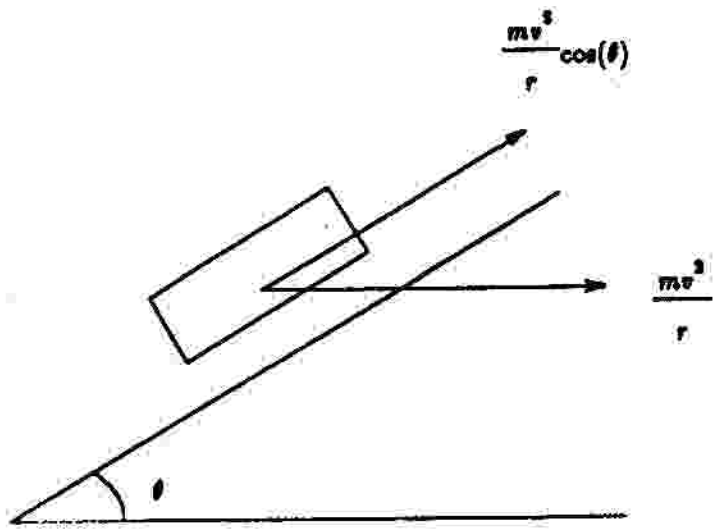

Figure 13. Centrifugal Force Parallel to the Road.

8.2. Snell's Law of Reflection and Refraction

Light waves have the ability to travel through certain materials. When it is traveling through free space, it travels at the speed of $2.998x10^8 m/s$. Whenever light travels in a transparent medium such as air or glass, the velocity is always lower. The *refractive index* is used to characterize this difference. The refractive index n is defined as the velocity of light in free space over the velocity of light in a medium. Values of n include 1.0003 for air at STP and 2.42 for diamond[5]. This index plays an important role in characterizing light rays travelling from one medium into another. Consider figure 14. Here, the light is travelling from medium #1 to medium #2. The incident ray hits medium #2 at an angle θ_1. It is broken into 2 parts, the reflected ray and the refracted ray. The refracted ray enters the medium #2 at the angle θ_2.

[5]according to [O'Dwyer, 1981]

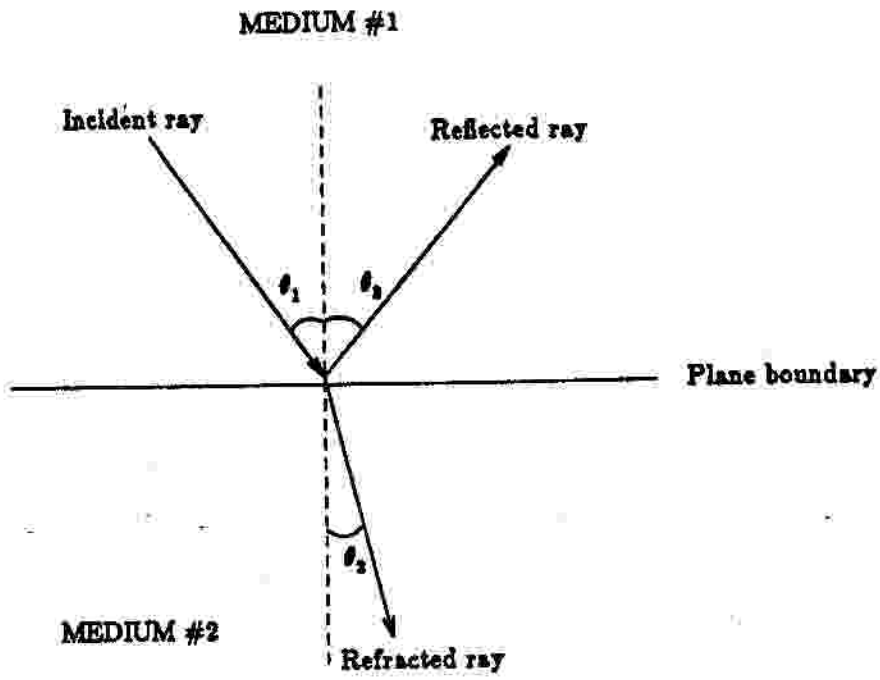

Figure 14. Reflection and Refraction of a Light Ray.

The law of refraction states that $\theta_1=\theta_3$. Medium #1 has a refractive index n_1 and medium #2 has the corresponding index n_2. Snell's law states that $n_1\sin\theta_1=n_2\sin\theta_2$. Thus, given the refractive indexes of the 2 medians along with the angle of the incident ray, the angle of the refracted ray can be found.

This experiment also demonstrates the power of dependencies in discovering equations containing non-linear terms. This equation is harder for ABACUS.2 to discover than the centrifugal force equation since there are two sin terms. It had to first discover these terms before it could find trends between the other terms.

8.3. Instantaneous Current

Ohm's law states that $I=\frac{V}{R}$. However, this is based on *average* values of I, V, and R. To test ABACUS.2's ability to discover non-linear terms consisting of a *combination* of user-defined variables, we gave it data corresponding to the instantaneous values of current and voltage. These are related by the equation $i=\frac{V_o\cos(\omega t)}{R}$, where i is the instantaneous current, V_o is the peak voltage value, ω is the angular frequency, and t is the time.

Although this relation is not used very often, it does help show another aspect of ABACUS.2. In this case, the term ωt had to be generated before the non-linear term cos could be detected. In the actual test run, $\cos(\omega t)$ was not discovered until the second level of the suspension search algorithm. Once it was found, ABACUS.2 called the dependency graph search on the original user-defined variables and $\cos(\omega t)$. It was then able to discover the desired equation. This experiment demonstrates the need for the modification to the original suspension search.

8.4. Angular Conservation of Energy

The angular momentum conservation principle states that the angular momentum of a system of bodies is conserved provided that the external net torque on the system is zero. An application of this principle can be shown by imagining a child of mass m running at v_o m/s in a direction which is tangent to the edge of a merry-go-round at rest which has a radius r. The conservation principle allows us to calculate the amount of energy dissipated when the child "collides" with the merry-go-round. This energy, denoted E_{diss}, is equal to the initial kinetic energy of the child and the merry-go-round minus the final kinetic energy: $E_{diss}=\frac{1}{2}mv_o^2-\frac{1}{2}mv^2-\frac{1}{2}I\omega^2$.

The data for this equation was given to ABACUS.2 along with several equation forms. It used the least-squares method to find the coefficients for each form. The best one had a constancy of 100% and the resulting equation was $.489mv_e^2=.501mv^2+.492I\omega^2+E_{diss}$. If ABACUS.2 failed to find the equation using linear regression, it would have then tried to find one (or more) equations using the heuristic search.

8.5. Stoke's Law of Falling Bodies

In this experiment, data from Stoke's law was given to ABACUS.2. This is the same data used by ABACUS and the purpose of this experiment was to demonstrate the clustering algorithm. Stoke's law describes the velocity of falling objects when travelling through different mediums such as a vacuum (see [Falkenhainer, 1985]). This experiment was run twice, once without any information regarding the general form of the equations, and once with the form for one of the equations. The main difference in the way ABACUS.2 discovered the equations in the first experiment and the way ABACUS did lies in the way each handles multiple equations. ABACUS.2 divided the events up before calling the equation discovery algorithm which is in contrast to ABACUS's approach which searches for the subsets during the search. The resulting equations and preconditions are the same as those generate by ABACUS:

- If [substance = vacuum] then $v=9.8t$
- If [substance = glycerol] then $rv=.9m$
- If [substance = casteroil] then $rv=.7m$

In the second run, ABACUS.2 was given the general equation form for the equation $v=9.8t$. This allowed it to discover it using linear regression. However, since the resulting equation fit only one of the subsets, it had to use the heuristic search method to find the other equation. This demonstrates the ability of ABACUS.2 to combine both methods whenever necessary.

9. CONCLUSIONS AND FUTURE WORK

A methodology for quantitative discovery has been presented. ABACUS.2 is an extension to the algorithm ABACUS which allows the discovery of equations which include non-linear terms. It presents a method which allows future implementations to easily extend the types of equations which can be discovered. It also uses a clustering algorithm to separate the events into groups based on the dependencies and the symbolic data such that one equation fits each subset. This allows the addition of curve fitting techniques to extend the power of the equation discovery algorithm even more.

9.1. Limitations

ABACUS.2 is still limited in some respects. First, the only non-linear terms it currently can discover are sin, cos, and x^2. Future implementations should develop more heuristics based on the local dependencies which would allow more non-linear terms to be included in the search space. Another problem is that it cannot discover multiple equations when non-linear terms exist. This is a fundamentally hard problem and the next section discusses a possible solution. Additionally, it does not allow the user to supply some additional information which might be helpful such as a list of variables which are "probably important" and "probably unimportant". Knowledge about the dependent and independent variables could also be helpful.

9.2. Future Work

One of the main contributions of this research is the use of local dependencies to allow a more "fine" measure of the behavior of two variables with respect to each other. This knowledge might be able to be used to help discover other non-linear terms such as log, ln, tan, etc. Additionally, this method can be taken one step further by measuring the *rate of change* of the dependencies. Such information might allow might allow the discovery of terms such as

$sin(kx+j)$.

Another area of future research is the integration of knowledge-intensive approaches such as that used in the HOTEP system. ABACUS.2 requires very little background knowledge to generate equations but the cost is an increased search space. If a particular domain is well-known, then it would benefit quantitative discovery to make use of this knowledge. Additionally, ABACUS.2 does not make full use of the information from the units for the variables. Dimensional Analysis can be integrated into this methodology as well. Kokar's work [Kokar, 1985] might be helpful for this.

The search algorithms used by ABACUS.2 are based on empirical observations and are designed to be as efficient as possible. However, there probably is room for improvement, particularly when non-linear terms are present. Future research could focus on integrating the current search algorithms with some possibly more efficient ones (or replacing them altogether).

The problem of clustering when non-linearity is present might be partially solved by looking at the symbolic data. When detecting changes in the dependencies, it might be useful to check to see if a nominal subgroup exists for the events corresponding to the "simplest" non-linear term and if one does, then remove the events which correspond to the non-linear term. For example, if the local dependency changed more than two times, check to see if the events corresponding to the first two changes form a nominal subgroup. If so, and if x^2 evaluates to a constant for these events, then assert x^2 as one of the final equations, remove the events which it fits, and continue the search. Using derivatives of the dependencies may also help with this problem.

APPENDIX I – USER'S GUIDE

ABACUS.2 is written in Common Lisp and has been tested on a Symbolics and Sun–3. It consists of the following files:

abacus.lisp – main ABACUS.2 routine
aq.lisp – A^q algorithm which uses the VL_1 representation
aq2.lisp – second A^q algorithm which uses a bit–vector representation
aq2–iface.lisp – interface for aq2.lisp
cluster.lisp – clustering algorithm
form.lisp – form handling routines
globals.lisp – global variables and structures
graph.lisp – dependency graph handling routines
hash.lisp – hash functions
io.lisp – input and output routines
least–squares.lisp – curve fitting routines
search.lisp – suspension search routines
minor.lisp – minor functions used by several routines
system.lisp – system information for Symbolics

A.1 Data Set

Input to ABACUS.2 consists of a single list called a data set which has 5 main sections. The data set must be in a form readable by LISP and it has the following format:

```
((DECLARATIONS
      <variable declarations>)

 (PARAMETERS
      <user–definable parameter settings, rewrite and augmentation rules>)

 (EQUATIONS
      <list of equation forms used by curve fitting algorithm>)

 (VARIABLES
      <ordered list of all user–defined variables>)

 (EVENTS
      <list of event tuples>)
)
```

The "PARAMETERS" and "EQUATIONS" section is optional but all other are required. If the former is left out, then the default values will be used. Figure A1 gives an example data set.

```
(
(declarations
  (f L ((meters 2) (seconds -2) (radians 1)))
  (r L ((meters 1)))
  (m L ((meters 1)))
  (v L ((meters 1) (seconds -1)))
  (theta L ((radians 1)))
  )

(parameters
  (*constancy* .87)
  (*uncertainty* .01)
  )

(variables
  (f r m v theta)
  )

(events
  (3221.6787    30    50     75    1.22)
  (8121.7944    30    50     75    0.523)
  (-4909.809    30    50     75    2.122)
  (-2885.0676   20    46     52    4.23)
  (1883.0659    20    46     52    5.02)
  (-4994.303    20    46     52    3.78)
  (5521.2207    23    61     50    6.87)
  (4954.888     23    61     50    7.01)
  (6549.0776    23    61     50    6.44)
  (-3080.7163   30    50     60    2.11)
  (-1848.4298   50    50     60    2.11)
  (-6161.4326   15    50     60    2.11)
  (3699.0632    20    40     44    .3)
  (4623.829     20    50     44    .3)
  (-691.4844    27    48     75    1.64)
  (340          30    2.197  75    .6)
  (340          30    6.103  45    .6)
                .
                .
                .
  )
)
```

Figure A1. Example Data Set.

A.1.1 Declarations

Each declaration is a list consisting of the variable name, its *type*, and the units. The type is either "L" or "N" indicating linear or nominal variable. A linear variable is one whose values are totally ordered and a nominal variable is one whose values consist of independent symbols or names (no structure is assumed). Each unit consists of a list of *universal units* along with the corresponding exponent. Universal units are units which are considered the base units and thus are not composed of other units. An example unit declaration is (A L ((METERS 1) (SECONDS -2))) which describes the units for accleration. The variable name is "A", the "L" indicates that it is a linear variable, the the dimensions are meters per seconds squared. The general form for a declaration is given below:

(DECLARATIONS
  ($variable\text{-}name_1$ $type_1$ (($unit\text{-}name_1$ exp_1)($unit\text{-}name_2$ exp_2)...))
  ($variable\text{-}name_2$ $type_2$ (($unit\text{-}name_1$ exp_1)($unit\text{-}name_2$ exp_2)...))
  .
  .
)

A.1.2 Parameters

There are many user-definable parameters which allow for greater flexibility. In addition to these parameters, the user can input *rewrite rules* and *augmentation rules*. The general form for the parameters is

(PARAMETERS
  ($parameter_1$ $value_1$)
  .
  .
  ($rewrite\text{-}rule_1$)
  .
  .
  ($augment\text{-}rule_1$)
  .
  .
)

The sequence for the parameters, rewrite rules, and augmentation rules is unimportant.

Rewrite rules are used to change the values of a current variable, for example, if we wanted to change x to $sin(x)$. In this case, the values of x would be replaced with $sin(x)$. The general form for a rewrite rule is (<=> variable-name formula units). The variable name must be one of those declared in the "VARIABLES" section of the data set. The formula consists of any Common Lisp arithmetic function. In the example above, the rule would be (<=> x sin(x) nil).

Augmentation rules are used to add new variables to the data set which can be defined in terms of the declared variables. The form for these rules is (<= new-variable-name formula units). As with rewrite rules, the formula must be a Common Lisp arithmetic form. For example, (<= v (/ m s) ((meters 1) (seconds -1))) can be used to create a new variable v for velocity based on the variables m and s.

The following are the user-definable parameter descriptions. The default values are shown in the parentheses immediately following each parameter name.

starLEF ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))
This provides the lexographic evaluation functions used by A^t to evaluate the current hypotheses and constrain the search space. Each function is associated with a *tolerance* which indicates the percentage of the complexes to retain based on the evaluation of the functions. The tolerance must be in [0..1]. The predefined functions for LEF are:

Max-Promise – select best ratio of % positive events covered to % negative events covered
Min-Covered – select those complexes which cover the fewest events
Max-Covered – select those complexes which cover the most events
Min-Selectors – select those complexes which have the fewest selectors
Max-Selectors – select those complexes which have the most selectors
Min-Weight – select those complexes of minimum total selector weight
Max-Weight – select those complexes of maximum total selector weight

maxstar (10)
This indicates the maximum number of complexes allowed in the star for A^t. This is used to implement the beam search by restricting the number of complexes at each level of the search. LEF is used to evaluate each complex. The best m complexes, where m is from the star $G(e \mid NEG, m)$, are retained.

covermode (ic)
Covers generated by A^t can be disjoint (dc) or intersect in certain areas (ic).

univignore (nil)
This indicates a list of variables which should not be required to be held constant when the dependency is being calculated. Sometimes holding certain variables constant prevents ABACUS.2 from finding the dependency. The value for this parameter consists of a list of numbers which indicate the column for the corresponding variable. The first column starts at "0".

uncertainty (.02)
This indicates the uncertainty. It is used to account for slight errors in the data (noise) and round–off error. Any number falling within the range of uncertainty is regarded as constant. When ABACUS.2 calculates the constancy, any event which evaluates to a constant within this range is considered to be part of the events in which the equation fits.

constancy (.95)
Specifies the number of total events which must evaluate to a constant for an equation to considered acceptable. A value of .95 indicates that 95% of the events must evaluate to a constant ($\pm$ *uncertainty*).

max–nodes (1000)
This specifies the maximum number of nodes allowed for the search tree. Exceeding this value causes the algorithm to stop and output the best equation up to that point.

filter–level (2)
This shows the maximum level in the suspension search in which suspended nodes are allowed. Beyond this point, the algorithm considers only the active nodes.

search–limit (3)
Indicates the maximum search depth for the suspension search.

direct–sw (70)
Indicates the percentage of events in which the dependency must be positively monotonic before $amon^+(x,y)$ can be asserted. For example, a value of 100 implies that the values for y must always increase when the values of x increase in order for $amon^+(x,y)$ to be asserted.

inverse–sw (30)
Indicates the percentage of events in which the dependency must be negatively monotonic before $amon^-(x,y)$ can be asserted. A value of 0 means that the values for y must always decrease when the values of x increase in order for $amon^-(x,y)$ to be asserted. Note that $*direct-sw* = 100 - *inverse-sw*$.

print–flags ((p))
This specifies the amount of information printed out after running ABACUS.2. There are two optional tables. The parameters table shows the parameter values which were used. The events table shows the events and the classes each was assigned to. A value of '(p e) specifies that both tables be outputted while NIL indicates that only the resulting equations be printed out.

debug (nil)
Debugging flag which, if T, will provide information during the execution of the program. Currently, only NIL and T are allowed but future revisions should change this to allow for degrees of debugging information.

minimum-link-value (50)
This parameter indicates the minimum value for the links in the group support graph. Any link below this value is ignored. This is used to "weed out" low-valued links which may adversely effect the global projection.

cluster-on (t)
This is a switch which allows the user to turn off the clustering component. If this is NIL, then the events will not be clustered and ABACUS.2 will attempt to fit one equation to the events.

cluster-order (gc nc rc)
This allows the user to indicate the order of the clusters. 'gc indicates the global projection (the projection created by using the greedy algorithm), in a single set, and 'rc stands for all other projections.

A.1.3 Equations

The equations consist of a list of equation forms. ABACUS.2 will use the least-squares method on each form to find the coefficients. If one of the forms exceeds the constancy threshold, then the discovery algorithm stops. The only requirements for the coefficients is that they consist of unique symbols. Each *function*$_i$ must be a LISP form which can be APPLY'd. The list must be of the following form:

(EQUATIONS
$(= dependent\text{-}var_1\ (+\ (*\ coeff_{1_0}\ function_1^{\,0}) \cdots (*\ coeff_1^{\,n-1}\ function_1^{\,n-1})))$
$(= dependent\text{-}var_2\ (+\ (*\ coeff_{2_0}\ function_2^{\,0}) \cdots (*\ coeff_2^{\,n-1}\ function_2^{\,n-1})))$
.
.
.
)

A.1.4 Variables

This section consists of simply a list of the user-defined variables in the order that they are presented in the "EVENTS" section. This informs the program which location in the event tuple corresponds to which variable.

A.1.5 Events

The events consist of a list of n-tuples in the following form:

(EVENTS

$(attribute\text{-}value_1^0\ attribute\text{-}value_1^1 \ldots attribute\text{-}value_1^{n-1})$

$(attribute\text{-}value_2^0\ attribute\text{-}value_2^1 \ldots attribute\text{-}value_2^{n-1})$

.

.

.

)

A.2 Compiling and Running ABACUS.2

To compile the files on the Symbolics, the "make-system" utilities are used. Before they can be used, the location of the sources must be known to the system. To do this, include in the lispm-init.lisp file the line: **(si:set-system-source-file "abacus" full-path-name>SYSTEM)**. For example, if the files are in "MACHINE-A:>smith>ABACUS.2", then the line should be **(si:set-system-source-file "abacus" "MACHINE-A:>smith>ABACUS.2>system")**. Alternatively, this line could just be typed at the prompt level each time you logged onto the machine. Now, to compile the files, type **(make-system 'abacus :compile :noconfirm)**. To compile them on the Sun, simply load "compile-files.lisp".

Loading the files on a Symbolics is done with the command **(make-system 'abacus)**. This assumes that the lispm-init.lisp file has the correcct line as described above. Loading the "load-files.lisp" is all that is required to load the system on a Sun.

To run, type "(run)" and a menu of commands is displayed which, amoung other options, allows the user to specify a data file and run the program.

APPENDIX II - EXPERIMENTAL DATA

B.1 Centrifugal Force

The data for the centrifugal force experiment consisted of 6 variables with one of them (the length) not included in the final equation. The uncertainty was set for 2%.

```
(
(declarations
  (f L ((meters 2) (seconds -2) (radians 1)))
  (r L ((meters 1)))
  (m L ((kilograms 1)))
  (v L ((meters 1) (seconds -1)))
  (theta L ((radians 1)))
  (length L ((meters 1)))
  )

(parameters
  )

(variables
  (f r m v theta length)
  )

(events
  (3221.6787   30   50   75   1.22   3212.3)
  (8121.7944   30   50   75   0.523  3212.3)
  (-4909.809   30   50   75   2.122  3212.3)
  (-3986.427   30   50   75   2.01   3212.3)
  (-2885.0676  20   46   52   4.23   1755.3)
  (1883.0659   20   46   52   5.02   1755.3)
  (1584.4714   20   46   52   4.97   1755.3)
  (-4994.303   20   46   52   3.78   1755.3)
  (5521.2207   23   61   50   6.87   2112.4)
  (6189.342    23   61   50   6.65   2112.4)
  (4954.888    23   61   50   7.01   2112.4)
  (6549.0778   23   61   50   6.44   2112.4)
  (3927.5881   23   61   50   7.22   2112.4)
  (1372.5167   30   50   60   1.34   3546.7)
  (1029.3876   40   50   60   1.34   3546.7)
  (823.5101    50   50   60   1.34   3546.7)
```

(2745.0334 15 50 60 1.34 3546.7)
(2774.2974 20 30 44 .3 4231)
(3699.0632 20 40 44 .3 4231)
(4623.829 20 50 44 .3 4231)
(5548.5947 20 60 44 .3 4231)
(8020.9575 27 48 75 .64 3211.8)
(5133.413 27 48 60 .64 3211.8)
(4005.488 27 48 53 .64 3211.8)
(9588.074 27 48 82 .64 3211.8)
(387.744 10 41 –10 .331 1998.7)
(2051.167 10 41 –23 .331 1998.7)
(5308.219 10 41 –37 .331 1998.7)
(9693.608 10 41 –50 .331 1998.7)
(340 30 2.1970859 75 .6 4538)
(340 30 4.9434433 50 .6 4538)
(340 30 13.731788 30 .6 4538)
(340 30 6.103017 45 .6 4538)
(612.413.1 27.725 –22.0 .93 1967.5)
(612.413.1 7.027 –43.7 .93 1967.5)
(612.413.1 5.001 –51.8 .93 1967.5)
(450 40 7.2548094 50 0.123 1455.6)
(450 50 9.068512 50 0.123 1455.6)
(450 60 10.882214 50 0.123 1455.6)
(450 70 12.695916 50 0.123 1455.6)
(500 493.6402 50 75 .5 2365.4)
(500 219.39563 50 50 .5 2365.4)
(500 140.41321 50 40 .5 2365.4)
(500 78.98243 50 30 .5 2365.4)
(210 7.37 34.5 –9 5.3 3425)
(210 87.52 34.5 –31 5.3 3425)
(210 141.38 34.5 –39.4 5.3 3425)
(300 20 –7.0686507 40 2.13 4368)
(300 20 8.980002 40 1.14 4368)
(300 20 3.7785466 40 0.123 4368)
(300 20 3.9558227 40 0.324 4368)
(540 25 50 16.51858 0.145 2436)
(540 25 50 34.35569 1.34 2436)
(540 25 50 17.540335 0.5 2436)
(540 25 50 18.788666 0.7 2436)
(540 25 50 27.296886 1.2 2436)

```
(440     263.5919    60     75     1.22    3544)
(440     -463.71912  60     75     2.22    3544)
(440     -305.1912   60     75     1.98    3544)
(440     660.64606   60     75     0.533   3544)
 )
)
```

=-=- Parameters -=-=

MAXSTAR = 10
COVERMODE = IC
UNCERTAINTY = 0.02
CONSTANCY = 0.95
INVERSE-SW = 30
DIRECT-SW = 70
UNIVIGNORE = NIL
DEBUG = NIL
PRINT-FLAGS = (P E)
FILTER-LEVEL = 2
SEARCH-LIMIT = 3
MAX-NODES = 1000
CLUSTER-ON = T
CLUSTER-ORDER = (GC NC RC)
MINIMUM-LINK-VALUE = 50.0

starLEF = ((MAX-COVERED 0.5) (MIN-WEIGHT 0.0) (MIN-SELECTORS 0.0))

=-=- Class Events -=-=

CLASSA
(F R M V THETA LENGTH)
(210.0 7.37 34.5 -9.0 5.3 3425.0)
(210.0 87.52 34.5 -31.0 5.3 3425.0)
(210.0 141.38 34.5 -39.4 5.3 3425.0)
(612.4 13.1 7.027 -43.7 0.93 1967.5)

(387.744 10.0 41.0 -10.0 0.331 1998.7)
(2051.167 10.0 41.0 -23.0 0.331 1998.7)
(5308.219 10.0 41.0 -37.0 0.331 1998.7)
(6549.0776 23.0 61.0 50.0 6.44 2112.4)
(3927.5881 23.0 61.0 50.0 7.22 2112.4)
(-4909.809 30.0 50.0 75.0 2.122 3212.3)
(4954.888 23.0 61.0 50.0 7.01 2112.4)
(500.0 219.39563 50.0 50.0 0.5 2365.4)
(540.0 25.0 50.0 16.51858 0.145 2436.0)
(9588.074 27.0 48.0 82.0 0.64 3211.8)
(5548.5947 20.0 60.0 44.0 0.3 4231.0)
(2774.2974 20.0 30.0 44.0 0.3 4231.0)
(340.0 30.0 6.103017 45.0 0.6 4538.0)
(1372.5167 30.0 50.0 60.0 1.34 3546.7)
(2745.0334 15.0 50.0 60.0 1.34 3546.7)
(-4994.303 20.0 46.0 52.0 3.78 1755.3)
(1584.4714 20.0 46.0 52.0 4.97 1755.3)
(9693.608 10.0 41.0 -50.0 0.331 1998.7)
(5521.2207 23.0 61.0 50.0 6.87 2112.4)
(6189.342 23.0 61.0 50.0 6.65 2112.4)
(500.0 493.6402 50.0 75.0 0.5 2365.4)
(500.0 140.41321 50.0 40.0 0.5 2365.4)
(500.0 78.98243 50.0 30.0 0.5 2365.4)
(540.0 25.0 50.0 34.35569 1.34 2436.0)
(8020.9575 27.0 48.0 75.0 0.64 3211.8)
(-3986.427 30.0 50.0 75.0 2.01 3212.3)
(3221.6787 30.0 50.0 75.0 1.22 3212.3)
(8121.7944 30.0 50.0 75.0 0.523 3212.3)
(440.0 -463.71912 60.0 75.0 2.22 3544.0)
(440.0 263.5919 60.0 75.0 1.22 3544.0)
(440.0 660.64606 60.0 75.0 0.533 3544.0)
(823.5101 50.0 50.0 60.0 1.34 3546.7)
(1029.3876 40.0 50.0 60.0 1.34 3546.7)
(4623.829 20.0 50.0 44.0 0.3 4231.0)
(3699.0632 20.0 40.0 44.0 0.3 4231.0)
(300.0 20.0 -7.0686507 40.0 2.13 4368.0)
(300.0 20.0 8.980002 40.0 1.14 4368.0)
(300.0 20.0 3.7785466 40.0 0.123 4368.0)
(340.0 30.0 4.9434433 50.0 0.6 4538.0)
(340.0 30.0 13.731788 30.0 0.6 4538.0)
(2885.0676 20.0 46.0 52.0 4.23 1755.3)
(440.0 -305.1912 60.0 75.0 1.98 3544.0)
(1883.0659 20.0 46.0 52.0 5.02 1755.3)
(540.0 25.0 50.0 27.296886 1.2 2436.0)
(450.0 70.0 12.695916 50.0 0.123 1455.6)
(450.0 60.0 10.882214 50.0 0.123 1455.6)
(450.0 50.0 9.068512 50.0 0.123 1455.6)
(450.0 40.0 7.2548094 50.0 0.123 1455.6)

(5133.413 27.0 48.0 60.0 0.64 3211.8)
(4005.488 27.0 48.0 53.0 0.64 3211.8)
(300.0 20.0 3.9558227 40.0 0.324 4368.0)
(340.0 30.0 2.1970859 75.0 0.6 4538.0)
(540.0 25.0 50.0 18.788666 0.7 2436.0)
(540.0 25.0 50.0 17.540335 0.5 2436.0)
(612.4 13.1 27.725 -22.0 0.93 1967.5)
(612.4 13.1 5.001 -51.8 0.93 1967.5)

=-=-= Hypotheses -=-=

CLASSA
Cover: Relation:
F * R = COS (THETA) * M * V^2

B.2 Snell's Law

Data for Snell's law consisted of 4 variables but the complex relationships made this experiment hard.

```
(
(declarations
  (N1 L ((refractions 1)))
  (N2 L ((refractions 1)))
  (THETA1 L ((radians 1)))
  (THETA2 L ((radians 1)))
)

(parameters
)

(variables
  (N1 N2 THETA2 THETA1)
)

(events
 (1.01          1.01    0.5     0.5)
```

(0.06 1.01 0.03 0.5)
(2.02 1.01 1.3 0.5)
(1.77 1.01 1.0 0.5)
(1.18 1.01 0.6 0.5)
(1.46 1.24 1.7 1.0)
(1.09 1.24 2.3 1.0)
(–1.18 1.24 4.0 1.0)
(0.20 1.24 3.0 1.0)
(–0.51 1.24 3.5 1.0)
(–7.26 1.8 5.0 0.24)
(0.42 1.8 6.34 0.24)
(5.58 1.8 7.113 0.24)
(–5.34 1.8 5.5 0.24)
(3.74 1.8 6.8 0.24)
(2.44 1.51 0.2 0.123)
(0.55 1.51 0.2 0.567)
(0.33 1.51 0.2 1.1)
(0.39 1.51 0.2 0.85)
(.03 1.51 .02 2.01)
(.05 1.51 .02 2.5)
(.21 1.51 .02 3.0)
(.09 1.51 .02 2.8)
(.12 1.51 .02 2.9)
(4.25 1.68 .3 6.4)
(.85 1.68 .3 6.9)
(.62 1.68 .3 7.2)
(.52 1.68 .3 7.5)
(1.52 2.01 0.3 0.4)
(1.50 1.98 0.3 0.4)
(0.93 1.23 0.3 0.4)
(1.13 1.5 0.3 0.4)
(2.35 6.77 0.3 1.02)
(2.35 2.93 0.75 1.02)
(2.35 2.24 1.1 1.02)
(2.35 4.17 0.5 1.02)
(.8 .37 1.7 .48)
(.8 .39 1.9 .48)
(.8 .49 2.3 .48)
(.8 2.61 3.0 .48)
(1.44 7.67 6.4 .67)
(1.44 1.54 6.9 .67)
(1.44 1.12 7.2 .67)
(1.44 .99 7.4 .67)
(2.6 0.62 0.98 0.2)
(2.6 1.85 0.98 0.634)
(2.6 2.91 0.98 1.2)
(2.6 2.24 0.98 0.8)
(2.6 1.21 0.98 .4)

```
(2.6          1.63   0.98   .55)
(1.08         1.31   0.19   2.91)
(1.08         0.69   0.19   3.02)
(1.08         -4.69  0.19   4.1)
(1.08         -2.00  0.19   3.5)
(1.788 -2.13  0.93   5)
(1.788 0.90   0.93   6.7)
(1.788 1.89   0.93   7.3)
(1.788 1.46   0.93   7.0)
(2.5          2.8    0.12   0.134)
(2.5          2.8    0.32   0.359)
(2.5          2.8    0.39   0.45)
(2.5          2.8    0.58   0.67)
(.7           1.1    .68    1.74)
(.7           1.1    .61    2.03)
(.7           1.1    .21    2.8)
(.7           1.1    .07    3.03)
(.5           1.1    .05    6.4)
(.5           1.1    .14    6.6)
(.5           1.1    .33    7.1)
(.5           1.1    .44    7.5)
 )
)
```

=-=- Parameters -=-=

MAXSTAR = 10
COVERMODE = IC
UNCERTAINTY = 0.02
CONSTANCY = 0.95
INVERSE-SW = 30
DIRECT-SW = 70
UNIVIGNORE = NIL
DEBUG = NIL
PRINT-FLAGS = (P E)
FILTER-LEVEL = 2
SEARCH-LIMIT = 3

MAX-NODES = 1000
CLUSTER-ON = T
CLUSTER-ORDER = (GC NC RC)
MINIMUM-LINK-VALUE = 50.0

starLEF = ((MAX-COVERED 0.5) (MIN-WEIGHT 0.0) (MIN-SELECTORS 0.0))

==-==- Class Events -==-==

CLASSA
(N1 N2 THETA2 THETA1)
(2.5 2.8 0.587704 0.67)
(0.8 2.6177955 3.0 0.48)
(2.5 2.8 0.11956984 0.134)
(0.62554383 1.68 0.3 7.2)
(-7.2614417 1.8 5.0 0.24)
(1.08 -4.6793623 0.19 4.1)
(1.788 -2.1388648 0.93 5.0)
(1.08 -2.0059733 0.19 3.5)
(0.8 0.37252843 1.7 0.48)
(0.8 0.39038712 1.9 0.48)
(0.8 0.49540132 2.3 0.48)
(2.6 0.62196493 0.98 0.2)
(1.08 0.69362235 0.19 3.02)
(1.788 0.9030106 0.93 6.7)
(1.44 0.99500585 7.4 0.67)
(1.7727169 1.01 1.0 0.5)
(1.1895255 1.01 0.6 0.5)
(1.01 1.01 0.5 0.5)
(0.06319116 1.01 0.03 0.5)
(0.7 1.1 0.6070415 2.03)
(0.5 1.1 0.33773357 7.1)
(0.5 1.1 0.14208728 6.6)
(0.7 1.1 0.070925675 3.03)
(0.5 1.1 0.05300177 6.4)
(1.44 1.126693 7.2 0.67)
(2.6 1.2191341 0.98 0.4)
(0.9334174 1.23 0.3 0.4)
(-1.1152318 1.24 4.0 1.0)
(-0.51691765 1.24 3.5 1.0)
(0.20795585 1.24 3.0 1.0)
(1.0988786 1.24 2.3 1.0)
(1.4613272 1.24 1.7 1.0)
(1.08 1.3125678 0.19 2.91)
(1.788 1.4653977 0.93 7.0)
(1.1383139 1.5 0.3 0.4)

(2.4451091 1.51 0.2 0.123)
(0.5585338 1.51 0.2 0.567)
(0.39930588 1.51 0.2 0.85)
(0.33661154 1.51 0.2 1.1)
(0.21398798 1.51 0.02 3.0)
(0.12621978 1.51 0.02 2.9)
(0.09014642 1.51 0.02 2.8)
(0.050458465 1.51 0.02 2.5)
(0.033364605 1.51 0.02 2.01)
(1.44 1.5459167 6.9 0.67)
(2.6 1.636353 0.98 0.55)
(4.2597756 1.68 0.3 6.4)
(0.85829824 1.68 0.3 6.9)
(5.587028 1.8 7.113 0.24)
(3.7416677 1.8 6.8 0.24)
(0.42999822 1.8 6.34 0.24)
(2.6 1.8545122 0.98 0.634)
(1.788 1.8968848 0.93 7.3)
(1.5025743 1.98 0.3 0.4)
(1.5253406 2.01 0.3 0.4)
(2.6 2.2457938 0.98 0.8)
(2.35 2.2468998 1.1 1.02)
(2.6 2.917892 0.98 1.2)
(2.35 2.9377053 0.75 1.02)
(2.35 4.176778 0.5 1.02)
(2.35 6.77603 0.3 1.02)
(1.44 7.6724596 6.4 0.67)
(2.0299163 1.01 1.3 0.5)
(0.7 1.1 0.6780504 1.74)
(0.5 1.1 0.44046894 7.5)
(0.7 1.1 0.2148228 2.8)
(0.52928984 1.68 0.3 7.5)
(-5.3426943 1.8 5.5 0.24)
(2.5 2.8 0.39885348 0.45)
(2.5 2.8 0.31908178 0.359)

=-=- Hypotheses -=-=

CLASSA
Cover: Relation:
N2 = ((SIN (THETA1) * N1) / SIN (THETA2))

B.3 Instantaneous Current

The experiment with the instantaneous current demonstrates the ability to find non-linear terms composed of more than one variable. In this example, the term ωt had to be generated before $\cos(\omega t)$ could be found.

```
(
(declarations
 (i L ((amperes 1)))
 (R L ((volts 1) (amperes -1)))
 (Vo L ((volts 1)))
 (omega L ((radians 1) (seconds -1)))
 (time L ((seconds 1)))
)

(parameters
 (*inverse-sw* 20)
 (*direct-sw* 80)
 (*debug* t)
)

(variables
 (i R Vo omega time)
)

(events
 (3.29      65.8       220    337    0.13)
 (1.72      125.8      220    337    0.13)
 (2.852 75.98     220    337    0.13)
 (0.403 110       198    412    0.5)
 (0.366 121       198    412    0.5)
 (0.248 178       198    412    0.5)
 (1.024 35.4      110    451    0.6018)
 (2.049 35.4      220    451    0.6018)
 (0.707 35.4      75.9   451    0.6018)
 (0.011 167       312    857.3  0.273)
 (-1.450    167        312    857.3  0.8473)
 (-0.215    408        110    483.2  0.9284)
 (0.038 408       110    483.2  0.172)
 (0.229 408       110    483.2  0.584)
 (0.9       -130.741   121    575    0.1034)
```

(0.9 -123.153 121 575 0.5744)
(0.9 129.874 121 575 0.382)
(2.87 -55.083 289 4428 0.832)
(2.87 -77.890 289 4428 0.7072)
(2.87 -91.089 289 4428 0.344)
(2.87 -93.951 289 4428 0.54)
(0.821 143 273.3 3948.3 0.8453)
(0.355 143 273.3 4837.3 0.8453)
(0.674 143 273.3 938 0.8453)
(-0.267 523.4 214.4 3824.3 0.712)
(0.403 523.4 214.4 873.4 0.712)
(-0.117 523.4 214.4 8363.2 0.712)
(0.006 1200 56.56 38 0.745)
(0.003 1200 56.56 29 0.745)
(0.046 1200 56.56 92 0.745)
(14.3 445.4 13580.7 382 0.743)
(14.3 445.4 -11384.0 2483.3 0.743)
(14.3 445.4 6490.25 338 0.743)
(7.5 323 20376.26 3289 413)
(7.5 323 6500.31 3289 0.413)
(7.5 323 5997.85 2918.2 0.413)
(7.5 323 -3920.03 238 0.413)
(5.16 318 -1654.29 2872.3 0.1323)
(5.16 318 -2484.70 2872.3 0.4383)
(5.16 318 -8932.97 2872.3 0.2828)
(0.987 102.3 2877.35 382.3 0.1439)
(0.987 102.3 183.19 382.3 0.1289)
(0.987 102.3 -24831.43 382.3 0.415)
(0.41 220 103 1.62 0.31)
(0.41 220 103 3.54 0.1422)
(0.41 220 103 7.45 0.0676)
(0.132 98 217 2.73 0.554)
(0.132 98 217 3.57 0.4233)
(0.132 98 217 16.98 0.089)
(2.09 31.04 75 848.8 0.3621)
(2.09 49.667 120 848.8 0.3621)
(2.09 72.430 175 848.8 0.3621)
(4.53 39.961 313 773.3 0.83)
(4.53 7.393 313 773.3 0.9)
(4.53 54.87 313 773.3 0.277)
(4.53 45.118 313 773.3 0.708)
(10.2 19.921 266 998.1 1.82)
(10.2 -19.182 266 998.1 0.783)
(10.2 -17.272 266 998.1 0.388)
(7.1 543 -3858.92 416.3 0.098)
(7.1 543 4160.03 416.3 0.799)
(7.1 543 18707.39 416.3 0.691)
(0.96 612 591.686 588 0.823)

```
(0.96          612          3503.028 588   0.382)
(0.96          612          638.594 588    0.384)
 )
)
```

=-=- Parameters -=-=

```
*MAXSTAR* = 10
*COVERMODE* = IC
*UNCERTAINTY* = 0.02
*CONSTANCY* = 0.95
*INVERSE-SW* = 20
*DIRECT-SW* = 80
*UNIVIGNORE* = NIL
*DEBUG* = T
*PRINT-FLAGS* = (P E)
*FILTER-LEVEL* = 2
*SEARCH-LIMIT* = 3
*MAX-NODES* = 1000
*CLUSTER-ON* = T
*CLUSTER-ORDER* = (NC GC RC)
*BEAM-VALUE* = 0.5
*MINIMUM-LINK-VALUE* = 50.0

*starLEF* = ((MAX-COVERED 0.5) (MIN-WEIGHT 0.0) (MIN-SELECTORS 0.0))
```

=-=- Class Events -=-=

```
 CLASSA
(I R VO OMEGA TIME)
(14.3 445.4 13580.729 382.0 0.743)
(5.16 318.0 -8932.971 2872.3 0.2828)
(2.87 -93.95117 289.0 4428.0 0.54)
(10.2 -17.272163 266.0 998.1 0.388)
(7.1 543.0 18707.385 416.3 0.691)
(7.1 543.0 4160.0293 416.3 0.799)
```

(7.1 543.0 –3858.9182 416.3 0.098)
(2.8526256 75.98 220.0 337.0 0.13)
(2.09 72.43076 175.0 848.8 0.3621)
(0.987 102.3 –24831.43 382.3 0.415)
(0.9 –123.15287 121.0 575.0 0.5744)
(0.9 129.87405 121.0 575.0 0.382)
(0.0107615795 167.0 312.0 857.3 0.273)
(–0.11722943 523.4 214.4 8363.2 0.712)
(–0.21546386 408.0 110.0 483.2 0.9284)
(–1.4499393 167.0 312.0 857.3 0.8473)
(–1.5206991 167.0 312.0 857.3 0.472)
(14.3 445.4 6490.2524 338.0 0.743)
(14.3 445.4 –11384.064 2483.3 0.743)
(10.2 19.9217 266.0 998.1 1.82)
(7.5 323.0 20376.268 3289.0 413.0)
(7.5 323.0 6500.317 3289.0 0.413)
(7.5 323.0 5997.8525 2918.2 0.413)
(7.5 323.0 –3920.034 238.0 0.413)
(5.16 318.0 –1654.2872 2872.3 0.1323)
(5.16 318.0 –2484.7036 2872.3 0.4383)
(4.53 7.392655 313.0 773.3 0.9)
(4.53 39.961258 313.0 773.3 0.83)
(4.53 54.87472 313.0 773.3 0.2771)
(3.293959 65.8 220.0 337.0 0.13)
(2.87 –77.89037 289.0 4428.0 0.7072)
(2.87 –55.08361 289.0 4428.0 0.832)
(2.09 49.666813 120.0 848.8 0.3621)
(2.09 31.04176 75.0 848.8 0.3621)
(2.0488336 35.4 220.0 451.0 0.6018)
(1.0244168 35.4 110.0 451.0 0.6018)
(0.96 612.0 3503.0276 588.0 0.3823)
(0.96 612.0 638.5944 588.0 0.384)
(0.9 –130.74095 121.0 575.0 0.1034)
(0.8212155 143.0 273.3 3948.3 0.8453)
(0.674562 143.0 273.3 938.0 0.8453)
(0.41 220.0 103.0 1.6253369 0.31)
(0.41 220.0 103.0 3.5432804 0.1422)
(0.41 220.0 103.0 7.453469 0.0676)
(0.40340808 523.4 214.4 873.4 0.712)
(0.4027866 110.0 198.0 412.0 0.5)
(0.36616963 121.0 198.0 412.0 0.5)
(0.3551025 143.0 273.3 4837.3 0.8453)
(0.24891306 178.0 198.0 412.0 0.5)
(0.22920708 408.0 110.0 483.2 0.584)
(0.046101775 1200.0 56.56 928.0 0.745)
(0.0381025 408.0 110.0 483.2 0.172)
(0.0056263898 1200.0 56.56 3827.0 0.745)
(0.002560051 1200.0 56.56 2937.0 0.745)

(-0.2674998 523.4 214.4 3824.3 0.712)
(10.2 -19.182266 266.0 998.1 0.783)
(4.53 45.117954 313.0 773.3 0.708)
(2.87 -91.08931 289.0 4428.0 0.344)
(1.7229134 125.6 220.0 337.0 0.13)
(0.987 102.3 2877.3518 382.3 0.1439)
(0.987 102.3 183.18755 382.3 0.1289)
(0.96 612.0 591.68634 588.0 0.823)
(0.70684767 35.4 75.9 451.0 0.6018)
(0.132 98.0 217.0 2.727704 0.554)
(0.132 98.0 217.0 3.5699222 0.4233)
(0.132 98.0 217.0 16.979193 0.089)

==-==- Hypotheses -==-==

CLASSA
Cover: Relation:
I * R = COS ((OMEGA * TIME)) * VO

B.4 Stoke's Law

Stoke's law helps demonstrate the need for clustering. Notice that in the first experiment using this data, the *CLUSTER-ORDER* is set so that the first cluster tried is the global cluster.

```
(
(declarations
   (v  L  ((meter 1) (sec -1)))
   (r  L  ((meter 1)))
   (m  L  ((kg 1)))
   (t  L  ((sec 1)))
   (h  L  ((meter 1)))
   (substance N nil)
   (location N nil)
```

```
)

(parameters
   (*univignore* (3 4))
   (*uncertainty* 0.02)
   (*cluster-order* (gc nc rc))
)

(variables
   (v r m t h substance location)
)

(events
   (11.9425 0.05  0.6231 0.0837 1.0 Glycerol DeathValley)
   (26.8705 0.075 2.1029 0.0372 1.0 Glycerol DeathValley)
   (47.7698 0.1   4.9847 0.0209 1.0 Glycerol DeathValley)
   (18.0642 0.05  0.9425 0.0554 1.0 Glycerol DeathValley)
   (40.6445 0.075 3.1809 0.0246 1.0 Glycerol DeathValley)
   (72.2569 0.1   7.5398 0.0138 1.0 Glycerol DeathValley)
   (9.1671  0.05  0.6231 0.1091 1.0 CastorOil DeathValley)
   (20.6260 0.075 2.1029 0.0485 1.0 CastorOil DeathValley)
   (36.6684 0.1   4.9847 0.0273 1.0 CastorOil DeathValley)
   (13.8662 0.05  0.9425 0.0721 1.0 CastorOil DeathValley)
   (31.1989 0.075 3.1809 0.0321 1.0 CastorOil DeathValley)
   (55.4648 0.1   7.5398 0.0180 1.0 CastorOil DeathValley)
   (11.8757 0.05  0.6231 0.0842 1.0 Glycerol Denver)
   (26.7204 0.075 2.1029 0.0374 1.0 Glycerol Denver)
   (47.5030 0.1   4.9847 0.0211 1.0 Glycerol Denver)
   (17.9633 0.05  0.9425 0.0557 1.0 Glycerol Denver)
   (40.4174 0.075 3.1809 0.0247 1.0 Glycerol Denver)
   (71.8532 0.1   7.5398 0.0139 1.0 Glycerol Denver)
   (9.1159  0.05  0.6231 0.1097 1.0 CastorOil Denver)
   (20.5107 0.075 2.1029 0.0488 1.0 CastorOil Denver)
   (36.4635 0.1   4.9847 0.0274 1.0 CastorOil Denver)
   (13.7887 0.05  0.9425 0.0725 1.0 CastorOil Denver)
   (31.0246 0.075 3.1809 0.0322 1.0 CastorOil Denver)
   (55.1549 0.1   7.5398 0.0181 1.0 CastorOil Denver)
   (11.9425 0.05  0.6231 0.1675 2.0 Glycerol DeathValley)
   (26.8705 0.075 2.1029 0.0744 2.0 Glycerol DeathValley)
   (47.7698 0.1   4.9847 0.0419 2.0 Glycerol DeathValley)
   (18.0642 0.05  0.9425 0.1107 2.0 Glycerol DeathValley)
   (40.6445 0.075 3.1809 0.0492 2.0 Glycerol DeathValley)
   (72.2569 0.1   7.5398 0.0277 2.0 Glycerol DeathValley)
   (9.1671  0.05  0.6231 0.2182 2.0 CastorOil DeathValley)
   (20.6260 0.075 2.1029 0.0970 2.0 CastorOil DeathValley)
   (36.6684 0.1   4.9847 0.0545 2.0 CastorOil DeathValley)
   (13.8662 0.05  0.9425 0.1442 2.0 CastorOil DeathValley)
   (31.1989 0.075 3.1809 0.0641 2.0 CastorOil DeathValley)
```

```
  (55.4648 0.1   7.5398 0.0361 2.0 CastorOil DeathValley)
  (11.8757 0.05  0.6231 0.1684 2.0 Glycerol  Denver)
  (26.7204 0.075 2.1029 0.0748 2.0 Glycerol  Denver)
  (47.5030 0.1   4.9847 0.0421 2.0 Glycerol  Denver)
  (17.9633 0.05  0.9425 0.1113 2.0 Glycerol  Denver)
  (40.4174 0.075 3.1809 0.0495 2.0 Glycerol  Denver)
  (71.8532 0.1   7.5398 0.0278 2.0 Glycerol  Denver)
  (9.1159  0.05  0.6231 0.2194 2.0 CastorOil Denver)
  (20.5107 0.075 2.1029 0.0975 2.0 CastorOil Denver)
  (36.4635 0.1   4.9847 0.0548 2.0 CastorOil Denver)
  (13.7887 0.05  0.9425 0.1450 2.0 CastorOil Denver)
  (31.0246 0.075 3.1809 0.0645 2.0 CastorOil Denver)
  (55.1549 0.1   7.5398 0.0363 2.0 CastorOil Denver)
  (4.4373  0.05  0.6231 0.4507 1.0 Vacuum    DeathValley)
  (4.4373  0.075 2.1029 0.4507 1.0 Vacuum    DeathValley)
  (4.4373  0.1   4.9847 0.4507 1.0 Vacuum    DeathValley)
  (4.4373  0.05  0.9425 0.4507 1.0 Vacuum    DeathValley)
  (4.4373  0.075 3.1809 0.4507 1.0 Vacuum    DeathValley)
  (4.4373  0.1   7.5398 0.4507 1.0 Vacuum    DeathValley)
  (4.4249  0.05  0.6231 0.4520 1.0 Vacuum    Denver)
  (4.4249  0.075 2.1029 0.4520 1.0 Vacuum    Denver)
  (4.4249  0.1   4.9847 0.4520 1.0 Vacuum    Denver)
  (4.4249  0.05  0.9425 0.4520 1.0 Vacuum    Denver)
  (4.4249  0.075 3.1809 0.4520 1.0 Vacuum    Denver)
  (4.4249  0.1   7.5398 0.4520 1.0 Vacuum    Denver)
  (6.2753  0.05  0.6231 0.6374 2.0 Vacuum    DeathValley)
  (6.2753  0.075 2.1029 0.6374 2.0 Vacuum    DeathValley)
  (6.2753  0.1   4.9847 0.6374 2.0 Vacuum    DeathValley)
  (6.2753  0.05  0.9425 0.6374 2.0 Vacuum    DeathValley)
  (6.2753  0.075 3.1809 0.6374 2.0 Vacuum    DeathValley)
  (6.2753  0.1   7.5398 0.6374 2.0 Vacuum    DeathValley)
  (6.2578  0.05  0.6231 0.6392 2.0 Vacuum    Denver)
  (6.2578  0.075 2.1029 0.6392 2.0 Vacuum    Denver)
  (6.2578  0.1   4.9847 0.6392 2.0 Vacuum    Denver)
  (6.2578  0.05  0.9425 0.6392 2.0 Vacuum    Denver)
  (6.2578  0.075 3.1809 0.6392 2.0 Vacuum    Denver)
  (6.2578  0.1   7.5398 0.6392 2.0 Vacuum    Denver)
 )
)
```

=-=- Parameters -=-=

MAXSTAR = 10
COVERMODE = IC
UNCERTAINTY = 0.02
CONSTANCY = 0.99
INVERSE-SW = 30
DIRECT-SW = 70
UNIVIGNORE = (3 4)
DEBUG = NIL
PRINT-FLAGS = (P S)
FILTER-LEVEL = 2
SEARCH-LIMIT = 3
MAX-NODES = 1000
CLUSTER-ON = T
CLUSTER-ORDER = (GC NC RC)
BEAM-VALUE = 0.5
MINIMUM-LINK-VALUE = 50.0

starLEF = ((MAX-COVERED 0.5) (MIN-WEIGHT 0.0) (MIN-SELECTORS 0.0))

=-=- Class Events -=-=

CLASSC
(V R M T H SUBSTANCE LOCATION)
(9.1159 0.05 0.6231 0.1097 1.0 CASTOROIL DENVER)
(9.1159 0.05 0.6231 0.2194 2.0 CASTOROIL DENVER)
(13.7887 0.05 0.9425 0.0725 1.0 CASTOROIL DENVER)
(13.7887 0.05 0.9425 0.145 2.0 CASTOROIL DENVER)
(31.0246 0.075 3.1809 0.0322 1.0 CASTOROIL DENVER)
(31.0246 0.075 3.1809 0.0645 2.0 CASTOROIL DENVER)
(36.4635 0.1 4.9847 0.0274 1.0 CASTOROIL DENVER)
(36.4635 0.1 4.9847 0.0548 2.0 CASTOROIL DENVER)
(20.5107 0.075 2.1029 0.0488 1.0 CASTOROIL DENVER)
(20.5107 0.075 2.1029 0.0975 2.0 CASTOROIL DENVER)
(55.1549 0.1 7.5398 0.0181 1.0 CASTOROIL DENVER)
(55.1549 0.1 7.5398 0.0363 2.0 CASTOROIL DENVER)
(9.1671 0.05 0.6231 0.1091 1.0 CASTOROIL DEATHVALLEY)
(9.1671 0.05 0.6231 0.2182 2.0 CASTOROIL DEATHVALLEY)
(13.8662 0.05 0.9425 0.0721 1.0 CASTOROIL DEATHVALLEY)

(13.8662 0.05 0.9425 0.1442 2.0 CASTOROIL DEATHVALLEY)
(31.1989 0.075 3.1809 0.0321 1.0 CASTOROIL DEATHVALLEY)
(31.1989 0.075 3.1809 0.0641 2.0 CASTOROIL DEATHVALLEY)
(36.6684 0.1 4.9847 0.0273 1.0 CASTOROIL DEATHVALLEY)
(36.6684 0.1 4.9847 0.0545 2.0 CASTOROIL DEATHVALLEY)
(20.626 0.075 2.1029 0.0485 1.0 CASTOROIL DEATHVALLEY)
(20.626 0.075 2.1029 0.097 2.0 CASTOROIL DEATHVALLEY)
(55.4648 0.1 7.5398 0.018 1.0 CASTOROIL DEATHVALLEY)
(55.4648 0.1 7.5398 0.0361 2.0 CASTOROIL DEATHVALLEY)

CLASSB
(V R M T H SUBSTANCE LOCATION)
(11.8757 0.05 0.6231 0.0842 1.0 GLYCEROL DENVER)
(11.8757 0.05 0.6231 0.1684 2.0 GLYCEROL DENVER)
(17.9633 0.05 0.9425 0.0557 1.0 GLYCEROL DENVER)
(17.9633 0.05 0.9425 0.1113 2.0 GLYCEROL DENVER)
(40.4174 0.075 3.1809 0.0247 1.0 GLYCEROL DENVER)
(40.4174 0.075 3.1809 0.0495 2.0 GLYCEROL DENVER)
(47.503 0.1 4.9847 0.0211 1.0 GLYCEROL DENVER)
(47.503 0.1 4.9847 0.0421 2.0 GLYCEROL DENVER)
(26.7204 0.075 2.1029 0.0374 1.0 GLYCEROL DENVER)
(26.7204 0.075 2.1029 0.0748 2.0 GLYCEROL DENVER)
(71.8532 0.1 7.5398 0.0139 1.0 GLYCEROL DENVER)
(71.8532 0.1 7.5398 0.0278 2.0 GLYCEROL DENVER)
(18.0642 0.05 0.9425 0.0554 1.0 GLYCEROL DEATHVALLEY)
(18.0642 0.05 0.9425 0.1107 2.0 GLYCEROL DEATHVALLEY)
(11.9425 0.05 0.6231 0.0837 1.0 GLYCEROL DEATHVALLEY)
(11.9425 0.05 0.6231 0.1675 2.0 GLYCEROL DEATHVALLEY)
(40.6445 0.075 3.1809 0.0246 1.0 GLYCEROL DEATHVALLEY)
(40.6445 0.075 3.1809 0.0492 2.0 GLYCEROL DEATHVALLEY)
(47.7698 0.1 4.9847 0.0419 2.0 GLYCEROL DEATHVALLEY)
(26.8705 0.075 2.1029 0.0372 1.0 GLYCEROL DEATHVALLEY)
(26.8705 0.075 2.1029 0.0744 2.0 GLYCEROL DEATHVALLEY)
(72.2569 0.1 7.5398 0.0138 1.0 GLYCEROL DEATHVALLEY)
(72.2569 0.1 7.5398 0.0277 2.0 GLYCEROL DEATHVALLEY)

CLASSA
(V R M T H SUBSTANCE LOCATION)
(4.4249 0.1 7.5398 0.452 1.0 VACUUM DENVER)
(4.4249 0.1 4.9847 0.452 1.0 VACUUM DENVER)
(4.4249 0.075 3.1809 0.452 1.0 VACUUM DENVER)
(4.4249 0.075 2.1029 0.452 1.0 VACUUM DENVER)
(4.4249 0.05 0.9425 0.452 1.0 VACUUM DENVER)
(4.4249 0.05 0.6231 0.452 1.0 VACUUM DENVER)
(6.2578 0.1 4.9847 0.6392 2.0 VACUUM DENVER)
(6.2578 0.075 3.1809 0.6392 2.0 VACUUM DENVER)
(6.2578 0.075 2.1029 0.6392 2.0 VACUUM DENVER)
(6.2578 0.05 0.9425 0.6392 2.0 VACUUM DENVER)

(6.2578 0.05 0.6231 0.6392 2.0 VACUUM DENVER)
(6.2753 0.1 7.5398 0.6374 2.0 VACUUM DEATHVALLEY)
(6.2753 0.1 4.9847 0.6374 2.0 VACUUM DEATHVALLEY)
(6.2753 0.075 3.1809 0.6374 2.0 VACUUM DEATHVALLEY)
(6.2753 0.075 2.1029 0.6374 2.0 VACUUM DEATHVALLEY)
(6.2753 0.05 0.9425 0.6374 2.0 VACUUM DEATHVALLEY)
(6.2753 0.05 0.6231 0.6374 2.0 VACUUM DEATHVALLEY)
(4.4373 0.1 7.5398 0.4507 1.0 VACUUM DEATHVALLEY)
(4.4373 0.1 4.9847 0.4507 1.0 VACUUM DEATHVALLEY)
(4.4373 0.075 3.1809 0.4507 1.0 VACUUM DEATHVALLEY)
(4.4373 0.075 2.1029 0.4507 1.0 VACUUM DEATHVALLEY)
(4.4373 0.05 0.9425 0.4507 1.0 VACUUM DEATHVALLEY)
(4.4373 0.05 0.6231 0.4507 1.0 VACUUM DEATHVALLEY)

=-=- Hypotheses -=-=

CLASSA
Cover:
24 [SUBSTANCE = VACUUM]
Relation:
V = 9.8175 * T

CLASSB
Cover:
24 [SUBSTANCE = GLYCEROL]
Relation:
R * V = 0.9556 * M

CLASSC
Cover:
24 [SUBSTANCE = CASTOROIL]
Relation:
R * V = 0.7336 * M

In the second experiment, the equation form (= v (+ (* c1 t))) was included which fit on of the subsets. The resulting equations were identical to the ones discovered by the heuristic search method.

=-=- Parameters -=-=

MAXSTAR = 10
COVERMODE = IC
UNCERTAINTY = 0.02
CONSTANCY = 0.97
INVERSE-SW = 30
DIRECT-SW = 70
UNIVIGNORE = (3 4)
DEBUG = NIL
PRINT-FLAGS = (P S)
FILTER-LEVEL = 2
SEARCH-LIMIT = 3
MAX-NODES = 1000
CLUSTER-ON = T
CLUSTER-ORDER = (GC NC RC)
BEAM-VALUE = 0.5
MINIMUM-LINK-VALUE = 50.0

starLEF = ((MAX-COVERED 0.5) (MIN-WEIGHT 0.0) (MIN-SELECTORS 0.0))

=-=- Hypotheses -=-=

CLASSA
Cover: Relation:
V = 9.81748 * T

CLASSB
Cover: Relation:
R * V = 0.9556 * M

CLASSC
Cover: Relation:
R * V = 0.7336 * M

B.5 Angular Conservation of Momentum

This simple experiment shows the use of linear regression in ABACUS.2 In this case, ABACUS.2 was given 3 equation forms, but only the last one had a constancy which exceeded the *CONSTANCY* threshold.

```
(
(declarations
 (m L ((meters 1)))
 (Vo L ((meters 1) (seconds -1)))
 (V L ((meters 1) (seconds -1)))
 (I L ((kilogram 1) (meters 2)))
 (omega L ((radians 1) (seconds -2)))
 (Ediss L ((joules 1))))

(parameters
 )

(variables
 (m Vo V I omega Ediss)
 )

(equations
 (= Ediss (+ (* c1 (* m Vo Vo)) (* c2 (- (* I omega omega)))))
 (= Ediss (+ (* c1 (* m V V)) (* c2 (- (* m V V)))))
 (= Ediss (+ (* c1 (* m Vo Vo)) (* c2 (- (* m V V))) (* c3 (- (* I omega omega)))))
 )

(events
 (40.0        6.77   3.11   720.0  0.667  563.05)
 (55.6        5.96   4.89   801.7  0.501  222.13)
 (34.65 4.86  2.3    685.6  0.473  240.86)
 (69.48 5.21  4.5    699.0  0.801  15.26)
 (34.5        4.09   2.51   598.8  0.542  91.93)
 (43.0        6.34   4.4    740.9  0.603  313.26)
 (50.2        7.21   4.89   798.2  0.66   530.75)
 (28.9        3.87   1.98   402.0  0.329  138.01)
 (47.5        4.32   3.95   574.5  0.354  36.67)
 (38.5        4.68   3.75   583.4  0.473  85.65)
 (47.5        5.68   4.53   754.0  0.385  222.97)
 (51.0        5.84   4.367  803    0.574  251.10)
 )
)
```

=-=- Parameters -=-=

MAXSTAR = 10
COVERMODE = IC
UNCERTAINTY = 0.02
CONSTANCY = 0.95
INVERSE-SW = 30
DIRECT-SW = 70
UNIVIGNORE = NIL
DEBUG = NIL
PRINT-FLAGS = (P E)
FILTER-LEVEL = 2
SEARCH-LIMIT = 3
MAX-NODES = 1000
CLUSTER-ON = T
CLUSTER-ORDER = (NC GC RC)
BEAM-VALUE = 0.5
MINIMUM-LINK-VALUE = 50.0

starLEF = ((MAX-COVERED 0.5) (MIN-WEIGHT 0.0) (MIN-SELECTORS 0.0))

=-=- Class Events -=-=

CLASSA
(M VO V I OMEGA EDISS)
(69.48 5.21 4.5 699.0 0.801 15.26)
(47.5 4.32 3.95 574.5 0.354 36.67)
(38.5 4.68 3.75 583.4 0.473 85.65)
(34.5 4.09 2.51 598.8 0.542 91.93)
(28.9 3.87 1.98 402.0 0.329 138.01)
(55.6 5.96 4.89 801.7 0.501 222.13)
(47.5 5.68 4.53 754.0 0.385 222.97)
(34.65 4.86 2.3 685.6 0.473 240.86)
(51.0 5.84 4.367 803.0 0.574 251.10)
(43.0 6.34 4.4 740.9 0.603 313.26)
(50.2 7.21 4.89 798.2 0.66 530.75)
(40.0 6.77 3.11 720.0 0.667 563.05)

=-=- Hypotheses -=-=

CLASSA
Cover: Relation:
EDISS = 0.4999994 * (M * VO * VO) + 0.4999996 * (- (M * V * V))
+ 0.49999753 * (- (I * OMEGA * OMEGA))

REFERENCES

[Becker, 1985]
Becker, J. M., "AQ-PROLOG: A Prolog Implementation of an Attribute-Based Inductive Learning System," UIUCDCS-F-85-930, ISG 85-1, Department of Computer Science, University of Illinois, 1985a.

[Chatterjee and Price, 1977]
Chatterjee, S., and Price, B. *Regression Analysis by Example.* New York: John Wiley and Sons.

[Daniel and Wood, 1980]
Daniel, C., and Wood, F. S. *Fitting Equations to Data.* New York: Wiley-Interscience, John Wiley and Sons.

[Dejong, 1986]
Dejong, G. and Mooney, R., "Explanation-Based Learning: An Alternative View," in *Machine Learning Journal*, vol. 2, 1986.

[Edwards, 1976]
Edwards, A. L., *An Introduction to Linear Regression and Correlation*, Freeman Press, San Francisco, 1975.

[El-Shafei, 1986]
El-Shafei, N. *Quantitative Discovery and Reasoning about Failure Mechanisms in Pavement.* TR, Artificial Intelligence Laboratory, MIT, Cambridge, Mass.

[Falkenhainer, 1985]
Falkenhainer, B. C., *Quantitative Empirical Learning: An Analysis and Methodology.* Master's thesis (UIUCDCS-F-85-947, ISG 85-16), Department of Computer Science, University of Illinois, Champaign-Urbana, Ill.

[Falkenhainer and Michalski, 1986]
Falkenhainer, B. C., and Michalski, R. S., "Integrating Quantitative and Qualitative Discovery: The ABACUS System," in *Machine Learning Journal*, vol. 3, 1986.

[Huntley, 1952]
Huntley, H. E. *Dimensional Analysis.* London: MacDonald and Co.

[Kokar, 1986]
Kokar, M. "Learning Arguments of Invariant Functional Descriptions," in *Machine Learning Journal*, vol. 1, 1986.

[Langhaar, 1951]
Langhaar, H. L. *Dimensional Analysis and Theory of Models.* John Wiley and Sons.

[Langley, Bradshaw, and Simon, 1985]
Langley, P., Bradshaw, G. L., and Simon, H. A. BACON.5: The Discovery of Conservation Laws. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 121-126).

[Langley, Zytkow, Simon, and Bradshaw, 1986]
Langley, P., Zytkow, J., Simon, H. A., and Bradshaw, G. L. The Search for Regularity: Four Aspects of Scientific Discovery, in *Machine Learning: An Artificial Intelligence Approach*, volume II, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga, Palo Alto, Calif., 1983.

[Michalski, 1975]
Michalski, R. S., "Synthesis of Optimal and Quasi-optimal Variable-valued Logic Formulas," *Proceedings of the 1975 International Symposium on Multiple-valued Logic*, Bloomington, Indiana, pp. 76–87, May 1975.

[Michalski and Larson, 1978]
Michalski, R. S., and Larson, J.B. *Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: The Underlying Methodology and the Description of Programs ESEL and AQ11* (TR UIUCDCS–R–78–867). Department of Computer Science, University of Illinois, Champaign–Urbana, Ill.

[Michalski, 1986]
Michalski, R. S., "Understanding the Nature of Learning." in *Machine Learning: An Artificial Intelligence Approach*, volume II, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga, Palo Alto, Calif., 1983.

[Minksy, 1985]
Minsky, M. *The Society of Mind*, MIT Press, Cambridge, 1985.

[Mitchell, 1986]
Mitchesll, T. M., Keller T., and Kedar–Cabelli, S., "Explanation–Based Generalisation: A Unifying View," in *Machine Learning Journal*, vol. 1, January 1986.

[O'Dwyer, 1981]
O'Dwyer, J., *College Physics*, Wadsworth Publishing Co., 1981.

[Simon, 1983]
Simon, H. A., "Why Should Machines Learn?" in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga, Palo Alto, Calif., 1983.