SIAM J. SCI. STAT. COMPUT.
Vol. 9, No. 1, January 1988

DINDE: TOWARDS MORE SOPHISTICATED SOFTWARE ENVIRONMENTS FOR STATISTICS*

R. W. OLDFORD† AND S. C. PETERS‡

Abstract. A prototype statistical system we call DINDE is described. DINDE is aimed at the professional statistician and provides a statistical analysis environment that is more sophisticated than the current generation of systems. In particular, it allows the analyst to keep careful track of the entire analysis as it progresses. General design philosophy and some issues of implementation are described and an example session is presented for illustration.

Key words. integrated programming environments, graphical interface, object oriented programming, statistical computing system

AMS(MOS) subject classification. 62-04

1. Introduction.

DINDE is a computer "system" for performing data analysis and statistics. More precisely, DINDE is an enrichment of an extensive interactive programming environment. The programming environment is Interlisp-D with LOOPS which runs on the Xerox 1109 personal workstation (Teitelman and Masinter (1981), Stefik et al. (1983) and Interlisp Reference Manual (1983)). DINDE provides the usual analytic, graphical and data management tools, which are accessible from an interpreted language. In addition, DINDE also organizes, tracks and occasionally guides the use of these tools, all in a distinctive visual format. A personal workstation provides the combination of high interaction, extensive graphics and powerful dedicated computing required by the ambitious aims of DINDE. DINDE is composed of many sophisticated interdependent procedures. We rely on the interactive programming environment to help us build, manage and especially experiment with this complex system.

Why did we invent another statistical system? First of all, like McDonald and Pedersen (1987), we think data analysts and practical statisticians do a special kind of programming work—especially when they are developing new methods. This kind of programming is referred to as "experimental," "exploratory" or "improvisational" in the computer science community. Interactive programming environments are the most appropriate and productive locales for doing experimental programming; this is borne out by our experience and is forcefully argued by Sheil (1983). We built DINDE to take advantage of the interactive programming environment available on the Xerox 1109. Our initial aim has been to facilitate our own research in data analysis and statistics. As we proceed we hope to organize our efforts and make them available to others.

Second, we have often been frustrated by working with existing interactive statistical systems. Especially vexing are the multiplicity of languages required to control them (these include command lines or an expression algebra, graphics subsystem commands, macros, "user function" or interface language, implementation language, preprocessors, etc.) and the paucity (often nonexistence) of language tools for the lot

* Received by the editors January 3, 1986; accepted for publication (in revised form) November 13, 1986. This research was supported by National Science Foundation grant IST-8420614, by the MIT Center for Computational Research in Economics and Management Science, by the MIT Statistics Center, and by an equipment grant from the Xerox Corporation.

† Present address, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Ontario N2L 3G1.

‡ Present address, Xerox AIS, Palo Alto, California 94304.

(i.e., interpreters and compilers, debuggers, inspectors, performance meters). By designing DINDE as an enrichment of the Interlisp-LOOPS environment, all the facilities and tools of Interlisp and LOOPS are available to the user. In particular, the same language features, compilers, debugging aids, etc. used by the systems programmers to construct Interlisp and LOOPS, and which we used to develop DINDE, are also available to the DINDE user. We think that we gain important leverage by embedding DINDE in this rich and painstakingly developed environment which allows us to focus our efforts on statistical programming.

Third, we want to explore and develop statistical software which is more sophisticated than programs currently available: we envision software which guides the choice of technique, interpretation of results and management of the analysis. Further, we are interested in studying the strategies used in practical statistical analysis and we perceive that this investigation requires a novel kind of statistical system (see Oldford and Peters (1986a)). DINDE is being constructed to meet what we anticipate might be the needs of such study.

The system we describe here is about a year old. Many details remain to be settled; others have been chosen by virtue of their expedience. DINDE is mostly a framework awaiting more hard work. Even so, the ease of use and usefulness of this approach are already becoming clear. In subsequent papers we plan to elaborate on the programming environment offered through DINDE. This paper introduces the novel user interface and some of the philosophy underlying the design of DINDE.

In building DINDE, we have been strongly influenced by the S data analysis and graphics system (Becker and Chambers (1984)) and often borrow ideas from it. The system design issues raised by David Donoho's DART (Donoho (1983)) have also had an impact on our work: we think many of Donoho's concerns are met by the programming environments now available on LISP machines. (DART stands for Data Analysis Research Tool; we regretted that the acronym had already been claimed and so had to settle for the self-referential "DINDE": DINDE Is Not DART Exactly. "DINDE" also has the virtue of being self-deprecatory, "dinde" being French for "turkey.") The plotting capabilities of DINDE rely on a collection of graphical functions developed by Jan Pedersen of Stanford University and Xerox AI Systems, who agreed to let us try out early versions of his work. (Pedersen's graphics package will also be incorporated in IDL, a system for data analysis developed at Xerox by Kaplan et al. (1981)).

Recent software that shares some of the objectives of DINDE has been described by Carr et al. (1984); its underlying philosophy has been detailed in Nicholson et al. (1984). That software appears to be very closely related to a formalism of data analysis proposed recently by Thisted (1985), but differs in many aspects from DINDE.

This paper is organized as follows. Section 2 is a very brief introduction to the characteristics of personal workstations, to the idea of an interactive programming environment and to the "object oriented" approach to programming which pervades DINDE. Readers already familiar with the advantages of these modern computing technologies may skip this section. DINDE is then introduced in § 3 with an example emphasizing the interactive, evolutionary nature of data analysis. The key themes underlying the implementation are developed in § 4, while § 5 relates DINDE to our current views on the study of statistical strategy. Some concluding remarks are given in the final section.

2. Prerequisites.

Our work is largely motivated by the present availability and promise of powerful, highly interactive, highly graphical, personal workstations. McDonald and Pedersen (1985a) and (1985b) carefully consider workstation architecture and its impact on data analysis. Our aim is to fashion a system for data analysis and statistics which takes advantage of the special nature of this new technology, some aspects of which we describe next.

The initial implementation of DINDE took place on the Xerox 1109 workstation. The 1109 incorporates a proprietary central processor developed by Xerox. The processor runs microcode which is specially tailored for Interlisp, so many Lisp functions (e.g., CONS, CAR, POLY (polynomial evaluation)) are actually machine language instructions, leading to impressive Lisp performance. Lisp attempts to present the illusion of an infinitely large, persistent memory space. To support this illusion, the processor manages a virtual memory of 32 Mbytes (real memory of 3.7 Mbytes), and assists the so-called garbage collection of unreferenced objects with microcode routines. A hardware floating point coprocessor is also installed on the 1109.

Central to the design of the workstation is its bit-mapped display and its "mouse" pointing device. The display screen is organized as an array of about 1000 by 1000 individually addressable pixels. A high capacity communications channel connects the display with a segment of processor memory, a 1000 by 1000 map of bits. Special processor instructions manipulate the bit-map, and dedicated hardware refreshes the display from the bit-map 38 times each second. This provides the foundation for the window system—a collection of overlapping screen regions capable of displaying dynamic, detailed, highly resolved, independent images. A pointing device, called the mouse, allows the user to focus attention and initiate action on particular portions of the display. As the user rolls the mouse along the tabletop, a small arrow (the pointer) moves in a corresponding fashion across the display screen. The mouse we use has two buttons, allowing three different combinations of buttons to be pressed: left, right and so-called "middle" (i.e., both buttons pressed). DINDE responds to the mouse in different ways depending on its screen location and the buttons being pressed. (In what follows, we occasionally refer to the action of pointing and then pressing the left, right or middle buttons as "mousing.") DINDE makes extensive use of windows, the mouse, pop-up menus, a variety of type fonts, shading, and point and line drawing (but no color yet). These facilities are of recent vintage and, for the most part, available together only in workstation configurations.

The workstation also has a keyboard, which we have come to regard as a clumsy companion to the mouse. A local area network (Ethernet) connects workstations to print, filing and mail services. Paging space and some file storage is maintained on a hard disk located on the workstation.

The personal workstation hardware just described supports a distinctive software system: the Interlisp-D integrated programming environment. (The following discussion is distilled from the Interlisp Reference manual (1983).) Interlisp is a programming system consisting of a programming language (Lisp) and an environment that supports the programmer by providing a variety of specialized programming tools. Lisp offers a rich collection of predefined programs that can be used either as direct, top-level commands or as subroutines in user programs (e.g., the text editor TEDIT that we have used to create this document is also called on internally by the DINDE system to add commentary to statistical objects). The environment supports program development and execution by providing an integrated set of programming tools which know a great deal about Lisp and can act as "assistants" to the programmer. These tools include not only program editors, compilers and debuggers, but also tools which assist the user by keeping track of changes to functions, variables and other data objects, and which "understand" certain dependency relations among these components. The programmer is thereby relieved of many mundane housekeeping and version management chores.

Our work with DINDE relies heavily on an extension of the Interlisp environment called LOOPS. LOOPS incorporates four important programming paradigms oriented, respectively, around procedures, objects, data access and rules. Procedure oriented programming, the most familiar, is offered by traditional programming languages (C, Pascal, etc.), and in LOOPS is provided by the underlying Lisp system. Object oriented programming was pioneered by Smalltalk (Goldberg and Robson (1983)) and the MIT Lisp Machine Flavors system (Weinreb and Moon (1981)). Object oriented programming is organized around entities called objects which have aspects both of procedures and of data. Computation in the object oriented scheme is accomplished by directing an object to execute a procedure and return a value, the procedure may refer to local state information unique to the object. (We elaborate on this description below.) In data oriented programming, action is potentially triggered when data are accessed. Sets of condition-action pairs (If-Then clauses) determine the behavior of programs built in the rule oriented paradigm. DINDE uses procedure and object oriented programming heavily at all levels, it uses data oriented programming internally and does not use rule oriented programming. The multiplicity of programming paradigms allows us to closely match each of the various structures comprising our system with an appropriate expression as a program.

(The following discussion is distilled from Bobrow and Stefik (1983); see also Goldberg and Robson (1983).) The object oriented paradigm of LOOPS is built on the following concepts.

Classes. A class is a description of one or more similar objects. An *instance* is an object described by a particular class. Every object is an instance of exactly one class. (At the outset it may be helpful to regard a class as a template. From time to time, instances of a class are created or instantiated. This entails filling in the template with specific details about the new object being created.) DINDE gives definitions for many classes related to data analysis and statistics; among these are the classes **Vector**, **ScatterPlot** and **BivariateRegression**.

Variables. There are two kinds of variables—instance variables and class variables. Instance variables contain information specific to an instance. Class variables are used to contain information shared by all instances of the class. A class variable is typically used for information about a class taken as a whole. Both kinds of variables have names, values and other properties. A class describes the structure of its instances by specifying the names and default values of instance variables. So for example, the class description for **ScatterPlot** directly defines two instance variables: X and Y (which identify **Vectors** giving the coordinates of the points), and also defines a class variable: **References** (which is a text string citing further general reading concerning scatterplots).

Methods. A class specifies the behavior of its instances in terms of their response to a special kind of procedure call (making such a call is usually called "sending a message"). The class associates a selector (a name) with a *method* (the function that responds to the selector). All instances of a class use the same selectors and methods. Any difference in response by two instances of the same class is determined by a difference in the values of their instance variables. The class **ScatterPlot**, for example, defines the method **PlotPoints**. An instance of **ScatterPlot** responds to the selector **PlotPoints** by drawing a scatterplot derived from its X and Y instance variables.

Inheritance. Inheritance is an important tool for organizing information in objects. It enables the easy creation of objects that are "almost like" other objects with a few incremental changes. Inheritance avoids redundant specification of information and simplifies modification, since information that is common is defined in, and need be changed in, only one place. An object may inherit some or all of its instance variable description and methods from one or more classes. The definition of a class identifies one or more parent classes from which variables and methods are inherited.

3. Using DINDE.

3. Using DINDE. DINDE is meant to be helpful to both the statistician doing research in data analysis and the statistician engaged in an actual analysis. Here, we discuss the latter situation: how a statistician is expected to use DINDE in a practical setting. Although we concentrate mainly on the visual screen and mouse interface to DINDE, it is important to remember that all the operations we describe can be accomplished in a Lisp *program* by calling functions or sending messages. By this mechanism the user can pursue analyses beyond those provided in the existing menu arrangements.

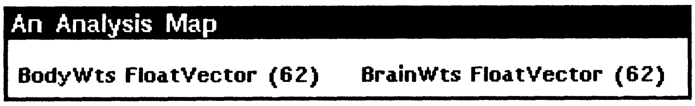
The basic idea to keep in mind is that, with DINDE, an analysis consists of creating, examining and performing operations on DINDE objects. A DINDE object represents either an element or a stage in a typical analysis; each object encapsulates some information generally considered to be relevant for that item or stage. (For example, a data vector or a scatterplot would be an element, while consideration of the regression of one variable on another, or the exploration of the structure of a two-way table, would be a stage.) As such, these objects are meant to constitute building blocks of the analysis.

Each DINDE object is created from a template called its class. Classes are organized hierarchically, that is, a class may have one or more parent classes, each of which is itself a class. DINDE gathers all these classes together and displays their hierarchy as an inheritance network in a window called the toolbox. Moving left to right in this display follows the inheritance from parent to child. (Section 4.1 displays several branches of the inheritance network which can be seen in the toolbox.) DINDE objects are instances of DINDE classes and may be created by selecting the class from the toolbox (i.e., by pointing at a label on the network),clicking a mouse key, and perhaps identifying components to be incorporated in the new object. Each class represented in the toolbox can briefly describe its intended use and the user has access to this toolbox at all times.

At the time a DINDE object is created, a visual symbol representing that object is placed in another window called an analysis map. There it can be examined in detail and directed to perform operations peculiar to its own kind. An object can be given a unique name and can have user supplied commentary (editable text) attached to it. DINDE objects often know how to spawn additional objects of like and unlike classes (e.g., the **LOG** transformation of an **Array** object delivers a new **Array**; the **BivariateRegression** object can produce specialized scatterplots)—our notion of software guidance rests largely on this ability.

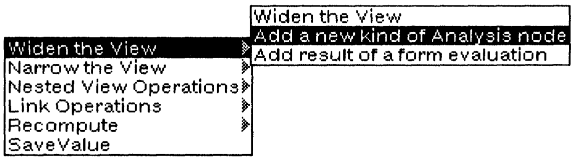
The analysis proceeds by creating more objects, examining them and linking them together. At any time the current state of the analysis is visible in the analysis map as a collection of statistically interesting objects some or all of which are linked together in one or more networks.

Before examining the details of the toolbox and the analysis maps (§§ 4.1 and 4.2), it may be best to consider a small example of their use. Assume that the user wants to fit a bivariate regression of Y on X. For brevity, also assume that the user has created two **FloatVectors** (e.g., using the function **C**—as in the S language, **C** concatenates small objects like numbers into a vector object) and has named them BrainWts and BodyWts. The data we will use in this example represent the average brain and body weights, respectively in grams and kilograms, of 62 mammals (Weisberg (1980, pp. 128–129)). These two objects appear as named **FloatVectors** of 62 elements in an Analysis Map window as shown below. (This map and subsequent displays are actual screen images—"bitmap" objects which have been copied directly into this document.)



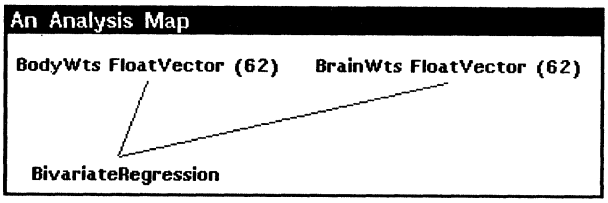
All DINDE windows are sensitive to mouse operations. In general, mousing in the title region of the window produces a menu whose entries offer operations relevant to the entire collection of objects displayed, while mousing on a particular object in the window produces menus allowing interaction with that object. There is a common set of housekeeping functions for all windows, and each DINDE window can briefly describe the way it may be used.

Data analysis using DINDE consists of building a network whose nodes are DINDE objects (for more detailed discussion of these networks and their display, see Oldford and Peters (1986b)). We begin the example analysis by moving the mouse to the title region of the map above and pressing the middle mouse button. This produces the following menu (arrows indicate that a further menu may be obtained by sliding the mouse to the right),



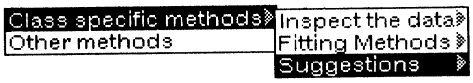
from which we choose to widen the view of this DINDE window by selecting the item "**Add a new kind of Analysis node**." DINDE prompts for the kind of node to be added (i.e., asks its class), which the user then selects from the toolbox. Suppose that the user selects the **BivariateRegression** class. Each instance of **BivariateRegression** incorporates a Y vector and an X vector. As part of the initialization for the new **BivariateRegression** object, the user is asked to identify a Y and an X. The user will typically respond by mousing the appropriate vectors already visible in the map, here designating BrainWts and BodyWts as Y and X, respectively.

At this point, the user is offered the opportunity to connect the newly created **BivariateRegression** object to existing nodes in the map (including none). In this example, we attach it to both existing vectors, resulting in the following map.



Here we might give the **BivariateRegression** a unique name and perhaps add some notes to it, possibly explaining why we are considering a bivariate regression analysis at all.

Selecting the **BivariateRegression** with the middle mouse button depressed produces the following menu of items.

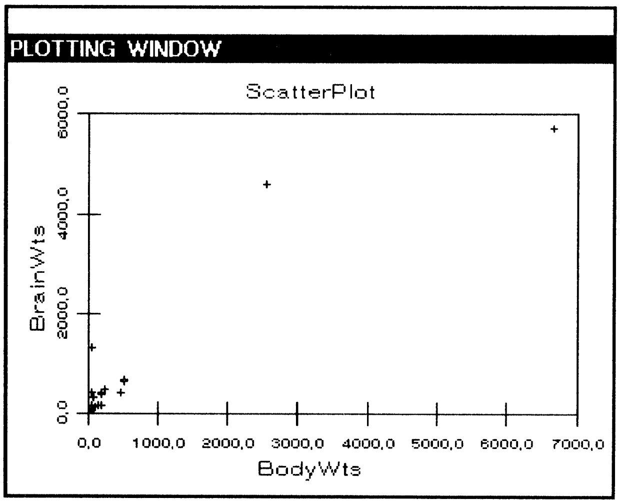


The leftmost stack of boxes provides access to two different collections of operations. **Class specific methods** represent those that are defined specifically for the class of **BivariateRegression** objects. **Other methods** are those which the **BivariateRegression** class has inherited from the classes that are its parents. We expect the user to examine **Class specific methods** first and thereby reveal the operations suited specially to the local context. Section 4.1, on the toolbox, discusses inheritance issues further. Sliding over the **Class specific methods** yields a menu where, given that we are interested in performing a bivariate regression, the next steps typically taken in the analysis are organized.

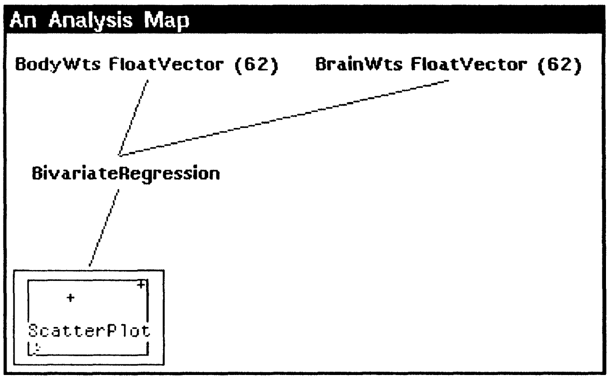
Selecting **Suggestions** from this menu will produce some general suggestions about how the user might perform a regression analysis. In this case, the suggestion is to visually inspect the data before doing anything else. More detailed suggestions, such as how to inspect the data, what fitting methods might be used and so on, are found by sliding the mouse over the **Suggestions** to get a more detailed menu. Note that these suggestions are data-independent: they remain the same regardless of the values attached to Y or X. (Data-dependent methods are possible as well, and are an open research question. See § 4.3 for further discussion.)

To inspect the data, the mouse is slid over **Inspect the data** to reveal the extended menu,

from which the **ScatterPlot** item is selected. The user is requested to use the mouse to size a window region for the plot and the scatterplot below is produced.

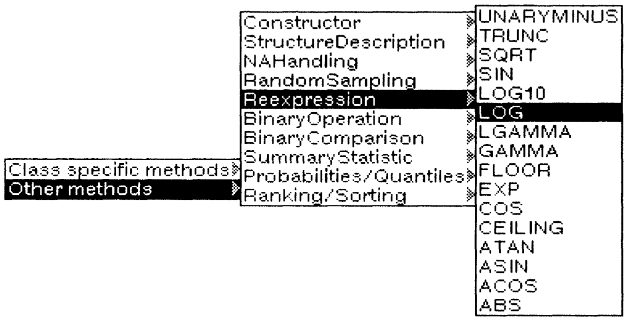


DINDE then attaches a **ScatterPlot** node to the **BivariateRegression** in the analysis map.



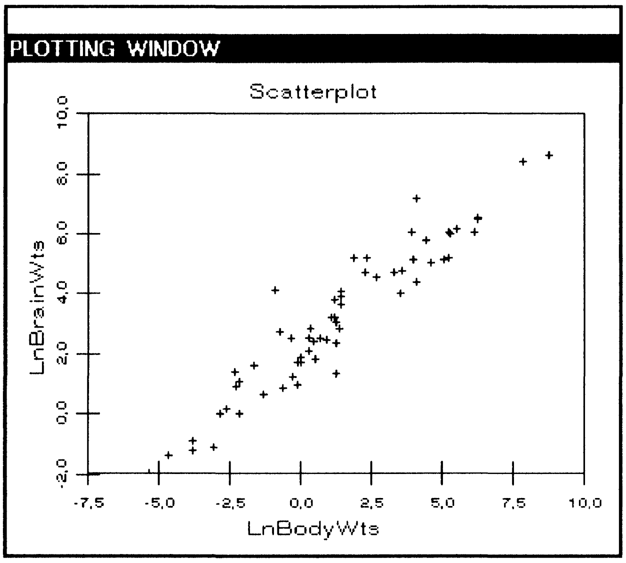
Two things have happened. First, the usual scatterplot has been produced for the user to view. Points that are selected may be deleted or labelled, axes may be scaled, and so on. (Jan Pedersen's code is used to display these plots.) Second, a **ScatterPlot** object was created and its presence noted in the map. This **ScatterPlot** can be named, can be documented and can respond to methods applicable to scatterplots. These methods are accessed by depressing the middle mouse button over the **ScatterPlot** object in the map or, alternatively, by depressing the same button in the title region of the plotting window. They include adding to the plot either an arbitrary straight line, the least-squares fitted line, a resistant line, or a running linear smooth.

At this point in the analysis, an experienced statistician would be uncomfortable about fitting a straight line to the data as they are. Points might be deleted from the plot to investigate those clumped in the left-hand corner, or various transformations might be applied to the data. For brevity, we note that taking the natural logarithm of Y and X works well on these data. Transforming to logs is accomplished by pointing in turn at each of the **FloatVectors** BrainWts and BodyWts, and pressing the middle mouse button. This gives us access to the re-expressions possible for **FloatVectors** through the following menu system.



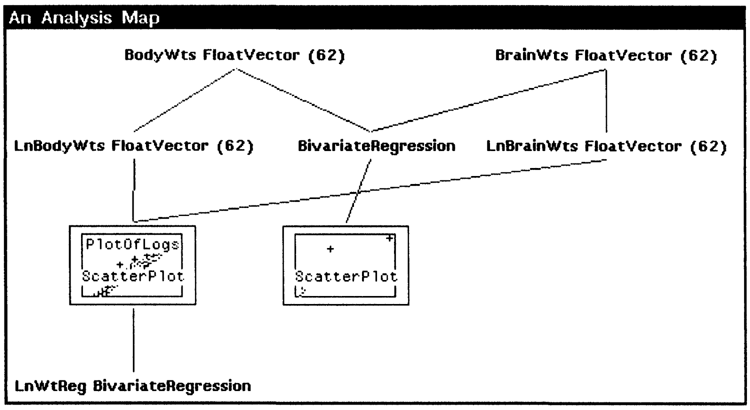
Selecting **LOG**, as indicated, spawns a new **FloatVector** from BrainWts, say, that has as elements the natural logarithm of the elements of BrainWts. This vector is automatically attached to BrainWts in the map.

We have done this for both BrainWts and BodyWts and have named the resulting **FloatVectors** LnBrainWts and LnBodyWts, respectively. The scatterplot of these two variables is the following.



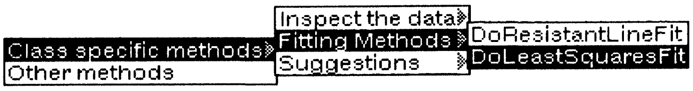
Now, we might consider fitting a straight line.

As before, we add an analysis node to the map, again choosing a **BivariateRegression** from the toolbox but now with Y and X identified as LnBrainWts and LnBodyWts, respectively. The analysis to date is summarized in the map given below.

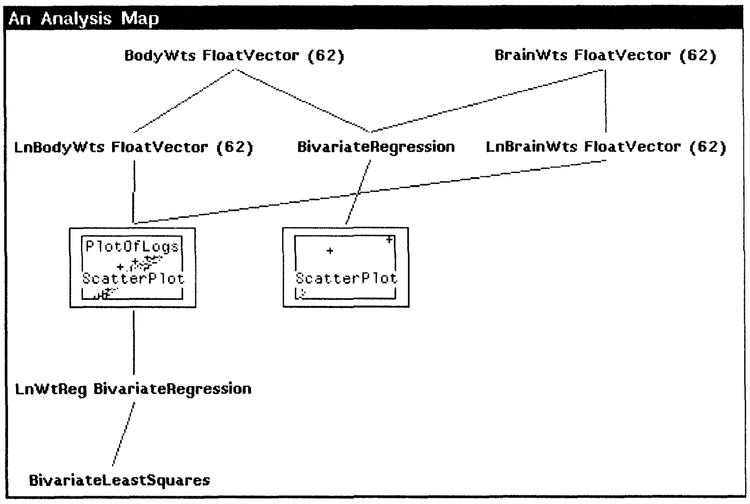

An Analysis Map
BodyWts FloatVector (62)
BrainWts FloatVector (62)
LnBodyWts FloatVector (62)
BivariateRegression
LnBrainWts FloatVector (62)
PlotOfLogs
ScatterPlot
ScatterPlot
LnWtReg BivariateRegression

Here, the **ScatterPlot** of the logged data and the new **BivariateRegression** have been given names. For the most part, the actual arrangement of the nodes and links is determined by the user. (It has been our experience that this arrangement is often a matter of personal style. Hence, it has been left to the user's discretion to produce a meaningful network description of the analysis.)

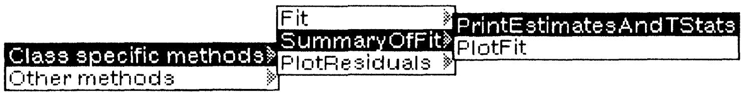
Pressing the middle mouse button on this new **BivariateRegression** produces the same extended menu as before. Having already looked at the relevant scatterplot, we slide over to select a fitting method and choose least-squares.



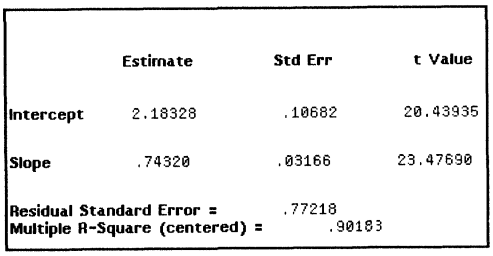
This causes a **BivariateLeastSquares** object to be produced and attached in the map to the **BivariateRegression** which spawned it.



Pressing the middle mouse button on this object produces a collection of methods which a **BivariateLeastSquares** can handle.



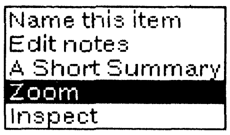
Selecting **PrintEstimatesAndTStats** causes the following table to be printed on the screen.



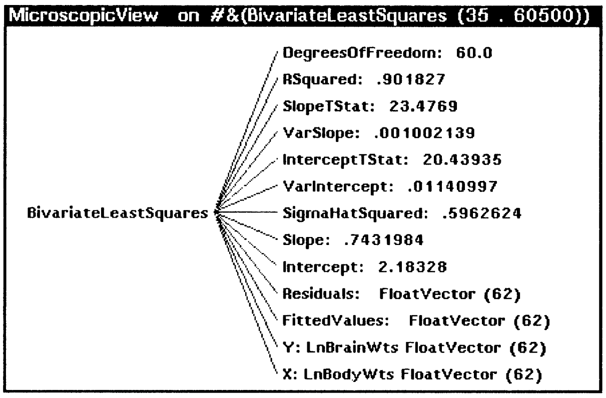
	Estimate	Std Err	t Value
Intercept	2.18328	.10682	20.43935
Slope	.74320	.03166	23.47690

```
Residual Standard Error =          .77218
Multiple R-Square (centered) =        .90183
```

Alternatively, these numbers and other details of this **BivariateLeastSquares** can be accessed by selecting it in the map with the left mouse button. This causes the following menu to pop up.



From this menu we select the item **Zoom**. (By the way, this is the menu that we have used throughout to name the various objects in the analysis.) This selection causes a new kind of DINDE window, a "Microscopic View" of the selected object, to be opened which displays the details of the object that has been zoomed-in on. For this **BivariateLeastSquares**, the new window looks like



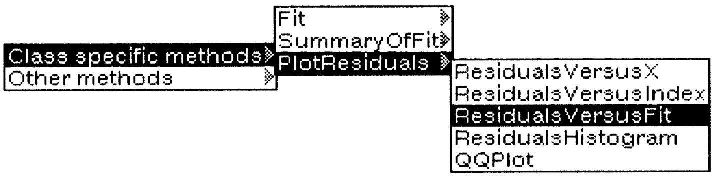
MicroscopicView on #&(BivariateLeastSquares (35 . 60500))

BivariateLeastSquares

DegreesOfFreedom: 60.0
RSquared: .901827
SlopeTStat: 23.4769
VarSlope: .001002139
InterceptTStat: 20.43935
VarIntercept: .01140997
SigmaHatSquared: .5962624
Slope: .7431984
Intercept: 2.18328
Residuals: FloatVector (62)
FittedValues: FloatVector (62)
Y: LnBrainWts FloatVector (62)
X: LnBodyWts FloatVector (62)

which is again a network of sorts. All DINDE objects can be zoomed in on to reveal their inner detail. An important point to note here is that each label in this network again represents an object which can be examined, named, or commented on and which responds to its own set of methods. For instance, we might now construct a **ScatterPlot** in the analysis which used the **Residuals** for its Y coordinates and the **FittedValues** for its X.

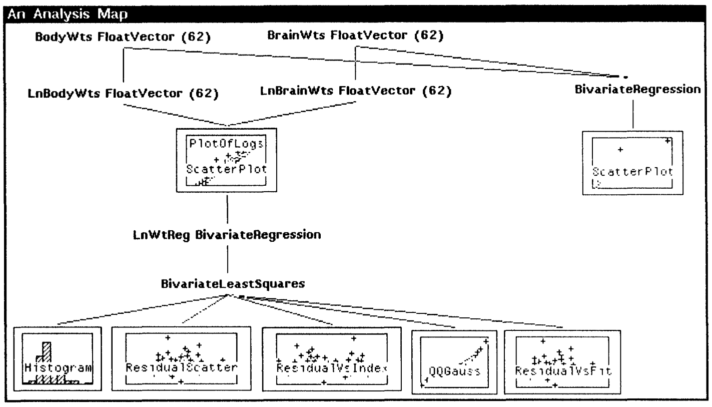
In the present case this is not necessary, since **BivariateLeastSquares** objects are equipped to produce such important plots easily. By getting the extended menu for this object we may produce a variety of residual plots.



Each of these produces its own kind of DINDE object having its own specialized capabilities. For example, the interpretation of a QQplot is quite different from an ordinary scatterplot and from these other residual plots. It therefore makes sense to separate the different kinds of plots.

Let us suppose that the analyst has elected to see each of the available residual plots and has explored them to his or her satisfaction. The analysis to date appears as the following map.

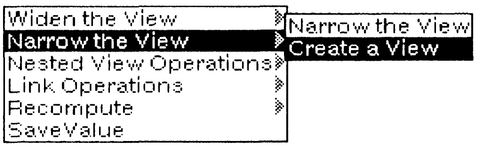
Note that the logical flow of this analysis can be easily seen in this display. For much larger analyses, the logic of the analysis will not always be so transparent. Therefore, a number of tools are made available to the analyst which allow information on the analysis to be added to its display.



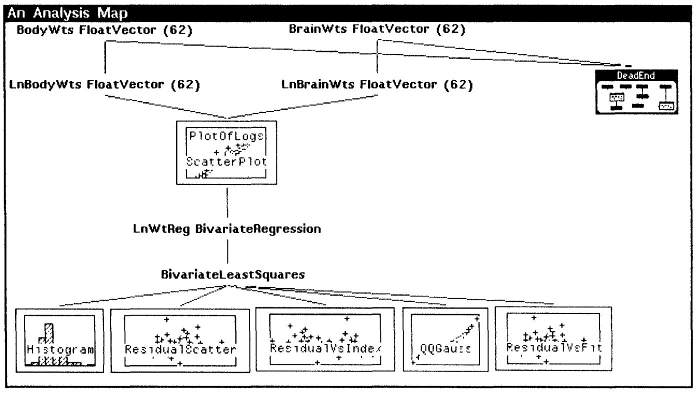
Minimally, the analyst may add notes to any node (to describe what the relevant features are at that node, why such and such a decision was taken, and so on). Additionally, DINDE objects called **Memos** (objects which contain nothing but commentary) can be usefully inserted as nodes at key decision points.

More direct methods exist which allow the analyst to alter the display so that it best reflects the logic of his or her analysis. These include the ability to make and break links between nodes (n.b., causal relationships between nodes cannot be rearranged in DINDE; see Oldford and Peters (1986b) for further details). In this way the network itself can be rearranged.

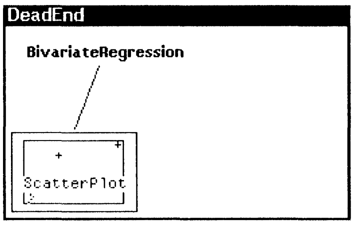
Perhaps the most important analysis management tool is the ability to compress parts of the analysis into smaller subanalyses. This is achieved by selecting the title bar of the analysis map with the middle button depressed and selecting the "Create a View" item from the resulting menu (as below).



This done, the analyst is prompted to identify those nodes which are to appear in the new analysis map. From the example, suppose that the analyst selects the two nodes representing the initial false start of the analysis, namely the first **BivariateRegression** and **Scatterplot** objects produced. The effect on the analysis can be seen below.



A small icon representing a different analysis map has appeared in the network at the location previously occupied by the **BivariateRegression** and **Scatterplot** objects. This new node can be interacted with in the same fashion as any other node. Indeed, the displayed node has already been named **DeadEnd** by the analyst. Zooming in on this icon produces the following analysis map.



This analysis map is identical in functionality to the larger one and, should the analyst choose to do so, the analysis may be continued in a more focused fashion from within this map. This facility enables the analyst to control the level of detail displayed and to break up the analysis into more manageable chunks. (Analysis maps are more fully discussed in Oldford and Peters (1986b).)

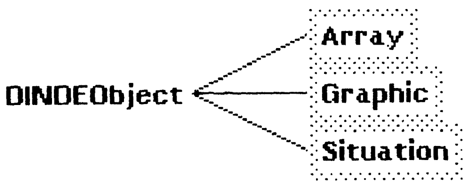
The user now has the entire analysis organized and displayed as a network (in fact a directed graph) having arbitrarily many levels of detail. The attached commentary, the ability to interact with a single node, and the visual flow of the analysis map should permit the analyst to reproduce the reasoning that was used, and the issues that were involved, in building the analysis. This has important consequences. Besides easing the production of a final report and allowing the user to profitably return to the analysis at another time, the analysis map affords the professional analyst the opportunity to visualize the overall structure of the analysis and hence more easily consider the statistical strategies that were employed.

4. Underlying ideas. In this section we describe the key ideas which underlie DINDE. The presentation has been divided into three subsections. In the first of these we describe the contents of the toolbox—that collection of basic DINDE classes which one may use to build an analysis. The way analyses are built and recorded in DINDE is treated in § 4.2. Section 4.3 discusses the manner in which we feel statistical expertise can be reasonably placed within a system like DINDE. The discussion here focuses on issues and philosophy of design; detailed documentation of the components appear elsewhere.

4.1. The toolbox. The toolbox contains those classes of objects which can be created and used in a statistical analysis. Since the analysis is represented as a collection of networks whose nodes are these objects, it is critically important that the classes represent meaningful chunks of information. This is perhaps one of the most challenging aspects of creating a sophisticated system like DINDE. It requires an identification and grouping of the elements that are important in a statistical analysis, a statistical taxonomy of sorts. To this end we suggest a coarse partition of such elements into six basic element types: (i) Data, (ii) Graphics, (iii) Situations, (iv) Models, (v) Tables and (vi) Designs.

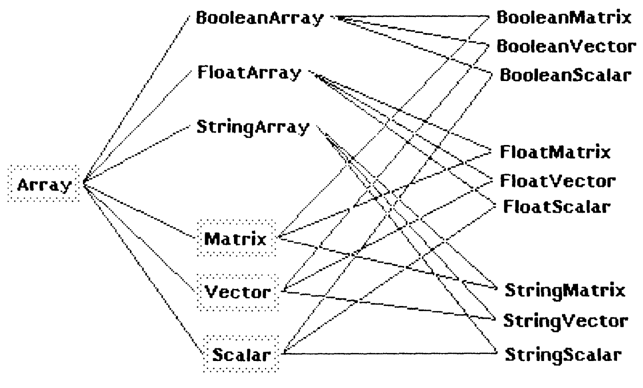
To date only the first three of these exist in DINDE. Only these were necessary to build a prototype bivariate regression analysis, but we anticipate that Models (such as parametric probability models), Tables (such as many-way contingency tables) and Designs (such as experimental and survey designs) will be required before long.

Object oriented programming is pervasive in DINDE and nowhere is this more evident to the user than in the toolbox. Here a network is displayed which shows both the classes of objects available and their relationship to one another. It begins with the most general class **DINDEObject**. The immediate children, or specializations, of this class are shown to the right of **DINDEObject** and attached to it by a line segment as below. (Far from serving as a simple place-holder, the class **DINDEObject** defines or "mixes in" a wide variety of important behaviors, among which are the ability to name and annotate objects, the basic linking operations and certain mouse responses. The descendants of **DINDEObject** for the most part inherit these behaviors "orthogonally," that is, without further specialization.)



The Data factor of our classification is represented here by the class **Array**. These classes have further specializations which appear in the toolbox. We discuss each in turn.

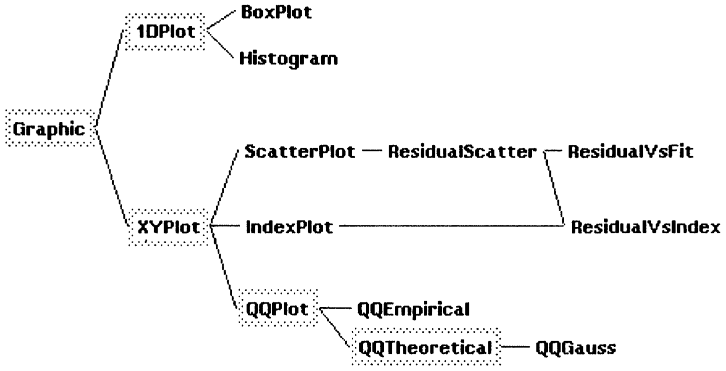
The class **Array** has three different specializations corresponding to the three modes for data values that we have adopted: floating point numbers, character strings and Boolean values. These subclasses are **BooleanArray** (an array of data which take on the values true, **T**, or false, **NIL**), **FloatArray** (an array containing floating point numbers) and **StringArray** (containing arbitrary character strings). **Array** is also specialized into the three familiar array shapes: **Matrix**, **Vector** and **Scalar**. Each of these in turn have specialized subclasses leading to the following network for the data.



The inheritance in this network runs left to right, from parent to child. It allows us to collect together those methods and variables which classes share and to attach them to a common ancestor. For instance, all **FloatScalars**, **FloatVectors** and **FloatMatrices** should be able to take the logarithm of their elements. Hence, the method **LOG** is defined for a **FloatArray** and is inherited down through the hierarchy by the others. It is not defined for **Array** because it makes no sense to try to take the log of either **BooleanArrays** or **StringArrays**. Similarly, **BooleanMatrix**, **FloatMatrix** and

StringMatrix are also all children of **Matrix**. This lets us attach strictly matrix attributes (like transpose) to the single class **Matrix**. (We now think that the **Array** hierarchy presents too much detail at too primitive a level. In a separate paper we describe new, higher-level data objects which address the special needs of statistics and data analysis (Oldford and Peters (1986c).)

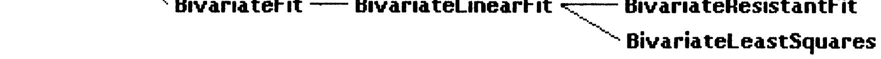
The network of classes below **Graphic** is the following.



At present, all DINDE graphics are based on simple one- and two-dimensional plots, as represented by **1DPlot** and **XYPlot**, respectively. For illustration, consider **XYPlot**. This graphic class has all the information necessary to produce a two-dimensional scatterplot of points. Information like the X and Y coordinates of the points, how to plot the points, how to label them and so on is collected here. **IndexPlots** are specialized **XYPlots** that automatically provide X coordinates which are the numbers 1 to N, indices for the components of Y. Similarly, **QQPlots** require both X and Y to be sorted. **ScatterPlots** are just those **XYPlots** to be used in a statistical analysis and hence have slightly more information attached to them, like how to add to the plot a least-squares fitted line, or some simple smooths.

This brings up an interesting point which was not encountered with the purely data constructs like **FloatVectors**. With **Graphics**, behaviors and information are now attached to a given class which are of interest only in a particular statistical context. Adding a smooth to a scatterplot is a statistical procedure useful for exploring the apparent dependence of one variable on another—it is not a useful adjunct to all possible **XYPlots**. The additional ability to smooth positive and negative Y's separately is a very useful device when the Y coordinates represent residuals; hence the further specialization to **ResidualScatter** plots. Specializations are thus created to sort out the pertinent statistical procedures and information. This results in having the tools most accessible when they are most needed.

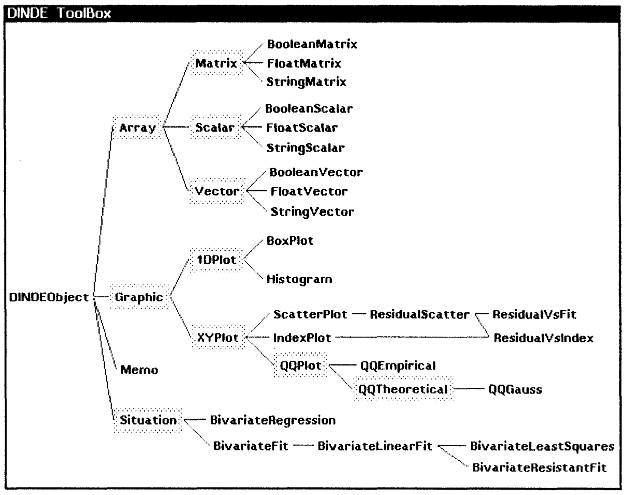
A straightforward extension of these ideas is to provide a grouping of statistical concepts, information and tools, which could be perceived as representative of some typical stage or decision point in an analysis. We have called such groupings **Situations**. While these are presently few in number, examination of those now available in DINDE should illustrate the idea. The figure below shows the **Situations** presently available in DINDE as they would appear in the toolbox.



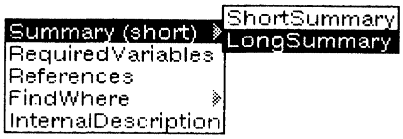
The **BivariateRegressionSituation** represents that point in the analysis where the analyst has decided to perform a bivariate regression of Y on X, and as such it must contain the minimal amount of information and set of tools required to make the next decision. It identifies the variables Y and X, contains some suggestions as to how to proceed and offers easy access to typical next steps (like plotting the points or fitting a straight line). If, at the **BivariateRegression** step, one elected to do a least-squares linear fit of Y to X, then a **BivariateLeastSquares** object would be produced. The **BivariateLeast-Squares Situation** is a specialization of a **BivariateLinearFit**, which is itself a specialization of **BivariateFit**. All **BivariateFits** have pointers to the X and Y vectors on which they are based and contain vectors of the fitted values and the residuals from the fit. Further, they can take a number of relevant actions such as producing a variety of residual plots. Additionally, **BivariateLinearFits** contain the parameter values which define the fitted line. **BivariateLeastSquares** contain yet more information such as variance estimates and t-statistics. This is distinguished from the **BivariateResistantLine**, representing the fit obtained by fitting the "resistant line" (see e.g., Velleman and Hoaglin (1981)), which has diagnostic information like the ratio of the half-slopes.

In **Situations** we see a need for much more work, both on those **Situations** we have created thus far and on new ones. **Situations** require the factorization, cataloguing and bundling of many statistical concepts, tools, techniques, etc., and the identification of the relationships between them (i.e., how one usefully leads to another). As such, the creation of each must be carefully undertaken. Their constituent parts must be based on sound statistical theory and practice, and their appropriate interconnection is often an open research question. We expect this to become ever more poignant as statistical situations more complex than simple bivariate regression are considered.

The complete collection of tools are made available to the user through a DINDE window which displays them in a network reflecting their familial relationships. This window, which we have been calling the toolbox, appears in its current entirety below. (For clarity, we have excluded the classes: **BooleanArray**, **FloatArray** and **StringArray**.)



The experienced statistical analyst should recognize all of the items in the toolbox, and should be able to locate the desired item by moving left to right through the network, from the general to the more specific, until the item is reached. In the event that it is not clear to the user what a particular class represents, or what it might be used for, various amounts of information on the class are readily available. The information is accessed simply by selecting the item with the left mouse button. This causes the following menu to pop up; from it the appropriate selection is made.



Short or long summaries of the class, a list of the components any instance of the class will require (e.g. **BivariateRegressions** must have a Y and an X variable) and references to the literature are all available if the user makes the corresponding selection. **FindWhere** identifies the parent from which any variable or method owned by that class was inherited, and **InternalDescription** gives a skeletal outline of the class in question showing its variables, methods and parents (super classes). (A possible extension to this menu would be a series of examples concerning the usage of instances of the class.) This kind of information is available for all classes in DINDE and is directly accessible from the toolbox.

4.2. Recording analyses. The toolbox represents a static component of DINDE and remains unchanged from analysis to analysis. The dynamic component lies in the construction and refinement of the analyses themselves. In this subsection we highlight the features of DINDE which allow the analyst to do just that. More detailed discussion can be found in Oldford and Peters (1986b).

A statistical analysis in DINDE consists of creating instances of one or more classes to produce DINDE objects that may then be joined together. Analysis maps are built in one of two ways. First, the user can select a class from the toolbox by attaching the resulting instance of that class anywhere in the map. Alternatively, the user could invoke a method from an existing node which might cause a new object to be created and attached to the selected node. The map always reflects the steps taken in the analysis so far and their logical interconnections. In addition, some partial time ordering of the nodes is implicit in the construction of the map. (All links are in fact directed links. Each node records the nodes to which it points "backward" and those to which it points "forward." Though not displayed as such, the maps are directed graphs.)

DINDE also lets the analyst make and break connections between nodes. In this way, time ordering may no longer be preserved. Inserting a **Memo** between nodes is a good example where time order might be sacrificed to yield a more understandable analysis. The only requirement of a network link is that it be meaningful to the analyst. Other links exist, possibly unknown to the user, which capture the legitimate causal relationships between DINDE objects. These cannot be made or broken by the user and are displayed in a different kind of DINDE window called a "Causal map" (see Oldford and Peters (1986b)).

For complex analyses the analysis map may become rather unwieldy. However, as analyses grow there will likely be segments or subsets of the map, submaps say, which can be thought of as representing small analyses in their own right. As was the case of the **DeadEnd** analysis of the example, it is helpful to encapsulate the smaller analysis as a single node in the map depicting the larger analysis. As a node in the larger analysis, it is able to respond to methods, can be named and can have commentary added to it just like any other node. **Zooming** on an **AnalysisMap** causes the corresponding analysis map to be opened, showing the details of the network it contains (which might in turn contain further **AnalysisMaps**, etc.). In this way, the analysis can contain many layers of detail, each one of interest at various stages in the analysis (see Oldford and Peters (1986a) for discussion). We anticipate that the number of layers will depend upon the particular statistical problem being undertaken. Therefore, no limit is placed on the number of layers allowed; it will depend upon the analyst to determine the layering appropriate for the problem.

Zooming is thus the vehicle used to descend deeper into the analysis map. At its bottommost level (the **MicroscopicView**) it displays the variable names and their values (other DINDE objects) and permits the same kind of interaction with these variables as would be allowed in an analysis map. This was demonstrated in § 3 on a **BivariateLeastSquares** object. Ascending back up through the analysis is no problem. Any number of maps/levels are displayed on the screen at one time. Windows corresponding to maps need only be shrunk or closed to remove them from display (shrinking replaces the map with a small icon indicating that it is a map, and giving its name if any). The subtleties of managing system resources when there are many windows on the screen is beyond the scope of our discussion. However, we note that this issue has been effectively addressed in the Boxer (Abelson and DiSessa (1986) and Notecards (Xerox (1985)) systems.

4.3. Guidance. It is our intention that the user be given free rein in the construction of the analysis. This is in keeping with having a professional statistician as the target user. Nevertheless, DINDE is constructed so as to inconspicuously guide the analysis where possible.

Two different approaches to this task can be taken. In one, the guidance is strictly independent of the data at hand. A metaphor might be the sort of guidance given over the telephone by an experienced statistician to a less experienced one. The advice-giving statistician has no access at all to the data, not even to the subject-matter context from which the data arose. The less experienced statistician only wants to know, for example, how one should perform a regression analysis, what to look for, and so on. (Argument for the credibility of this metaphor can be found in Oldford and Peters (1986a).)

In DINDE, we mimic this kind of guidance through the selection and design of classes, in particular the **Situation** classes. Each class represents a stage or element in the overall analysis. Hence, the choice of classes in DINDE represents our cataloguing of the statistical procedures and concepts we regard to be relevant to different kinds of analyses (**BivariateRegression**, **BivariateLeastSquares**, **UnivariateTimeSeriesAnalysis**, etc.). The organization of the methods that a particular class of object responds to indicates those steps that are generally taken next in the analysis. For example, a **BivariateLeastSquares** object offers a selection of methods which each produce a useful residual plot. In this way, what appears to be merely convenient for the experienced analyst turns out to be guidance for the less experienced.

This data-independent channelling of the analysis is made more explicit by adding **Suggestions** to each **DINDEObject**. In this way, some text may be given which organizes the alternatives according to various characteristics of the data and analysis to date. The onus is then on the user to determine how the characteristics of the current data and analysis match those outlined in the **Suggestions**. Self-explanatory examples would also be helpful in this regard.

The second approach is the data-dependent one, whereby suggestions made to the user are based upon having the system investigate, and give some interpretation to, the properties of the data in hand. Bivariate regression analysis in REX is an important example of this approach (see Pregibon and Gale (1984)).

In DINDE, we plan to adopt this approach in an inconspicuous and quite local manner. It is local in two senses. First, such data checking operations would be attached to particular DINDE classes and would be available to the user as a method of that node type in the analysis. Second, the method itself would have access to only two sources of information—either that attached to the object, or that to be had by querying the user. In particular, no information regarding the analysis map or history would be used. In this way, no attempt is made to provide the user with a global analysis—all suggestions are local and therefore the analyst may justifiably choose to ignore them in light of the rest of the analysis.

For instance, a **BivariateRegression** might have a method called **CheckFunctional-Form**, which would try to determine whether different transformations of the X and Y data might be better suited to a linear regression analysis, and provide evidence for any comments made. (Hopefully, the logarithmic transformation would be suggested in the example.) The user would then be free to follow the suggestions or not.

While both approaches have merit, we feel that data-independent channelling is more easily implemented and provides much of the guidance that can be competently given (for further discussion, see Oldford and Peters (1986a)). In any case, it seems to be the natural precursor to the data-dependent form of guidance.

5. DINDE and statistical strategy. In Oldford and Peters (1986a), we argue that complete software representations are not likely to be constructed for many of the analysis strategies used by practicing statisticians. Statistical strategies that show the most promise are those that both depend little upon correctly interpreting the subject-matter context surrounding the problem, and are applicable to only a relatively small number of problems.

Strategies that most easily fit this description are those we call low-level low-context strategies; an example is the strategy employed to assess the number of near dependencies in a regressor matrix. "Low-level" means that the strategy typically appears as a low-level operation in some wider analysis. "Low-context" indicates that the strategy relies little upon the subject-matter context for its correct application. Typically, these strategies are applied in a narrow domain of well-defined problems, yet depend on some judgment in their application.

Representations of these kinds of strategies appear in DINDE as methods of the objects for which they are most relevant. If, for instance, a particular strategy to check the correctness of the functional form in a bivariate regression is desired, then it would appear on the **BivariateRegression** class as the method **CheckFunctionalForm**. The method could invoke a procedure or create a more detailed object that contained the necessary tools and information to go about applying the strategy.

Higher-level strategies are of more interest; consider, for example, the strategy employed by an experienced statistician using multiple regression tools to describe the relationships between two sets of variables. At this level, effective strategies typically depend critically upon correct interpretation of the background context, making it difficult to construct complete representations for them. Therefore, we propose that the context-dependent elements of the strategy be removed; the telephone conversation between the experienced and inexperienced statisticians provides a simple metaphor for what might remain (see § 4.3).

To achieve this, the strategy is first broken into its constituent parts (Pregibon (1985) has outlined rules for doing this). These components could be connected together and provided to the user as a single overall strategy, as in REX for bivariate regression (Pregibon and Gale (1984)), but we prefer that they be made directly available to the user and that the connections only be pointed out as often being appropriate. The decision to consider a component or make a connection is then left entirely to the user: at this level of strategy the analyst is more competent at interpreting the relevance of the various components than is a software representation.

In DINDE, these component pieces are represented as distinct classes. Interconnections are indicated by the methods attached to the class—by their presence and by their application. Data-independent **Suggestions** describe a strategy to choose the next component; data-dependent analogues would be implementations of some of these small strategies. The example in § 3 shows some of the components and interconnections of a strategy for bivariate regression analysis.

For high-level strategy, visually presenting the parts and connections makes these aspects of the strategy apparent. The analyst sees the analysis develop, parts of it that are according to the canned high-level strategy, and more important, parts of it that are according to the analyst's personal strategy. The analyst may then be tempted to think more about the strategy he or she employs. (Various network operations, **AnalysisPaths**, and user-added commentary aid the development of the latter.) Further, by collapsing pieces of the network into **SubMaps** that can be named and that can have user-added commentary the analyst can define the levels of abstraction appropriate for the problem and strategy. These various levels of abstraction help the analyst to both better manage the analysis and better illuminate the personal strategy being used. The analysis, then, is seen to be an intertwining of two strategies, that of the analyst and that of the developer.

6. Concluding remarks. At present DINDE exists as a vehicle for research into statistical system design and questions of statistical strategy. As yet we have only produced the statistical tools necessary to perform a bivariate regression. Incorporating other and more complex statistical situations requires new and different research in applied statistics and statistical computing. Our purposes here are to outline some of the design considerations involved in pursuing such research and to encourage others to join in the endeavour.

This paper has explored several techniques which place greater responsibility on the computing environment for support of an evolving data analysis. From the outset, managing the complexity of the analysis has been a primary concern for our design. In summary we think there are three aspects. The first involves the selection and application of statistical methodology; at present DINDE offers a limited choice of techniques. Yet we believe our design will scale up gracefully as more, and more complex, techniques are installed. Next, experimentation and improvisation with new methods are simplified in the Loops environment. The user interface and all that lies behind it are expressed in a single language and exploit a common set of programming tools. Last, we offer a model (based on acyclic graphs) for the *process* of data analysis. We think that the subsequent discipline (attaching nodes, zooming, shrinking, partitioning the graph into subanalyses, etc.) eases the analyst's burden in organizing the analysis.

Finally, it is our intention to make DINDE code available to other researchers in these areas: the code is easily extended using the tools available in LOOPS and the Interlisp-D environment. We also watch with interest the development of CommonLisp (Steele (1984)) for workstations. Some CommonLisp implementations allow incorporation of compiled Fortran algorithms, an obvious requirement for a DINDE production system. Further, we believe that much of what has been done could be ported to other object oriented environments (e.g., Smalltalk, or CommonLoops built on CommonLisp) with minimal difficulty.

REFERENCES

H. Abelson and A. DiSessa (1986), *Boxer: a reconstructible computational medium*, Comm. ACM, 29, pp. 859-868.

R. A. Becker and J. M. Chambers (1984), *S: An Interactive Environment for Data Analysis*, Wadsworth Press, Belmont, CA.

D. G. Bobrow and M. Stefik (1983), *The LOOPS Manual*, Xerox PARC, Palo Alto, CA.

D. B. Carr, P. J. Cowley, M. A. Whiting and W. L. Nicholson (1984), *Organizational tools for data analysis environments*, Proc. American Statistical Association: Statistical Computing Section, Washington, D.C., pp. 214-218.

D. L. Donoho (1983), DART: *A tool for research in data analysis*, Ph.D. thesis (unpublished), Dept. of Statistics, Harvard University, Cambridge, MA.

A. Goldberg and D. Robson (1983), *Smalltalk*-80: *The language and its implementation*, Addison-Wesley, Reading, MA.

P. J. Huber (1985), *Environments for supporting statistical strategy*, in Artificial Intelligence and Statistics, W. A. Gale, ed., Addison-Wesley, Reading, MA, pp. 285-294.

Interlisp (1983), *Interlisp Reference Manual*, Xerox PARC, Palo Alto, CA.

R. M. Kaplan, B. A. Sheil and E. R. Smith (1981), *The Interactive Data-analysis Language Reference Manual*, Xerox PARC, Palo Alto, CA.

J. McDonald and J. Pedersen (1985a), *Computing Environments for Data Analysis* I. *Introduction*, this Journal, 6, pp. 1004-1012.

——— (1985b), *Computing Environments for Data Analysis* II. *Hardware*, this Journal, 6, pp. 1013-1021.

——— (1988), *Computing Environments for Data Analysis* III. *Programming Environments*, this Journal, to appear.

W. L. Nicholson, D. B. Carr, P. J. Cowley and M. A. Whiting (1984), *The role of environments in managing data analysis*, Proc. American Statistical Association, Statistical Computing Section, Washington, D.C., pp. 80-84.

R. W. Oldford and S. C. Peters (1986a), *Implementation and study of statistical strategy*, in Artificial Intelligence and Statistics, W. A. Gale, ed., Addison-Wesley, Reading, MA, pp. 335-353.

——— (1986b), *Data analysis networks in* DINDE, in Proc. American Statistical Association, Statistical Computing Section (plus videotape), Washington, D.C., pp. 19-24.

——— (1986c), *Object-oriented data representations for statistical data analysis*, COMPSTAT-86, Rome, Italy, Physica-Verlag, Heidelberg.

D. Pregibon (1985), *A* DIY *guide to statistical strategy*, in Artificial Intelligence and Statistics, W. A. Gale, ed., Addison-Wesley, Reading, MA, pp. 389-400.

D. Pregibon and W. P. Gale (1984), REX: *an expert system for regression analysis*, COMPSTAT-84, Prague, Physica-Verlag, Vienna, pp. 242-248.

B. Sheil (1983), *Power tools for programmers*, Datamation, February, pp. 131-143.

G. Steele (1984), *Common Lisp*, Digital, Billerica, MA.

M. Stefik, D. G. Bobrow, S. Mittal and L. Conway (1983), *Knowledge programming in* LOOPS: *report on an experimental course*, The AI Magazine, 3, pp. 3-13.

W. Teitelman and L. M. Masinter (1981), *The Interlisp programming environment*, IEEE Trans. Comput., 14, pp. 25-34.

R. A. Thisted (1985), *Representing statistical knowledge and search strategies for expert data analysis systems*, in Artificial Intelligence and Statistics, W. A. Gale, ed., Addison-Wesley, Reading, MA, pp. 267-284.

P. Velleman and D. Hoaglin (1981), *Applications, Basics and Computing of Exploratory Data Analysis*, Duxbury Press, Boston, MA.

D. Weinreb and D. Moon (1981), *Lisp Machine Manual* (*Fourth edition*), MIT Artificial Intelligence Laboratory, Cambridge, MA.

S. Weisberg (1980), *Applied Linear Regression*, John Wiley, New York.

Xerox (1985), *NoteCards Release* 1.2*i Reference Manual*, Xerox Special Information Systems, Pasadena, CA.