Title
WES, A WELL TEST ANALYSIS EXPERT SYSTEM

Author
Mensch, A.

Publication Date
1988-06-01

LBL-25523
UC-11

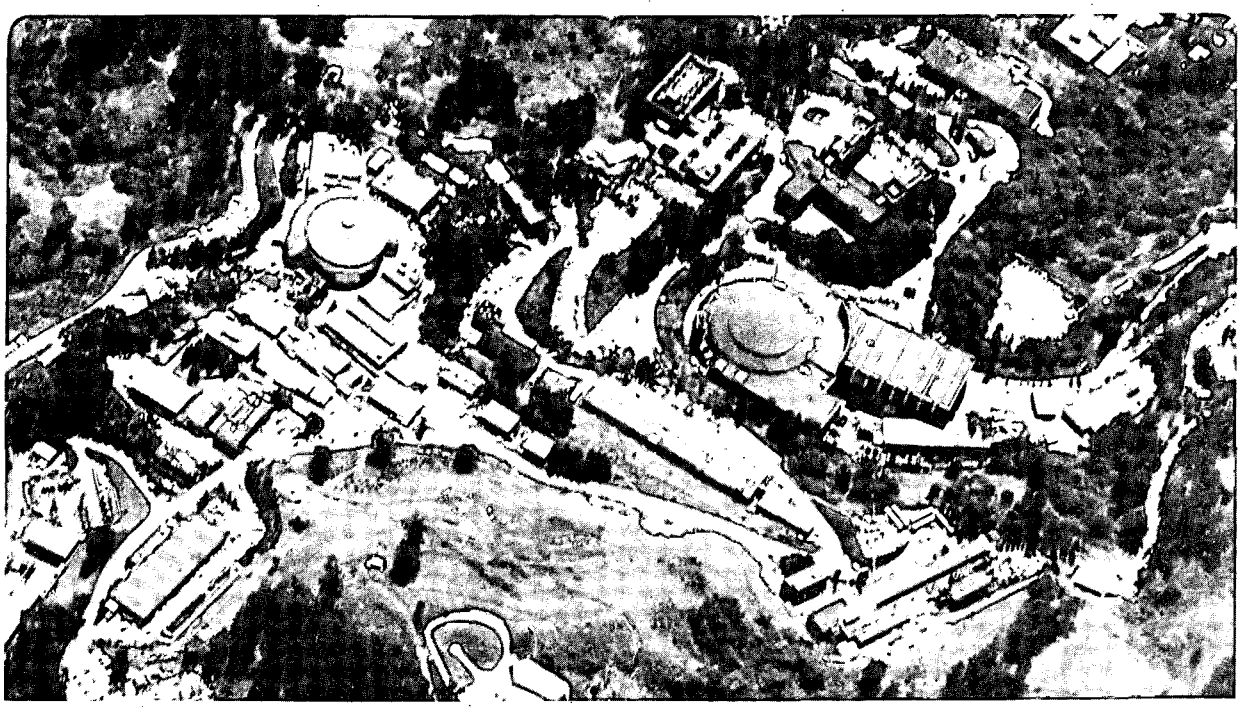
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA, BERKELEY

Information and Computing Sciences Division

WES, a Well Test Analysis Expert System

A. Mensch

June 1988



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

LBL-25523

WES, a Well Test Analysis Expert System

Antoine Mensch

Information and Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, California 94720
USA

June 1988

This research was supported by the Repository and Technology Program of the Office of Civilian Radioactive Waste Management of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

REFERENCES

1. Clark, D.G. and Van Golf-Racht, T.D. (1985). Pressure-Derivative Approach to Transient Test Analysis: A High-Permeability North Sea Reservoir Example, *Journal of Petroleum Technology*, November 1985, 2023-2039.
2. Bourdet, D. et al (1983). A New Set of Type Curves Simplifies Well Test Analysis, *World Oil*, May 1983, 95-106.
3. Bourdet, D., Ayoub, J.A., and Pirard, Y.M. (1984). Use of Pressure Derivative in Well Test Interpretation, paper SPE 12777 presented at the 1984 SPE California Regional Meeting, Long Beach, April 11-13.
4. Bourdet, D., Ayoub, J.A., and A. Alagoa (1985). How To Simplify The Analysis of Fractured Well Tests, *World Oil*, October 1985, 98-101.
5. Warren, J.E. and Root, P.J. (1963). The Behavior of Naturally Fractured Reservoirs, *Journal of Petroleum Technology*, September 1963, 245-255.
6. Agarwal, R.G., Al-Hussainy, R., and Ramey, H.J.(1970). An Investigation of Wellbore Storage and Skin Effect in Unsteady Liquid Flow: I. Analytical Treatment, *Journal of Petroleum Technology*, 1970.
7. Gringarten, A.C. (1984). Interpretation of Tests in Fissured and Multilayered Reservoirs With Double Porosity Behavior: Theory and Practice, *Journal of Petroleum Technology*, April 1984, 549-563.
8. Vongvuthipornchai, S. and Raghavan, R. (1988). A Note on the Duration of the Transitional Period of Responses Influenced by Wellbore Storage and Skin, *SPE Formation Evaluation*, March 1988, 207-214.

9. Onur, M. and Reynolds, A.C. (1988). A New Approach for Constructing Type Curves for Well Test Analysis, *SPE Formation Evaluation*, March 1988, 197-206.
10. Barua, J., Horne, R.N., Greenstadt, J.L., and Lopez, L. (1988). Improved Estimation Algorithms for Automated Type-Curve Analysis of Well Tests, *SPE Formation Evaluation*, March 1988, 186-196.
11. Tiab, D. and Puthigai, S.K. (1988). Pressure-Derivative Type Curves for Vertically Fractured Wells, *SPE Formation Evaluation*, March 1988, 156-158.
12. Stehfest, H. (1970). Numerical Inversion of Laplace Transforms, *Communications of the ACM*, January 1970, 47-49.

ACKNOWLEDGEMENTS

First, I would like to thank Esther Schroeder and Daniel Billaux for their advice and guidance throughout this work. They have also helped me a great deal during my acclimatation period in the lab and more generally in Berkeley. Without their support, this period would not have been as pleasant and instructive as it has been.

I would like to thank Sally Benson for her help in the field of well-test analysis. Although she was very busy during this period, she always gave me the informations or references I needed.

Craig Eades and Les Tabata have helped me use the different computers and devices I needed during this period. They were always prepared to help me with technical problems.

I would like to express my special thanks to Elinor Sigler, English teacher at the Ecole Centrale de Paris. She helped me to keep in touch with the school and gave me the administrative informations I needed.

This work was supported by the Repository and Technology Program of the Office of Civilian Radioactive Waste Management of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

INTRODUCTION

This report describes part of the development of an expert system in the domain of well-test analysis. This work has been done during my final internship, completed at the Lawrence Berkeley Laboratory between March 29 and June 20. This internship is the beginning of a longer period of at least six months (and probably two years). Therefore the work described in this report must be considered as part of a more important project, and not at all as a whole entity in itself.

This project was six months old when I first began to work on it. Therefore the first few weeks were employed to master the existing system and the expert system shell it uses. During the same time, some bibliographical researches were done about well-test analysis, which was a new subject for me too.

The report is divided in three parts: the first one gives a description of the state of the project at the time I first began to work on it, and raises some problems that have to be solved. The second section shows the results that have been reached, and the last one draws conclusions from these results and proposes extensions that would be useful in the future.

I would like to apologize for the poor writing style used in this report: I have tried to make my English as understandable as possible, therefore repetitions and gallicisms often appear.

PART I - DESCRIPTION OF THE PROJECT

1.1. WELL TEST ANALYSIS

1.1.1. THEORY

1.1.1.1. Introduction

This chapter exposes the main articulations of the reasoning in well test interpretation. This could not be considered as the theory of well test analysis (which is beyond the scope of this report), but rather as a simplification of this theory that enhances the applicability of expert systems in that field. This part is the summary of a few papers[1 to 11] published in specialized publications ("Journal of Petroleum Technology", "World Oil") between 1984 and 1988. It includes the description of the pressure-derivative approach, the most recently developed method[2,3,4] (1984) of well test analysis.

The interpretation of transient test data is the main source of information on the hydraulic characteristics and the dynamic behavior of a reservoir, especially if permeability is high, because a test can investigate an appreciable volume of the formation. During such tests, the hydrogeologic regime is perturbed by pumping liquid, water or oil, in or out of a well, and the response of the underground flow system to the perturbation is generally monitored over a long length of time, yielding curves of pressure versus time.

When an expert works on the interpretation of pumping tests, he looks for unique characteristics, or "signatures" on one or several representations of the pressure-time data. Classical representations involve semilog and log-log curves, and a newer method uses a pressure-derivative curve. Well-known types of hydrogeologic systems have differents signatures on these curves, such as straight lines, humps, etc... The task of the expert is therefore to draw the curves, recognize those features and draw conclusions from them.

Conclusions consist of both a conceptual model, which describes the nature of the ground water flow system, and an estimate of the parameters needed to properly characterize this flow system. In this chapter, only a limited set of conceptual models will be considered: two general types of reservoir (homogeneous and double porosity), two different inner boundaries (wellbore storage effect and fractured well effect), and three outer boundaries (no boundary, no-flow boundary, pressure-maintenance boundary).

1.1.1.2. Traditional And Derivative Theories

In recent years, the science of transient well test interpretation has progressed rapidly, most notably through the increased use of type curves, the introduction of new reservoir models and the advent of computer interpretation packages. The major improvement still is the introduction of the pressure-derivative analysis method, which deals directly with rate of pressure change, rather than absolute pressures. This new method has been allowed by the advent of new electronic bottomhole pressure gauges, which has dramatically improved data quality.

Practical transient test interpretation methods have become polarized in recent years between two analysis techniques - conventional and global. The former basically consists of fitting straight lines to data regions, for example on the semilog plot (Horner analysis). The latter involves the use of various type curves to include the entire data set in the process of reservoir system diagnosis, flow regime identification, and evaluation of system parameters. It is commonly accepted that great confidence in interpreted results is obtained by an iterative combination of the two techniques that starts with the global approach.

The recently developed pressure-derivative approach has combined the most powerful aspects of the two previously separate methods into a single-stage interpretative plot.

Both methods (traditional and derivative) are described in this part for the most common model of a well with wellbore storage and skin in a homogeneous reservoir[1,6,8] (the possibility of comparing results obtained by two different approaches could be a very useful tool for an expert system, therefore both methods are kept).

The most useful traditional type curves depict, on a log-log scale, the evolution of the dimensionless pressure, p_D, with the dimensionless time group t_D/C_D (C_D is the dimensionless wellbore storage constant). For the basic model considered here, the individual curves are dependant on the wellbore condition group $C_D e^{2S}$ (S is the skin factor).

These curves have two main drawbacks:

- first, there is the uniqueness of the diagnosis. The various curves have similar shapes, particularly when the effect of wellbore storage is very short-lived. The main regime of interest, for the evaluation of reservoir parameters, infinite-acting radial flow, has no characteristic shape on a log-log plot.
- second, late- and intermediate-time deviations from the diagnosed trend (i.e. outer boundary effects) are compressed to the extent that recognition is unlikely.

Both necessitate the supplementary use of semilog scale plots (conventional analysis) to obtain more accurate results and to help recognize and improve evaluation of nonhomogeneous behavior.

In dimensionless terms, infinite-acting radial flow is conventionally written as

$$p_D = 0.5\ (ln\ (t_D/C_D) + 0.80907 + ln\ (C_D e^{2S})).$$

This is the "semilog approximation" and is valid only after the wellbore storage effect has become negligible. This approximation is represented by a straight line on the semilog plot, but, as explained above, has no characteristic shape on the log-log plot.

When fluid movement is confined entirely to expansion or compression in the wellbore, pure wellbore storage is given by

$$p_D = t_D/C_D,$$

and is represented by a unit-slope straight line on the log-log plot.

In the pressure-derivative approach, the semilog slope of the dimensionless pressure response is plotted on a log-log plot in place of the dimensionless pressure. The y axis is the derivative of pressure with respect to the natural log of time:

$$dp_D/dln(t_D/C_D) = (t_D/C_D)p'_D \qquad (p'_D = dp_D/d(t_D/C_D)).$$

When applied to the infinite-acting radial flow equation, for which the slope is constant on a semilog scale, the equation becomes

$$(t_D/C_D)p'_D = 0.5,$$

a more easily diagnosed horizontal line on a log-log plot.

Pure wellbore storage becomes

$$(t_D/C_D)p'_D = t_D/C_D,$$

this again is a unit-slope line on log-log paper, as was the underived form. Hence, the endpoints of all the curves are fixed by two common asymptotes with a hump-shaped transition whose shape is controlled by the wellbore condition group C_De^{2S}.

The type curves for this model are shown Figure 1-1.

Several types of heterogeneities commonly are characterized by either middle-time or late-time deviations from infinite-acting radial flow, represented by a straight line on semilog paper and a horizontal line on the log/time derivative form. In general terms, the pressure-derivative method can be considered as a normalization of a semilog plot. In doing this, other characteristic behaviors that often are not easily discernable by traditional methods are accented,

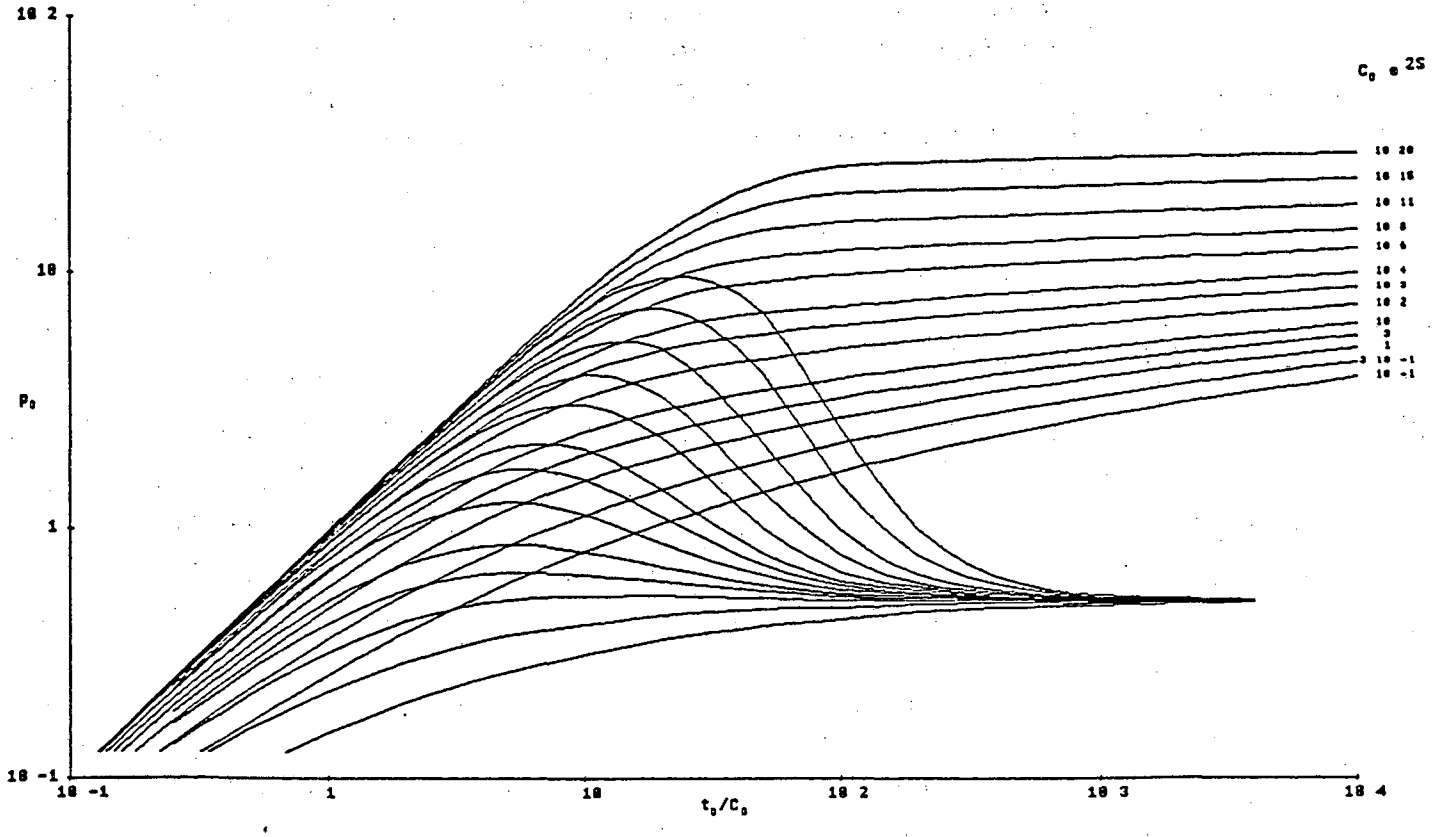


Figure 1-1: Combined derivative and pressure type curves.

facilitating model diagnosis, flow regime identification and parameter evaluation.

One of the most useful characteristics of the pressure-derivative approach is the correspondence of the straight lines on the derivative plot to the straight lines on the traditional plots:

- A straight line of slope m on the pressure plot is represented by another straight line of similar slope on the derivative plot (when both plots are on log-log paper).
- A straight line on the semilog plot of the pressure is represented by a horizontal straight line on the log-log plot of the pressure derivative.

1.1.1.3. Well System Analysis

1.1.1.3.1. DRAWDOWN

This part describes the most commonly observed well system responses for the case of a drawdown from initial conditions (a drawdown consists of pumping liquid out of a well with constant flow rate). As will be discussed later, the responses obtained during buildup can be more complex and can even alter in trend

HOMOGENEOUS RESERVOIRS

A homogeneous reservoir is a well in which reservoir properties can be represented by a single system model. Departure from this infinite-acting radial-flow model are caused by inner and outer boundary conditions. The most common case (Col.1 in Figure 1-2), that of wellbore storage and skin (inner boundary effect), was described above.

- Hydraulically fractured wells[4,11] (Col. 3 in figure 1-2). This is another inner boundary effect. Unless obscured by wellbore storage, the half-slope line characteristic of fracture linear flow initially is observed on a traditional log-log plot. In dimensionless terms, this is given by

$$p_D = (\pi t_{Df})^{1/2}.$$

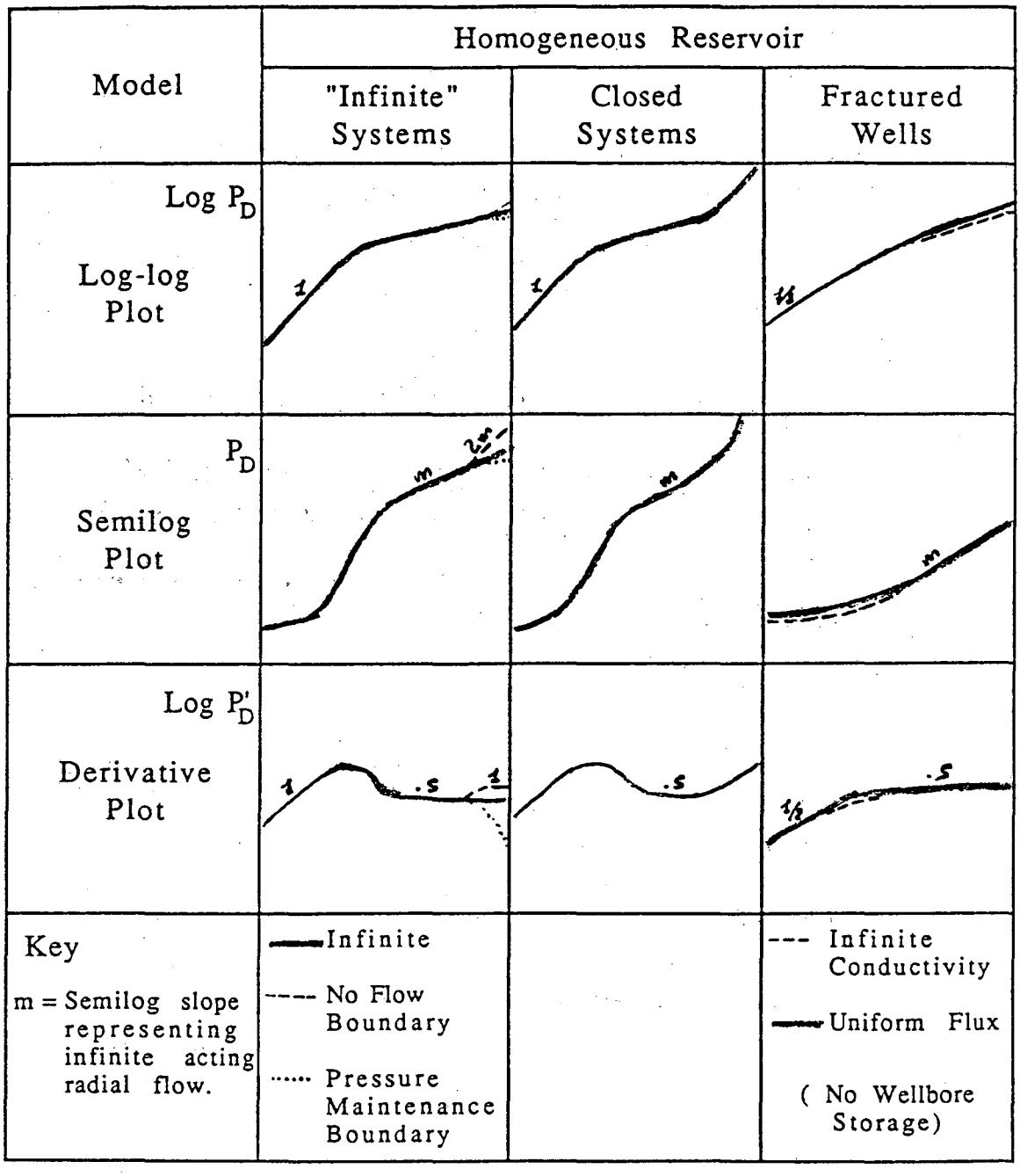


Figure 1-2: Summary of homogeneous reservoir model responses.

When differentiated with respect to the log of time function, this becomes

$$t_{Df}p'_D = 0.5(\pi t_{Df})^{1/2},$$

again giving a half-slope line on log-log.

In radial flow, a difference between two types of fracture appears: the pressure response is approximated by

$$pD = 0.5\ (ln(t_{Df}) + 2.2) \text{ for infinite conductivity}$$

and

$$pD = 0.5\ (ln(t_{Df}) + 2.81) \text{ for uniform flux.}$$

The difference in the constant term manifests itself on log-log paper by the stabilization of the final curves at different levels.

On the derivative plot, radial flow again is represented by a horizontal straight line:

$$t_{Df}p'_D = 0.5.$$

The two models differ only in the transitional period and give different curves that are analogous to different skin ($C_D e^{2S}$) on the wellbore storage and skin type curves.

- Bounded systems (Col. 1 in figure 1-2). The two most common outer boundaries are barriers to flow, such as faults and pinchouts, and pressure maintenance from a gas cap or aquifer. Neither of these has an easily observable characteristic form on a log-log scale. A semilog plot normally is used for conventional or type-curve analysis.

A single no-flow boundary results in the establishment of a semiradial flow regime with an increased rate of pressure change. On semilog plots this produces a doubling of the slope. Consequently,

after a transition period, the boundary appears on the derivative plot as a horizontal line:

$$(t_D/C_D)p'_D = 1.$$

A further no-flow boundary can double this to 2.

Pressure-maintenance boundaries, on the other hand, result in a reduction in the pressure changes at late times. This manifests itself as a flattening on the semilog plot as the maximum pressure differential is attained; on the derivative plot, the curve slopes downward to zero.

In a closed system (Col. 2 in figure 1-2), the late time pseudosteady-state response can be written as:

$$p_D = a(t_D/C_D) + b,$$

where a and b are constants dependent on reservoir size, shape, and properties. On the log-log plot, the curve tends asymptotically to a unit slope.

When the derivative with respect to the log of time is taken, the second constant is lost:

$$(t_D/C_D)p'_D = a(t_D/C_D),$$

which gives a unit-slope straight line on a log-log scale.

HETEROGENEOUS RESERVOIRS

There are many reservoirs where the pressure response is the product of the interplay of more than one conductive medium. Most current models are composed of two homogeneous media dispersed throughout the reservoir, with a large permeability contrast between them[5,7].

The models currently used most extensively are the two considered for double-porosity systems. Here the observed response is the result of two media, usually considered as fissures and blocks

in naturally fractured reservoirs. For simplicity, this terminology is often extended to represent the high- and low-permeability layers, respectively, in multilayered reservoirs. Initially, flow is almost entirely from the high-permeability, low-storativity fissure system. Eventually, there is pressure support from the high-storativity block system before the two systems finally stabilize; the subsequent response is that of the total system. This behavior has been best described with component type curves that use the concept of two homogeneous system responses (fissure and total), with a transition regime during the period of pressure support. Two main flow types between blocks and fissures (interporosity flow), pseudosteady-state (Col.1 in Figure 1-3) and transient (Col.2 in Figure 1-3), with different transition responses have been envisaged.

As both double-porosity models are based on two infinite, homogeneous systems with a reduction in the rate of pressure change during transition, the ideal result on a semilog plot shows two parallel lines (Top illustrations in Figure 1-3), with a transition portion of smaller slope. (a half-slope region theoretically can be observed with transient interporosity flow). The existence of the first straight line, however, is often shadowed by wellbore storage.

When the derivative is taken with respect to the log of time, the main trend is the same as that of the infinite, homogeneous system, with a drop in the derivative when the rate of pressure change decreases during transition. In that region, the response is similar to that produced by pressure-maintenance boundaries, in the case of pseudosteady-state interporosity flow. Radial flow is again characterized by a horizontal straight line at

$$(t_D/C_D)p'_D = 0.5.$$

The position of the transition dip is controlled by the interporosity flow coefficient, λ, and the depth and length of the transition period are dictated by the storativity ratio, ω.

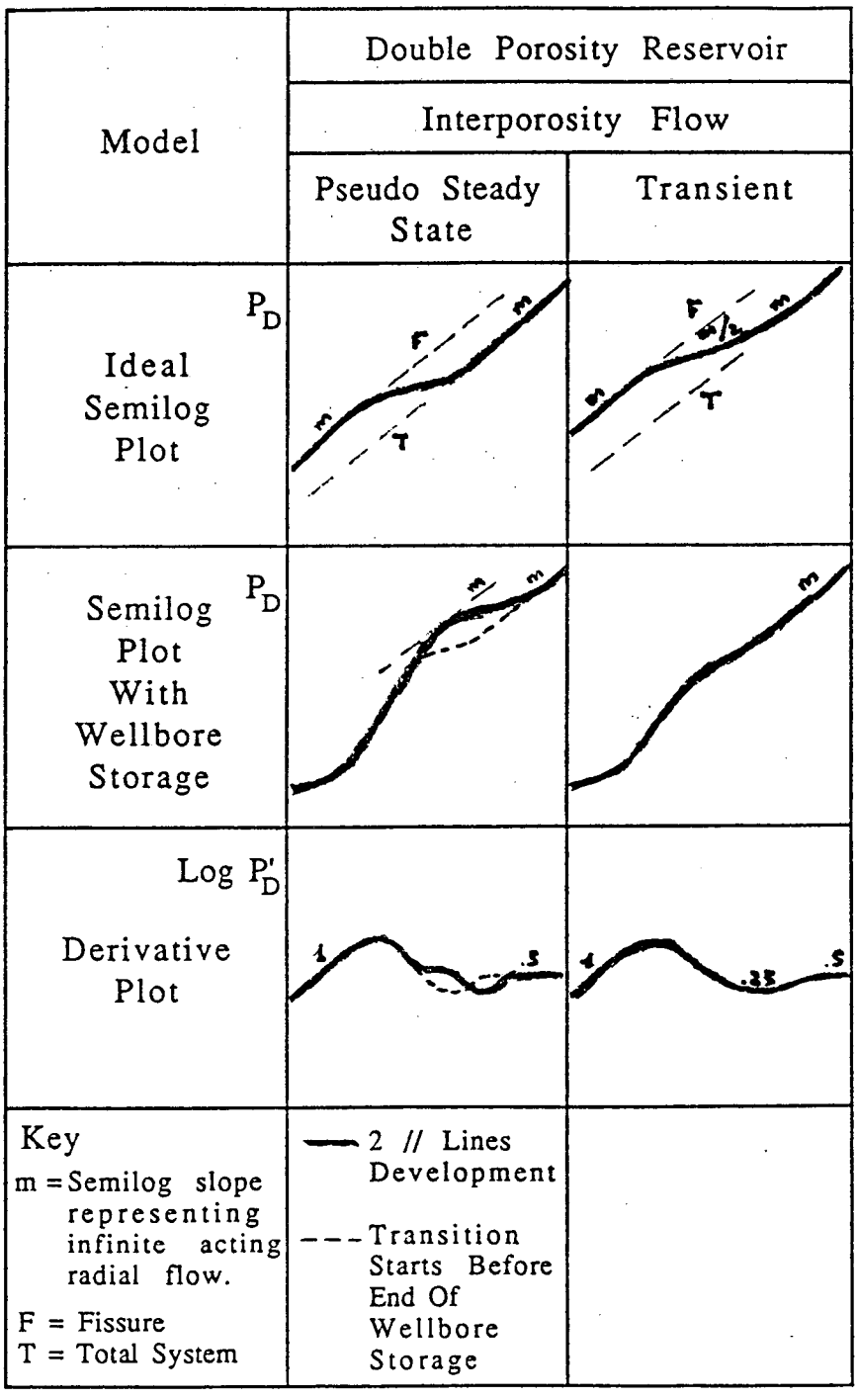


Figure 1-3: Summary of double porosity reservoir model responses.

The transition period is usually longer with transient interporosity flow than with pseudosteady-state interporosity flow, with the theoretical possibility for the development of a straight line with a slope equalling half of that which corresponds to radial flow in the total system. When viewed in the derivative format, the half slope would be represented by a horizontal line at

$$(t_D/C_D)p'_D = 0.25.$$

The derivative plot gives much more distinctive shapes for the different double-porosity models although the pseudosteady-state model with large ω values still can be confused with the transient model. In most cases, if the transition drops below 0.25, then a pseudosteady-state model can be inferred.

1.1.1.3.2. BUILDUP

Up to this point, the models and type curves have been for an initial drawdown from static conditions at a constant rate. The production rate generally is not sufficiently stable for analysis of the drawdown. Consequently, most transient test analysis is focused on the buildup when the rate, at the surface at least, is well-defined and constant -that is, zero (a buildup consists of closing a well after a production period and letting it reattain its natural hydrogeologic static regime).

For conventional analysis methods to be useful for buildups, they must be modified to account for preceding flow periods. The most commonly used plot, semilog, is thus replaced by a superposition plot, where the log of time function $ln\Delta t$ is replaced by the superposition function[1]

$$\frac{1}{q_n - q_{n-1}} \left[\sum_{i=1}^{n-1} (q_i - q_{i-1}) \, ln \left(\sum_{j=i}^{n-1} \Delta t_j + \Delta t \right) \right] + ln\Delta t$$

where t_0 = time at start of flow-test, Δt_i = elapsed time when the ith flow rate (q_i) stopped and Δt_{n-1} = elapsed time at shut-in.

The superposition plot restitutes the semilog straight line resulting from infinite-acting radial flow by compressing the later section of the time axis.

For the global approach of type curve matching, the problem of buildup analysis is more difficult to overcome, as the response is restricted in magnitude to the final flowing-pressure difference of the previous flow period. Therefore, the buildup response asymptotically approaches this level, resulting in a flattening of the trend at late times. This flattening means that matching buildup data on drawdown type curves is a very risky process. Although it is time-consuming, the best method to overcome that problem is the generation of the appropriate multirate type curves for each buildup, thus accounting for the preceding changes in flow rate.

If the derivative of the pressure is taken with respect to the superposition function, then the normalizing effect, which restitutes the radial-flow straight line, will also reproduces the characteristic horizontal line of the derivative plot. More precisely, if the preceding drawdown is sufficiently long for the complete combinations of conditions and boundaries to have been encountered, then the buildup data can be matched successfully on drawdown type curves.

1.1.2. Applicability Of Expert Systems

As seen above, the process of reasoning in well-test analysis can be summed up in two main steps: first, reservoir system diagnosis from the characteristic patterns on the different plots, and second, estimation of system parameters using the appropriate set of type curves. A human expert is able to select at first time a small set of models to which the considered well could be relevant. This first analysis is improved by the use of the various type curves, which yields parameters and model for the well.

An expert system will have more difficulties to determine the appropriate model for a well: first, it will have to recognize the

characteristic. shapes on the plots, second, it will have to infer the right model from these shapes. As the different models are not too numerous and have sufficiently characteristic shapes (except for one or two, which will need more specific analysis involving quantitative evaluations), this second part should not be too difficult for an expert system: it would be relatively easy to build a knowledge base which contains the rules to extract the system model from the patterns, as soon as those patterns are well-defined. One typical rule could be:

> *"If there is a hump at the beginning of the derivative plot followed by a horizontal straight line, then the model of the well is homogeneous with infinite-acting radial flow and wellbore storage and skin."*

As the data are often very noisy, the main difficulty will probably appear during the shapes recognition. Two different approaches can be used to solve this problem: the first one consists of having different numerical algorithms to filter, then compute derivative and patterns, as precisely as possible; the second basically uses the same algorithms, but with larger error bounds, to be sure to extract all the characteristic shapes from the curves. A second step consists of checking the patterns on the different plots, to get rid of the wrong ones. This second method involves more expertise, and an expert system could probably deal very well with it.

The second part of the analysis, the one which uses type curves, has been developed in recent years with the help of computer packages. These numerical tools[10] work very well with data not too far from the analytical models, but give poor results when abnormal shapes appear on the curves (for instance, in the case of atmospheric pressure changes during the well-test). An expert system would be able to select the significant parts of the curves before trying to match them on the type curves, as a human expert would do.

In a domain like well-test analysis, where the input of the problem (the patterns on the different plots) can not be exactly defined, expert systems are probably able to do much better than

classical programs, because they can deal much more easily with symbolic knowledge (as a hump or a straight line on a plot), and therefore reproduce part of the human reasoning.

1.2. THE EXISTING SYSTEM, WES

1.2.1 INTRODUCTION

WES, Well-test analysis Expert System, was designed at the beginning to work in the field of nuclear waste disposal. This narrow field has been extended now to the more general domain of well-test analysis, method involved not only in waste storage problems, but also in oil reservoir exploitation.

The following description represents the state of the program at the beginning of my internship. Although the current version is very different in its form, the main architecture remains the same. Therefore this architecture must be described. In the following sections, each time the text will refer to the "current state of the program", the original state must be considered. The differences between the original and current versions will be explained in parts II and III.

In its current state, the prototype is by no means a finished system, and solves only a small part of the different kinds of problems for which it is conceived. However, it can be useful to give a quick description of this prototype, to discuss what should be improved and what should be added to the existing program.

The system is based on the theory described in the preceding parts, the guiding principle in its design has been not only to try to reproduce the results obtained by the expert, but also to try to mimic the way the expert reaches a conclusion.

In its present state the system can accommodate only a single testing well, and this test must be a drawdown. It possesses knowledge about only a limited set of conceptual models, more or less the ones which have been described in the first section of this

report. However, it underlines the difficulties encountered during the realization of what has been called in the theory "the first part of the analysis", i.e. the identification of the model.

The system is written in ART (Automated Reasoning Tool, from Inference Corp., Clayton, 1984). ART provides an easy interface to LISP, so part of the system is written in LISP.

The following sections describe the data structure, the architecture of the system, the various algorithms used for computing derivatives or extract global characteristics of the curves and the ways the system uses to deal with the different conclusions he reaches during the run.

1.2.2 DATA STRUCTURE

The system uses a frame-oriented approach to structure its data and results obtained during execution. The basic architecture of objects refers to wells and characteristics of wells, but objects are also used for graphics and user interface (ART allows windows, curves and mouse-sensitive icons to be represented as objects).

The object architecture relative to wells (Figure 1-4) is slightly different before and during execution. Before execution, only generic objects (**well** and its four children, **well-semilog**, **well-loglog**, **well-derivative** and **well-model**) are defined, along with all the slots that may be used to characterize these objects. These generic objects are basically empty structures that reproduce the relational knowledge.

At execution time, any well-test the program studies (i.e. Lm40) is viewed by the program as a specified instance of the generic parent object **well**. When needed during the run, the system also creates instances of the four children objects (i.e. Lm40-semilog, Lm40-loglog, Lm40-derivative and Lm40-model). These different

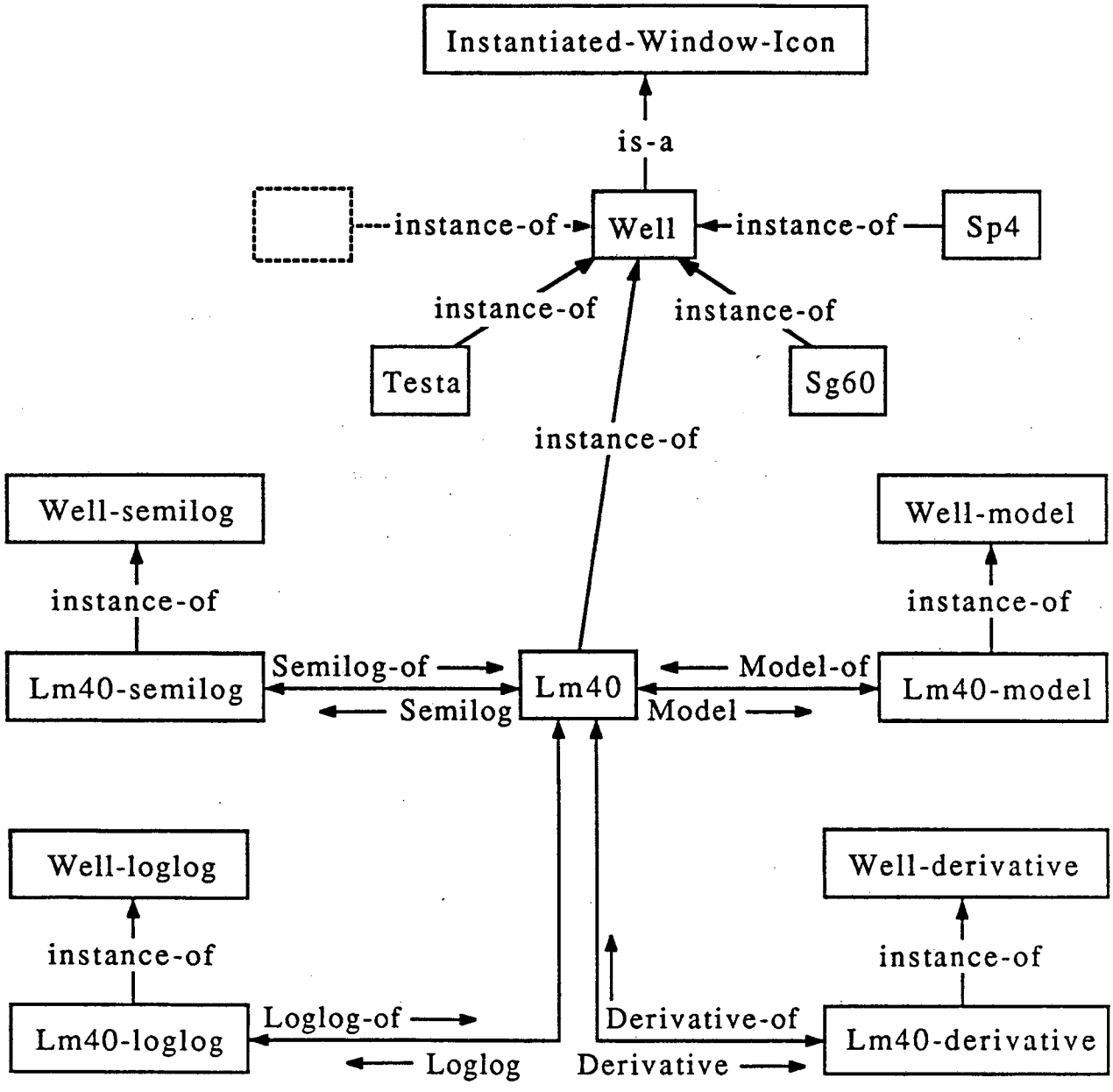


Figure 1-4: Object architecture for wells in WES.

instances are replica of the generic objects, but the empty slots are replaced by the characteristics of the well (i.e. Lm40).

The generic objects may have several different instances simultaneously during the run: each time a new well is considered for analysis, a new set of specific objects is created. Knowledge is kept in these specific objects, therefore avoiding any confusion between the different well-tests.

The main slots of the five generic objects are shown in the table below:

Well	Well-semilog	Well-loglog	Well-derivative	Well-model
initial-time initial-pressure time pressure semilog loglog derivative model x-scale y-scale initial-curve filtered-curve common-icons window	semilog-of straight-lines semilog-curve semilog-icons	loglog-of straight-lines loglog-curve loglog-icons	derivative-of model time p-derivative p-d-derivative straight-lines humps derivative-curve derivative-icons	model-of reservoir boundary wellbore-storage fractured early intermediate late

Table 1-1: Main slots of the generic objects.

Initial knowledge is stored in the parent object (instance of the generic parent object well) for a given well. It consists of the initial and filtered time and pressure data, the links to the four children objects and other objects such as the dimensions of the window associated with this particular well.

Well-semilog contains knowledge about the semilog curve, such as straight lines. **Well-loglog** contains the same knowledge for the log-log curve. **Well-derivative** contains the pressure derivative, the second order derivative, and characteristics of the pressure derivative curve such as straight lines and humps.

Well-model is the object that contains symbolic assertions about the well, that is, deductions reached by the system according to the different characteristics of the semilog, log-log and derivative plots. **Reservoir** describes the overall behavior of the medium. It can take the values *homogeneous* or *double-porosity*. **Boundary** describes the boundary behavior of the system. Its values can be *no-flow*, *infinite* or *pressure-maintenance*. The attributes **wellbore-storage** and **fractured** contain only *true* or *false* depending on the occurrence of such phenomena.

These five objects contain also graphic-related attributes. In ART, all graphics (windows, lines, rectangles, text, etc...) are also represented by objects and attributes. For example, a line object will have attributes such as origin, endpoint or thickness. The attributes in the table ending in **-curve** or **-icons** point to such objects.

1.2.3. RULE ARCHITECTURE

Even though WES is a rule-based system, in which the flow of control should be by nature opportunistic, the system always proceeds sequentially through a number of steps. Within each of these steps, several rules are executable at the same time, depending on whether their left-hand-side (or 'if' part) conditions are satisfied by facts in the current state of the system or not. The rules do not refer to a specific well, but rather can be executed "simultaneously" for more than one well so that parallel analysis of many wells is possible.

The differents steps of the execution are:

- Data extraction and initial curve display.
- Data filtering and curves computation.
- Curve characteristics extraction on semilog, log-log and derivative plots.
- Generation of hypotheses.
- Selection of one or more possible models.

1.2.3.1. Data Extraction - Initial Curve Drawing

The initial data (readings of pressure at different times) is read from a file when a well is selected for analysis. This data can be stored in different units (hours, seconds for time, psi, pascal for pressure), but will always be converted to seconds and psi.

The initial curve, which represents exactly the data read from the file, is then drawn in a window, which also contains five mouse-clickable icons (**initial**, **filtered**, **semilog**, **loglog**, and **derivative**). The reasoning process on a specific well is started by clicking on the filtered icon. After that, the five icons have the same function of displaying the corresponding curve (and its characteristics, such as straight lines or humps) into the window.

1.2.3.2. Data Filtering - Curves Computation

Since data sets can be very different in size and may sometimes contain a lot of data points, the system first filters the initial data set and selects a specified number of data points (sixty in the present state). The time scale is then divided in sixty constant intervals (on a log scale, since the abscissa for all the curves used in the analysis in the log of time): one data point on the filtered plot is the result of the averaging of all the data points which are in the corresponding interval on the initial plot.

Since pressure data is generally recorded at fairly constant time intervals, the density of data points on a log scale increases dramatically with the time. The filtering process thus results often in a drastic reduction of the number of late data points, whereas the system keeps most the data points in the early part of the initial curve.

Besides discarding too numerous data points the filtering phase smooths the data in order to avoid undesirable and meaningless noise.

The system then computes the curves used in the analysis, i.e. the semilog, the log-log and the derivative curve. The program also checks the validity of the latter curve: since the test under analysis is a drawdown, the pressure is always diminishing, thus the derivative must always be negative. In case the system finds a positive pressure derivative at some particular time, a warning message is displayed and the value is set to 0, or rather -0.01, which is the biggest number that the system can admit (All negative numbers with an absolute value lower than 0.01 are also set back to -0.01).

In order to achieve the shapes extraction from the curves, the system will need the slope at each point on the different plots. Thus the second order derivative (slope of the first order pressure derivative, i.e. d(ln(dp/dlnt))/dlnt) is computed, as well as the derivative of the log-log curve (dlnp/dlnt). The slopes on the semilog plot are already known, since they correspond to the values on the derivative plot.

One of the main inconveniences of the pressure derivative approach is that it cannot be measured directly but rather must be computed from discrete data. The algorithm that is used currently to compute the various derivatives is inspired from the one described in Bourdet et al. It may still be improved but gives acceptable results, that is, it preserves most of the meaningful response of the system while removing most of the noisy parts.

The algorithm computes the weighted mean of the slopes between the point under study and a point preceding it, and between the point under study and a point following it. The two points are not the points closest to the point of interest, but instead are defined by skipping several points to go from the point under study to the two points where the slope will be taken. Since in WES all the points are equally spaced on a log scale, this amounts to using points at constant intervals from the point of interest. The current number of intervals used by the system is 3.

With the notations of Figure 1-5, the slope p' is given by:

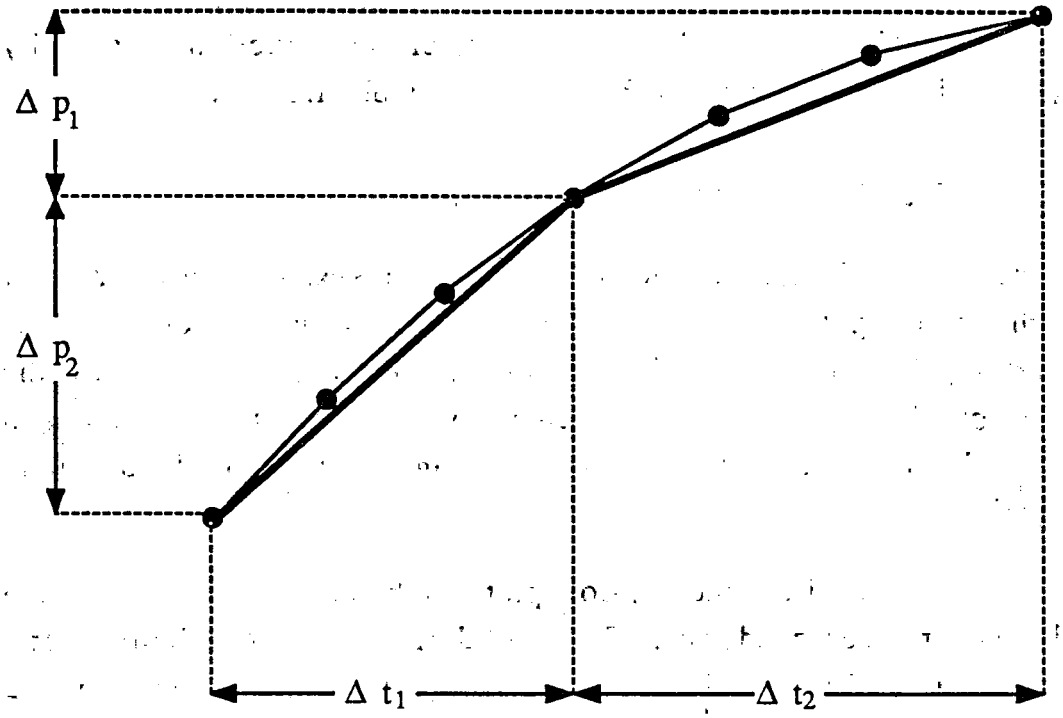


Figure 1-5: Notations used for computing derivatives.

humps is symmetrical to this one and the two are actually implemented as one LISP function, which return whether the hump identified is a *hill* or a *valley* hump.

Since the extremum computed by this algorithm is not necessarily the real extremum on the curve (the second order derivative is often very noisy), the program then looks for an extremum on the derivative plot in the neighborhood of the computed one.

```
        i = 0
1       if p"i ≥ 0 then goto 2 else i = i+1; goto 1
2       j =
        noise = [tj, tj]
3       if tj+1 exists and length(noise) ≤ significant-length
        then if p"i ≥ p"j ≥ p"j+1
                then before-noise = tj
                     noise = [tj, tj]
                     hump = [ti, tj]
                     if p"jp"j+1 < 0 then top = tj
             else noise = [before-noise, tj]
             j = j+1
             goto 3
        else if length(hump) ≥ 3 significant-length
             and length([ti, top]) ≥ significant-length
             and length([top, tj]) ≥ significant-length
             then return hump: ti, tj, top
             if tj+1 does not exist then stop
        i = j
        goto 1
```

with the same notations as the ones used for the preceding algorithm.

1.2.3.4. Generation Of Possible Models

Partial conceptual models are generated from the descriptions of the curve in terms of straight lines and humps. Rules try to represent the decision criteria used by the expert. An example of such a rule is:

"If there is a hill hump followed by a horizontal straight line on the derivative plot, then generate the hypothesis of a homogeneous and infinite system."

These rules can generate hypotheses based on small parts of the pressure versus time curves, for example wellbore storage from early time data, or the type of boundary from the late part of the curves. Those partial hypotheses can then be grouped together to form a complete model, such as wellbore storage in a homogeneous medium with a no flow boundary. The program can only group compatible partial model. For example, grouping an infinite system with a no-flow boundary is not allowed. These incompatible hypotheses are to be kept separate in different hypothetical worlds.

This concept of hypothetical worlds is one of the most interesting features of the system and should be examined with more details. During the analysis, an expert is likely to follow several alternative interpretations, until he is eventually convinced that one is more likely than the others. The analysis may also end-up with a dead-lock, the expert concluding that more information is needed to reach a reliable conclusion.

WES models this kind of behavior by generating hypotheses about the actual model. These different hypotheses are kept in separate "worlds". Each of these hypothetical worlds is the state of the data base resulting from one hypothesis, or one group of compatible hypotheses. The program maintains these worlds simultaneously, and rules of inference can be activated independently in each of them depending on the facts present in the particular state of the data base. The rules generating hypotheses are fairly weak, and as soon as there is a slight chance for a particular

model to be true, the corresponding hypothesis is generated. This is the way the system takes care of the inherent imprecision of the theory.

Note that the basic data architecture described in 1.2.2. is common to all the subsequent hypothetical worlds generated by the system. The "raw" data, that is, the time and pressure data and the different curves are also common to all the hypothetical worlds. The distinction between these worlds appears for the **well-model** object.

This concept of hypothetical world is achieved by making use of the more general concept of *viewpoints* available in ART. The system uses one level of viewpoints, and each viewpoint represents one hypothetical world.

1.2.3.5. Selection Of One Or More Models

In the present state of the program, all possible complete models are simply printed on the screen. A complete model is a model which gives at least the type of medium (homogeneous or double porosity) and the kind of boundary (infinite, no flow or pressure maintenance). The early time characteristics wellbore storage or fractured formation near the well, are optional. In the current implementation of WES, the same model may be printed more than once in some instances. It means that the system reached the same conclusion by two or more paths of reasoning. A model that appears more than once is thus more likely to be true than a model printed only once.

In some case, for some data sets, no models are proposed. It means either that the expertise present in the system is not sufficient to analyze this particular case or that this data set has some abnormal shape, due to some external event.

1.3. PROBLEMS

1.3.1. General Description

The different problems that are to be solved can be divided in two parts: the first one consists of improving the solutions that have been used in the prototype described in the preceding section, and, for some of these solutions, trying to propose new possibilities. The second kind of problems is related of the part of the analysis that is not taken in account in the present state of the program, that is, introduction of new models and use of the different sets of type curves essentially.

In an other field, the translation of the program written in ART in a lower level language, such as LISP or C, will have to be studied in the future. ART is a very powerful tool to build an expert system, but still has two main drawbacks: first its price ($50000 for private companies), and second its performances and the environment it needs. ART is very slow and needs to be run on a SUN. If the program has to be used in the future in real-time conditions, that is, if the well-test has to be analyzed during its execution, those drawbacks would become real problems. As will be seen below, this problem of translation has appeared sooner than expected.

The following sections describe respectively the work that has to be done in the numerical field, the well-test analysis field and about the conversion problem.

1.3.2. Numerical Problems

During the first part of the analysis, the system has to compute the curves that will be used for the analysis, and then tries to extract the signatures of these curves using several algorithms. This is

already done in the existing program, but it appears that the results are not always as good as expected. Since the following of the analysis depends heavily on this part, it could be worth studying with more accuracy this question.

Since the derivative plot takes a large importance in the analysis, the first thing to do is to try different algorithms and figure out which is the best in the particular case of well-test analysis: this algorithm will have to keep the meaningful response of the reservoir while removing the noisy parts. The algorithm in use presently is certainly one of the best, but may still be improved, especially for the points in the early and late parts of the curve (these parts are very important to determine inner and outer boundary effects).

The computation of the main shapes, such as straight lines and humps, is also achieved by the present system, but suffers of a kind of rigidity, since the different algorithms use absolute error bounds to determine whether a point belongs or not to a straight line or a hump. This first possibility, the easiest one, would be to modify the existing algorithms (described in section 1.2.3) and introduce more relative error bounds. A more important modification would be to introduce more expertise during the pattern recognition, that is, to combine this recognition with the well model extraction. In the present state of the program, these two steps are totally distinct, and this does not correspond to the way a human expert works.

The present description of the curves uses the concepts of straight lines and humps, which are in turn based on the value of the second order derivative. This is an other drawback of this method, because this derivative is often extremely noisy, and sometimes totally unusable. This difficulty could probably be solved by the introduction of a slightly different description of the curves that will be based on straight lines only.

An other numerical problem that has to be studied is the generation of the different sets of type curves. Since this question is

more directly relevant to the well-test analysis field, it will be described in the following part.

1.3.3. IMPROVEMENT OF THE ANALYSIS

In its present state, the system possesses knowledge about only a limited set of models, which represents the most general types of wells that can be found. These models will have to be refined by introducing new properties that will be able to describe characteristics such as shape or size of the reservoir. This is certainly a very important thing to be done in the long term, but the current state of the program needs a different kind of work, since even the basic models are not always recognized. The first difficulty to overcome is then to try to obtain a reliable response for the models the system is able to handle actually before thinking of increasing the number of these models.

This can be achieved using three different ways: first, improve the pattern recognition part using either better algorithms or more expertise; this possibility has been discussed in the preceding section. Second, introduce type curves for the different models the program has selected for the well under study to refine the analysis (this will work only if the right model belongs to the set of selected models, what is not always true in the present state). Third, use more analytical sets of data to build the knowledge base (the data the program uses now are often very noisy, sometimes even too noisy to be usable by a human expert !), and then introduce some specific noises in these analytical data to see how the rules react. As ever in similar cases, a combination of these three methods will probably lead to the best results.

The generation of type curves and analytical solutions will probably need a lot of bibliography researches, to find the equations for the different kinds of behavior a well is able to have. The second step will be a numerical one, since some of these equations (for example, the one that models wellbore storage effects) are given in

an integral form. The use of tables could also be an acceptable solution for some of these type curves. The introduction of different sorts of noise in the curves will need more help from the expert, since there are few chances to find something about this problem in the existing papers.

The introduction of the first set of type curves in the existing program, written in ART, had an important "side effect": the ART expert system shell was not able to handle the quantity of facts that have been introduced, and the execution speed was dramatically slowed down (Since the system is written in LISP, he uses the "Garbage Collector" to manage the memory space. This tool is called automatically when needed. The introduction of the new facts resulted in an important rise of the number of calls to GC). This problem leads to the more general question of the translation of the system, or at least part of the system, in a lower level language, such as LISP or C.

1.3.4. TRANSLATION OF THE PROGRAM

The problem of the conversion of the existing program, written in ART and LISP, to an other language was considered at the beginning in the long term, but the difficulties that appeared with the introduction of type curves in the ART program give to this problem a higher priority, and even the highest. The facts present in the database during the run are obviously too numerous in the present state of the system, and the important thing to consider is that only few of these facts are directly useful for the analysis. In fact, a large majority of them, such as window descriptors or mouse-sensitive icons, is used for graphics and for the user interface. This part of the program is an important one, and have to be kept, because it gives to the system a lot of flexibility, and therefore mimic, in one way, the behavior of a human expert. In the other hand, there is no real need to keep it in ART, since graphics are not directly used for the expertise. Although this interface is excellent, it

should be possible to translate at least part of it in an other language, to save the memory space for the facts used for the analysis, such as straight lines or humps.

The question of the final conversion of the system will appear in the long term, but will concern only the "expert" part, since the graphic interface will already be in a low-level language. Once the structure of the system is well-defined, this part might be rewritten, even if it would probably take a lot of time.

PART II - RESULTS

2.1. NUMERICAL ALGORITHMS

2.1.1. INTRODUCTION

This chapter describes the research that has been done in the field of numerical algorithm improvement. This is not directly related to Artificial Intelligence and Expert Systems, but is still an important problem in the particular case of WES: since all the analysis is dependent on this first step, the results it gives must be reliable enough.

The two major problems are derivative computation and shapes recognition (since the shapes are computed with numerical algorithms in the present state of the system). As explained in the first section, both are actually linked together, since the straight lines and humps are computed from the slopes of the different curves, which are in turn given by the derivatives. The different results obtained for these two problems are given in the two following sections. The possibility of using more expertise in the pattern recognition part has not been studied yet, but will be discussed in the last part of this report.

In the existing program, the initial data is filtered at the beginning of the run. This filtering has an interesting smoothing effect, but also represents a loss of information. As will be seen below, the different differentiation algorithms also include a smoothing effect, and thus the initial filtering of the data is no longer useful (Since all the programs used for testing these algorithms are written in C, the problem of speed that appeared in ART with numerous data sets is also solved).

2.1.2. DERIVATIVE COMPUTATION

The problem of derivative computation is not generally considered as a major difficulty in applied mathematics. In the domain of well-test analysis, this derivative has to be computed from real discrete data that is often very noisy. The algorithm should be therefore able to smooth enough the curve to remove the meaningless noise while keeping its characteristic shapes. Since the values of slopes in the early and late parts of the curves are highly useful in well-test analysis, this particular problem must also be considered with extreme attention.

The following sections describe respectively the selection of the algorithm, the estimation of parameter influence, such as the length of the differentiation interval, and other less general problems, such as introduction of the superposition function in the case of a buildup. The validity of the second order derivative (in terms of errors) is also checked.

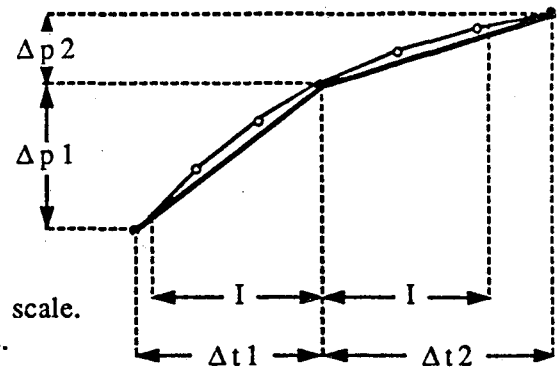
2.1.2.1. Algorithm Selection

Three methods were basically tested, each of them using two different ways. **Method 1** is the one in use in the existing program (cf 1.2.3.2.), with some modification in the way the preceding and following points are computed: instead of choosing these points as the third from the point of interest in each direction, the program takes the first point outside of a given interval in each direction. The main inconvenience of the method based on a fixed number of points appears when the points are not regularly spaced on the log of time scale. It was assumed in the design of WES that all the filtered data points were at fairly constant distances, since each of them was in a different interval of constant length (cf 1.2.3.2). Problems appear with data sets that contains less than sixty points (number of filtered data points currently used by the program): in that case some of the intervals are empty, especially in the first part of the curve, and this can lead to important errors in the estimation of the derivative.

Methods 1 and 4:

$$\text{Slope} = \frac{\Delta t2 \frac{\Delta p1}{\Delta t1} + \Delta t1 \frac{\Delta p2}{\Delta t2}}{\Delta t1 + \Delta t2}$$

Method 1: p' = slope. Δt on a log of time scale.
Method 4: p' = t.slope. Δt on a time scale.

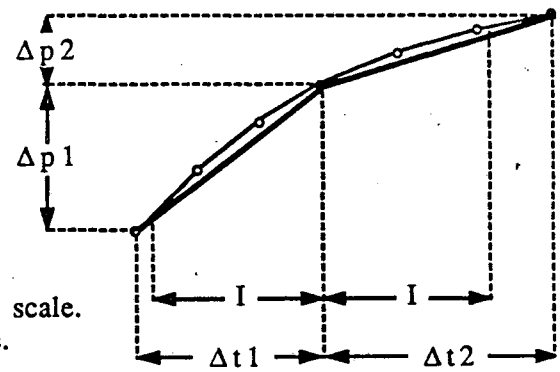


Methods 1 and 4: Notations.

Methods 2 and 5:

$$\frac{\Delta t1 + \Delta t2}{\text{Slope}} = \frac{\Delta t2}{\frac{\Delta p1}{\Delta t1}} + \frac{\Delta t1}{\frac{\Delta p2}{\Delta t2}}$$

Method 2: p' = slope. Δt on a log of time scale.
Method 5: p' = t.slope. Δt on a time scale.

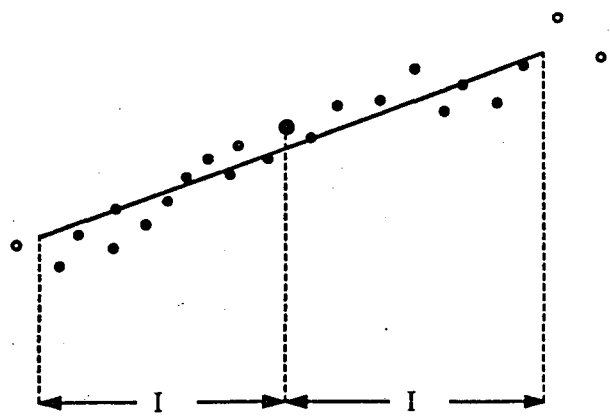


Methods 2 and 5: Notations.

Methods 3 and 6:

The slope at the point of interest (big black point) is computed with a least-squares method using all the black points.

Method 3: p' = slope. Abscissas on a log of time scale.
Method 6: p' = t.slope. Abscissas on a time scale.



Methods 3 and 6: Notations.

Figure 2-1: Summary of the derivative methods.

Method 2 is based on the same principle of computing the weighted average of two slopes, but the program takes the geometrical weighted average instead of the classical arithmetical one. This "exotic" method was successfully used in one particular case (computation of the wellbore storage type curve derivatives, where the type curves were given by tables - cf 2.2.), this is why it is described here.

Method 3 uses a least-squares algorithm. An interval of given length is chosen on each side of the point under study, yielding a subset of data points. The "best" straight line, according to the least-squares criteria, is then computed for this subset of points: the slope of the straight line represents the derivative at the point under study.

These three methods compute derivatives with respect to the log of time function, that is, all the interval lengths are given on a log scale.

Methods 4, 5 and 6 are respectively the same as **methods 1, 2 and 3**, but the derivative is computed with respect to the time function, and then multiplied by the time at the point of interest (according to the formula: dp/dlnt = t.dp/dt).

These different methods are summarized in Figure 2-1.

The algorithms were tested on an analytical function, to compare the results obtained from discrete derivation with the theoretical one. The function used is an approximation of the type curve for double porosity reservoir model[5] (see 2.2. for more details) and is given by

$$f(\tau) = 1/2\ \{ln\tau + 0.80908 + E_i[-\lambda\tau/\omega(1-\omega)] - E_i[-\lambda\tau/(1-\omega)]\}$$

where $\lambda = 5.10^{-6}$ and $\omega = 0.1$. E_i is given by

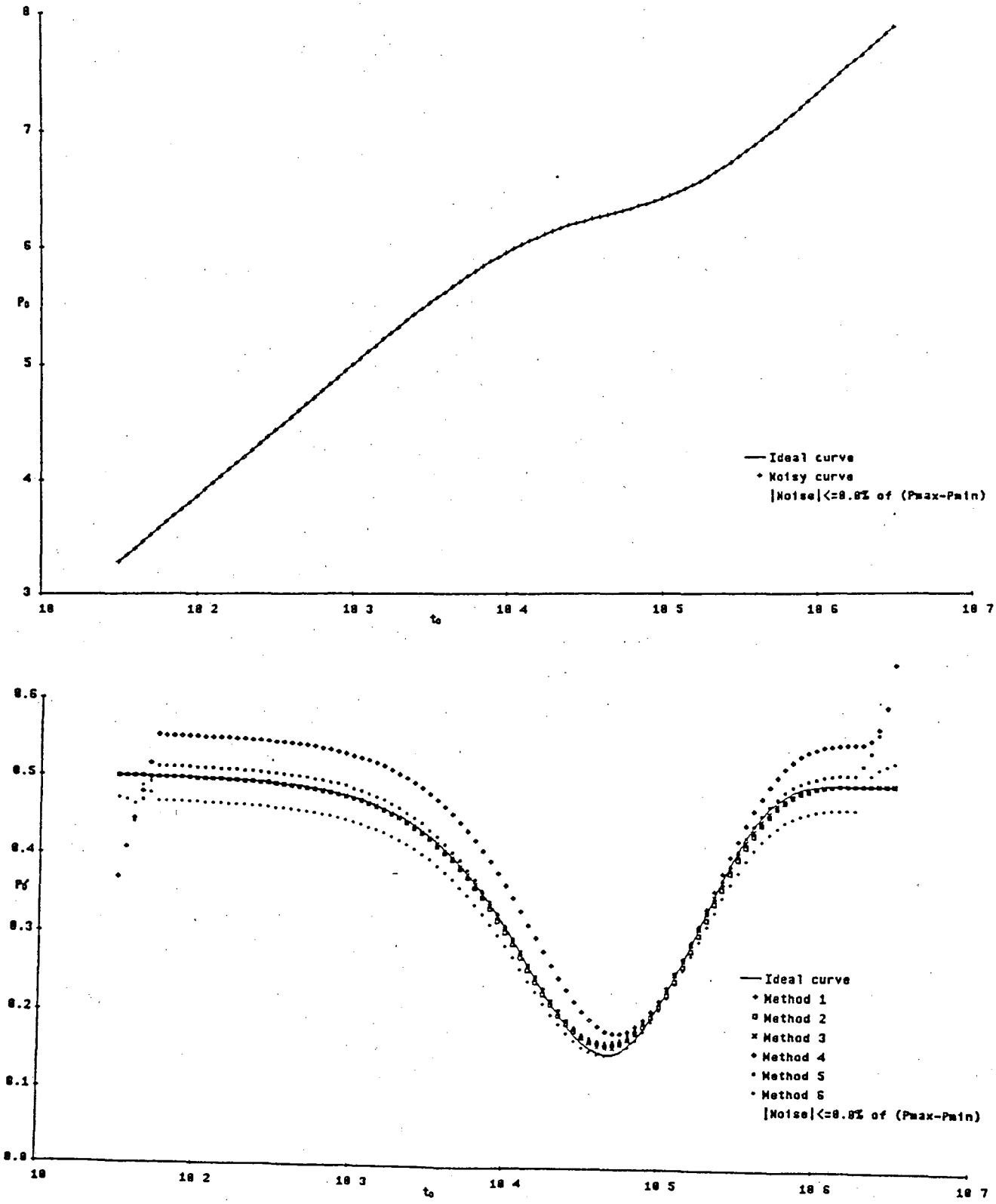


Figure 2-2: Semilog and derivative curves for the six derivative methods. Noise range = 0%.

$$-E_i(-x) = \int_x^{\infty} \frac{e^{-u}}{u} du$$

This type curve is by no means the most characteristic one (one including wellbore storage would have been better), but is the only one which is given by a simple equation: the others type curves are often given in the Laplace domain or by integral equations.

Results for real data sets are shown in Appendix A.

Random noise of different ranges has been added to the theoretical curve to study the responses of the algorithms. The results are given either on plots or in tables. The tables give for a given range of noise and for each method the maximum deviation and the average deviation from the theoretical derivative (computed without noise). These deviations are given in percentage of the theoretical curve maximum range. The length of the derivation interval used for all the methods is one half of a natural log cycle, i.e. 0.215 decimal log cycle.

On Figure 2-2, the curves corresponding to the six different methods for an initial curve without noise have been plotted: it is obvious that the three algorithms based on the derivative with respect to the time function give important systematical errors Moreover, the values given for the early and late parts of the curve are very bad too. Methods 1, 2 and 3 give very similar results, smoothing the peak of the analytical derivative. These three methods are obviously better than methods 4, 5 and 6, and are the ones that will be considered in the remainder of this study.

On Figure 2-3, the range of the random noise is equal to 2% of the maximum range of the initial data. It shows that method 2 seems to give worse results than the two others ones. This is confirmed by Table 2-1, where the results are summarized for the six methods. When the range of noise increases to 5% of the initial data range,

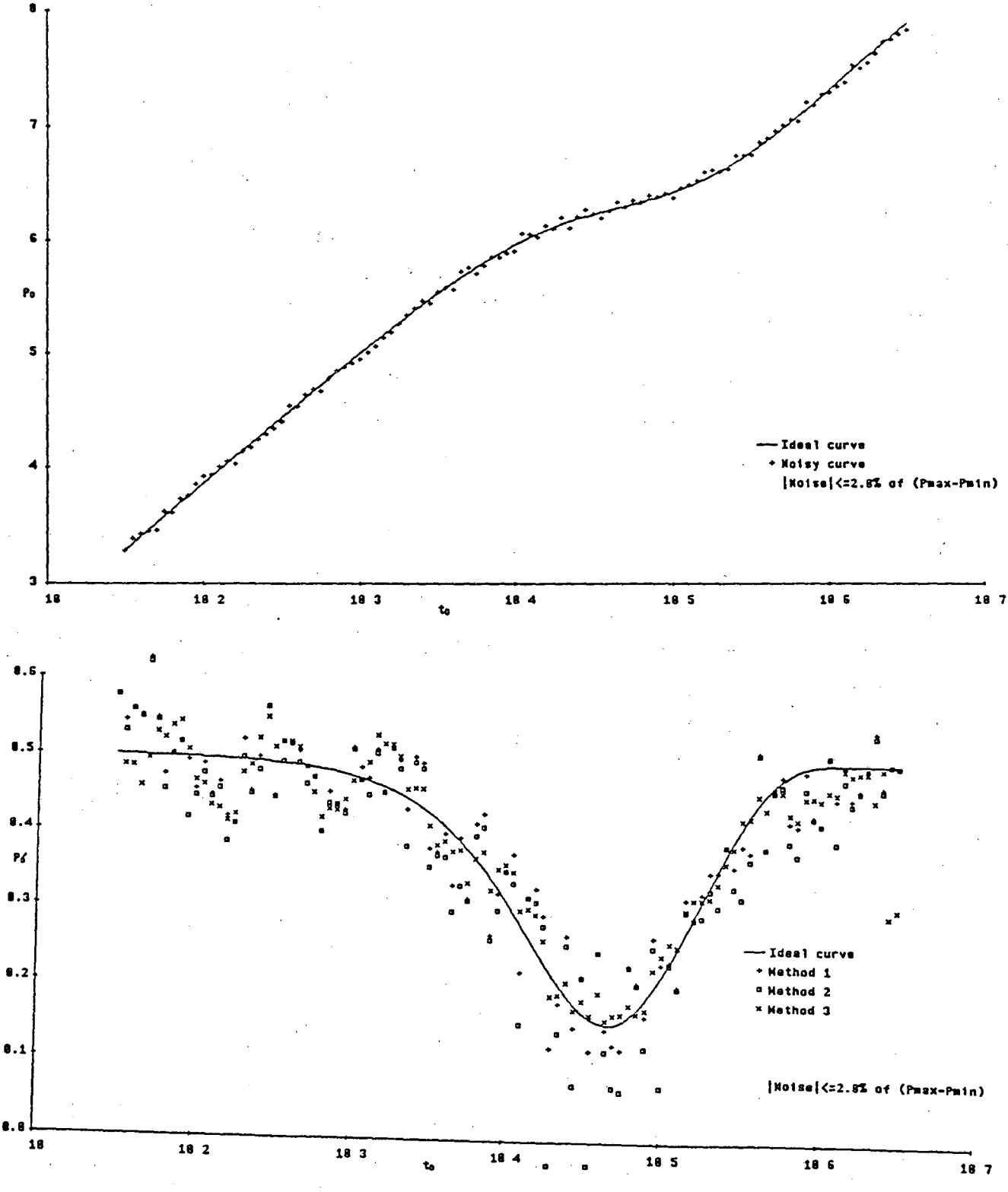


Figure 2-3: Semilog and derivative curves for the methods 1 to 3. Noise range = 2%.

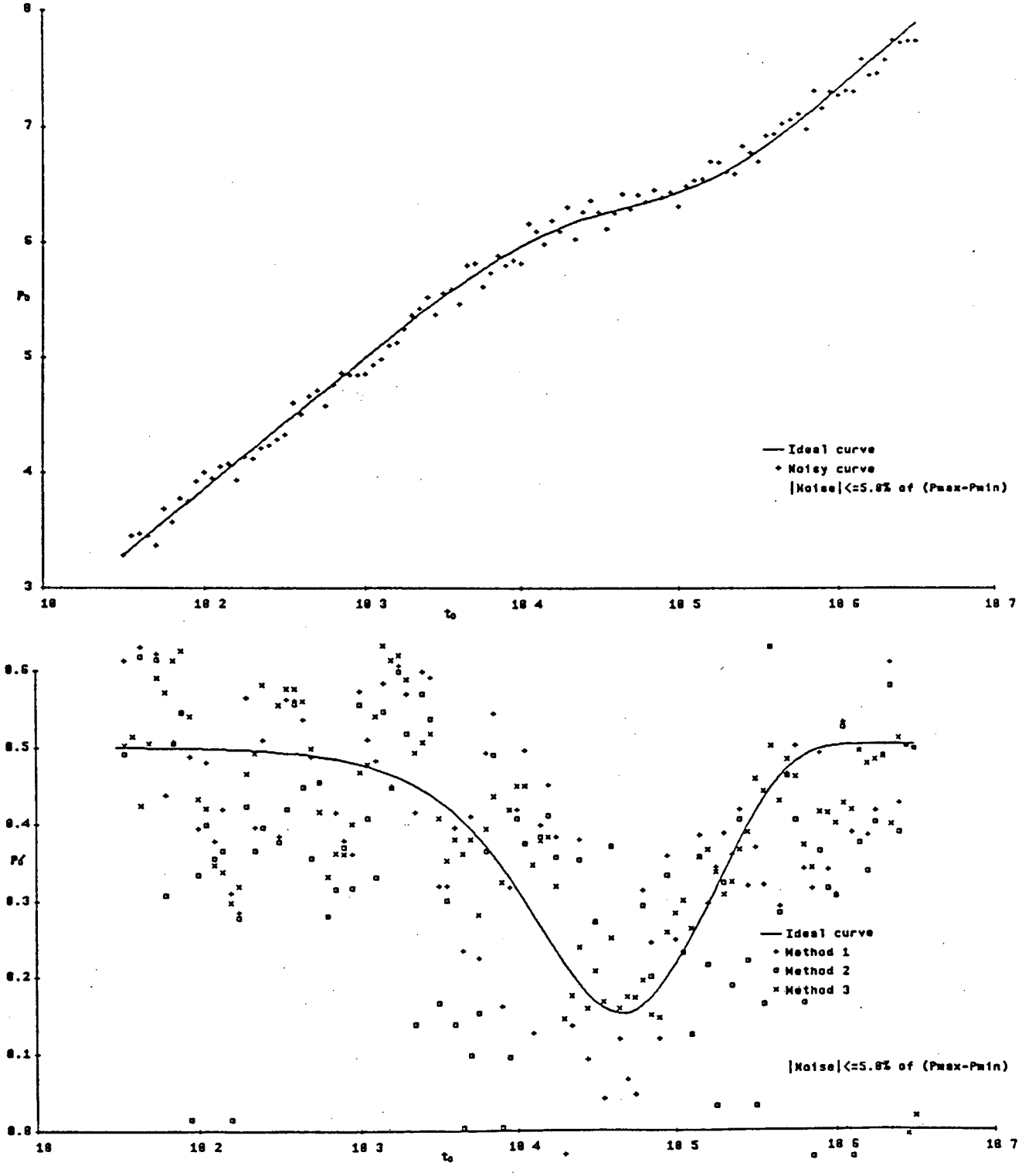


Figure 2-4: Semilog and derivative curves for the methods 1 to 3. Noise range = 5%.

Figure 2-4 is obtained. It shows that the computed derivative is hardly usable, at least with the derivation interval that has been used in this example. A more complete set of curves (with more values for the range of noise) is given in Appendix A.

Noise (%)	0.0		1.0		2.0		5.0		10.0	
Deviation	Max D	Av. D	Max D	Av. D	Max D	Av. D	Max D	Av. D	Max D	Av. D
Method 1	3.4	0.8	17.6	5.6	35.2	10.9	88.0	27.2	176.0	54.3
Method 2	3.3	1.0	24.9	6.4	68.8	14.5	883.3	66.6	-----	----
Method 3	2.4	0.6	53.4	4.9	107.1	9.4	267.7	23.2	535.4	52.3
Method 4	43.8	12.6	43.3	13.5	52.2	18.7	108.6	40.4	202.5	79.5
Method 5	43.8	4.1	43.3	7.4	69.0	14.7	906.3	66.8	-----	----
Method 6	10.1	6.7	42.7	8.0	93.2	11.2	244.8	23.8	497.4	46.2

Noise given in percent of Pmax - Pmin. Deviation given in percent of P'max - P'min.

Table 2-1: Results for methods 1 to 6.

There is little difference between the average error ranges given by method 1 and method 3 (cf Table 2-1). In the other hand, method 1 gives better results in terms of maximum error ranges, and this is confirmed by the plots, which show that the deviation for the early and late parts of the curves is more important with method 3. Thus the first method seems to be the best (among the ones considered in this report). Method 3 has also a particular characteristic: it gives more continuous curves than method 1 (the derivative is computed on an interval, instead of being computed on three points), and this leads to some kind of periodical behavior (Figure 2-3 and 2-5). This could become a real drawback for wells that have a real periodical trend (for example tidal effects), because the two periodical effects could interfere.

The next section will discuss the effect of the derivation interval length on the computed derivatives. Results will be given for both methods 1 and 3, although method 1 has been described as the best.

2.1.2.2. Derivation Interval Length

The derivatives have been computed, for several ranges of noise, with five different interval lengths: 0, 0.2, 0.5, 1 and 2 decimal log

cycles. Figure 2-5 shows the results for both methods 1 and 3, with a range of noise equal to 5% of the initial data range. Only curves corresponding to the intervals of 0.2, 0.5 and 1 log cycle have been plotted, because the other ones have too important errors (see Table 2-2 and 2-3). The results for large intervals (0.5 and 1 log cycle) could be considered as acceptable, even if the points obtained with method 1 and an interval of 0.5 are very dispersed: the main trend still appears. Here again, the curves obtained with method 3 are by far more continuous than the ones obtained with method 1. This does not mean that the results obtained with this method are better, since the deviation is still very important at the ends of the curves. The pseudo-periodical trend is still present too.

Noise (%)	0.0		1.0		2.0		5.0		10.0	
Deviation	Max D	Av. D	Max D	Av. D	Max D	Av. D	Max D	Av. D	Max D	Av. D
I = 0.0	0.1	0.0	71.1	24.4	142.1	48.9	355.1	122.2	710.1	244.5
I = 0.2	3.4	0.8	14.6	5.4	28.7	10.6	71.9	26.3	143.8	52.5
I = 0.5	13.1	3.4	15.8	5.1	20.3	7.3	39.3	14.9	72.0	28.0
I = 1.0	35.6	10.4	35.3	10.8	37.9	11.6	45.9	14.5	59.2	20.9
I = 2.0	59.4	23.6	58.3	23.8	57.4	23.9	56.5	24.6	63.4	25.2

Noise given in percent of Pmax - Pmin. Deviation given in percent of P'max - P'min. I given in decimal log cycle.

Table 2-2: Different interval lengths used with method 1.

Noise (%)	0.0		1.0		2.0		5.0		10.0	
Deviation	Max D	Av. D	Max D	Av. D	Max D	Av. D	Max D	Av. D	Max D	Av. D
I = 0.0	0.5	0.1	71.5	24.9	144.3	51.7	-----	170.4	-----	----
I = 0.2	3.4	0.7	53.4	5.0	107.1	9.5	267.7	23.5	535.4	53.7
I = 0.5	9.3	2.2	53.4	4.4	107.1	6.7	267.7	14.0	535.4	29.0
I = 1.0	25.6	6.5	53.4	8.6	107.1	10.7	267.7	16.7	535.4	29.3
I = 2.0	48.2	11.5	53.4	13.7	107.1	15.7	267.7	21.6	535.4	33.0

Noise given in percent of Pmax - Pmin. Deviation given in percent of P'max - P'min. I given in decimal log cycle.

Table 2-3: Different interval lengths used with method 3.

One can draw an other interesting conclusion from the different tables: for a given interval length, the error range seems to be directly proportional to the noise range. This is obvious for small intervals, and it is probably the same for larger ones, but the ranges

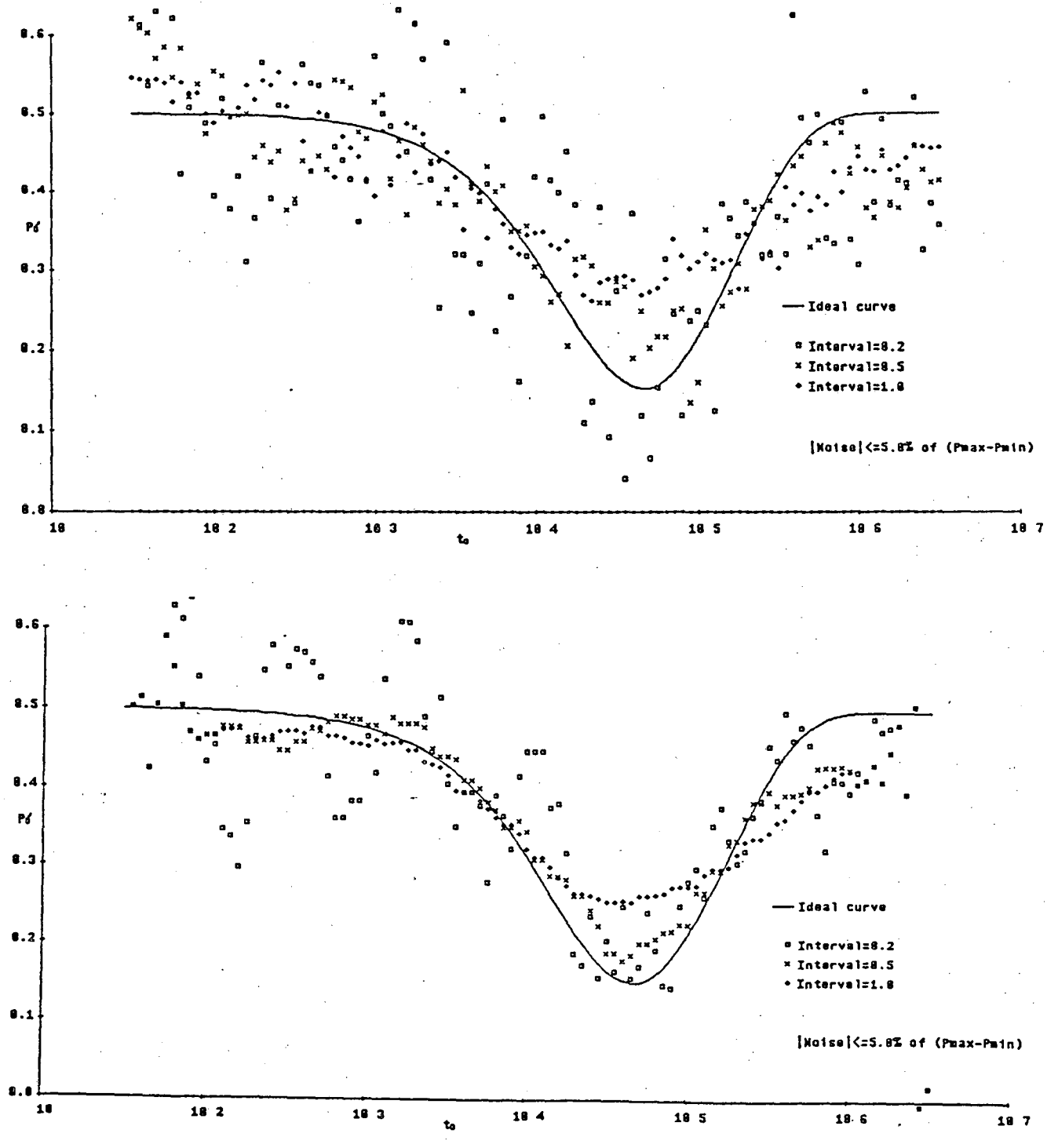


Figure 2-5: Influence of interval length for method 1 (top) and method 3 (bottom).

of noise are not enough big to show it. For large intervals, the error range is almost constant for small values of noise, and then begins to increase when the noise range reaches a given value. This could be a very interesting characteristic to use in WES: if one can estimate the noise range for the curve under study, one might be able to find the best interval length for computing the derivative of this curve.

All the graphic results obtained for this problem are shown in Appendix A.

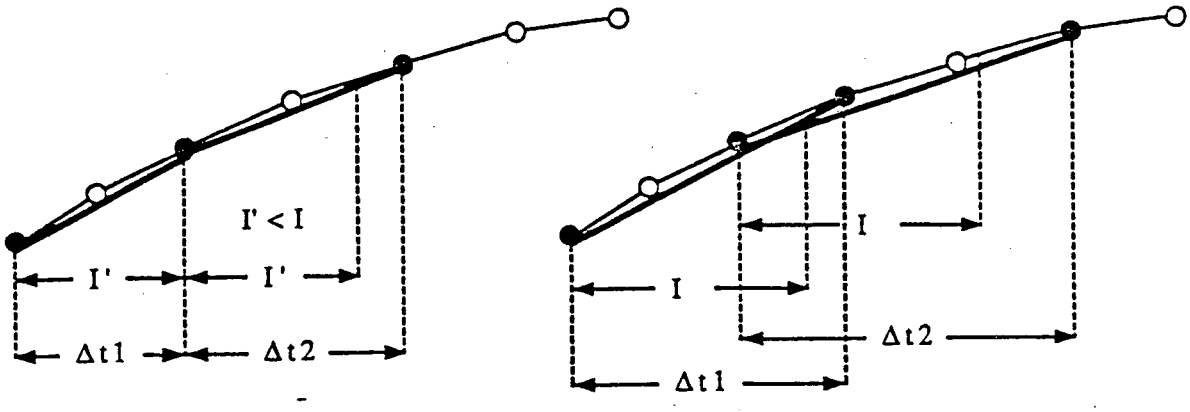
2.1.2.3. Methods For Computing Early And Late Values

The values of the derivative corresponding to the early and late parts of the data set are very important for the analysis. Therefore these values must be computed in a way that minimizes error ranges. Two algorithms have been tested to study this problem. Both of them use a derivation interval of 0.5 natural log cycle (0.21 decimal log cycle) and are based on Method 1 (see 2.1.2.1.). The only difference appears in the way the derivation interval is computed: in the first algorithm, the length of the interval is reduced when the point under study is near the extremity. Basically, on one side of the point of interest, the interval is defined by this point and the first (or last) point, and on the other side, the symmetrical interval is taken. In the second method, the length of the interval is kept constant: on the internal side of the point of interest, the interval is computed as usual, and on the other side, the interval is defined by the first (or last) point, and by the first point farther than the given length. Graphic explanation and results are shown on Figure 2-6.

The second algorithm seems to give the best results, and is the method that will be used in the future (in its present state, the system uses the first algorithm).

2.1.2.4. Influence Of The Superposition Function

As explained in the first part, the superposition function is used in the case of a buildup. It is a modification of the log of time



Derivation intervals for method 1. Derivation intervals for method 2.

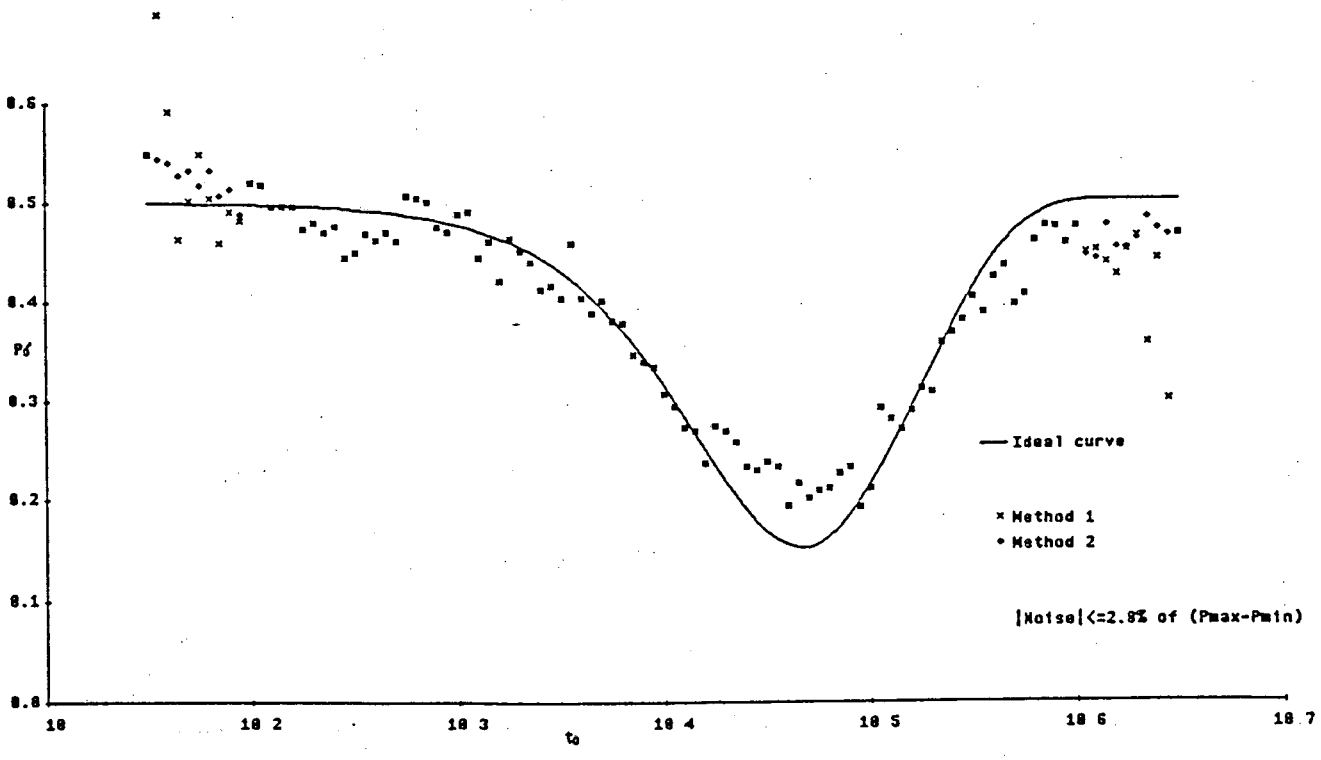


Figure 2-6: Methods for computing derivative at curve extremities (top). Results (bottom).

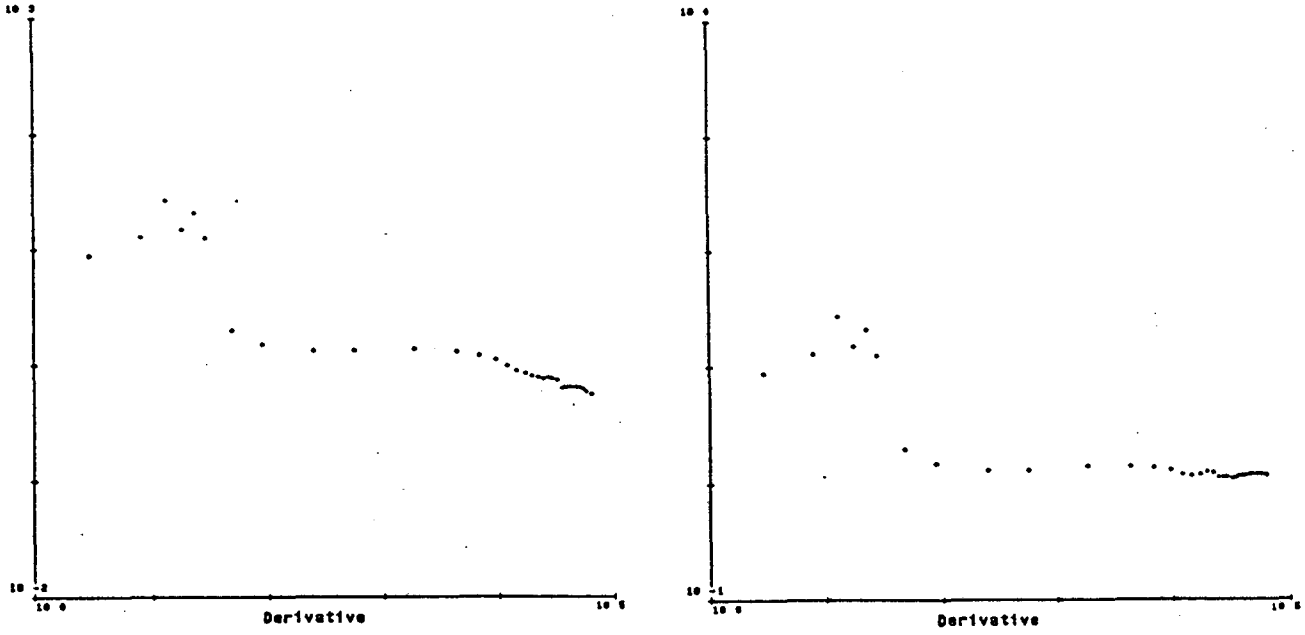


Figure 2-7: Influence of the superposition function on a buildup well-test. Results obtained with (right) and without (left) the use of the superposition function.

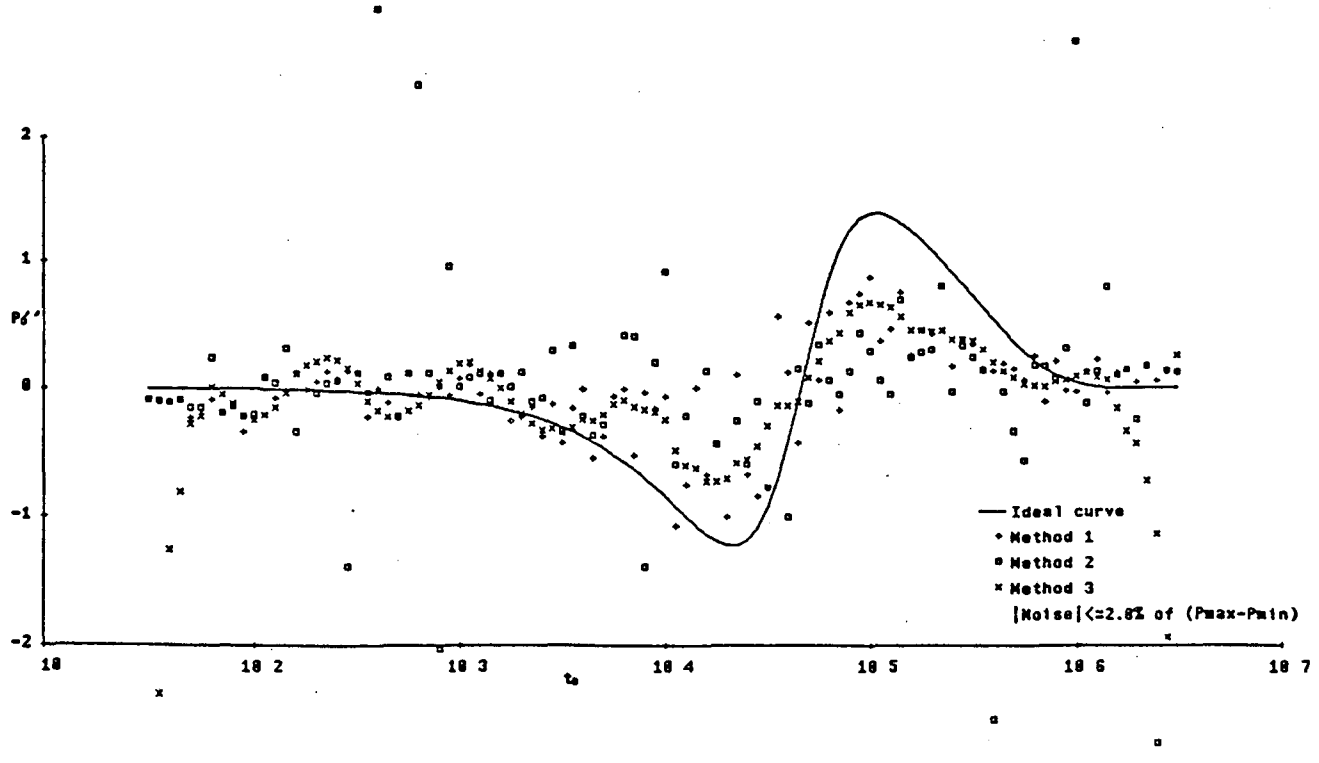


Figure 2-8: Second order derivative for methods 1 to 3. Derivation interval length: 0.215. Noise range: 2%.

function, which is used to keep in consideration the flow history of the well. Its main advantage is to restore the semilog straight line for a homogeneous model. When computed with respect to the superposition function, the derivative also keeps its characteristic horizontal straight line. This is shown on Figure 2-7 for a real data set that is currently used by the system. The time axis would have been modified in real analysis, but in the present case it makes no difference, since the straight line is horizontal (the superposition function compresses the late part of the time axis).

2.1.2.5. Validity Of The Second Order Derivative

In the current state of the program, the second order derivative is used to compute the slopes on the derivative plot. These slopes are then used by the program during the patterns recognition process. As seen above, the derivative algorithms are very sensitive to random noise. The different tables show that for a noise on the initial data of 2% range, errors of 10% range are usually obtained. This means that computing the second order derivative could lead to huge error ranges (50%, according to the tables).

The second order derivative has thus been studied for the curves that have been used in the preceding parts. The curve obtained for an interval length of 0.5 natural log cycle and a noise range of 2% is given in Figure 2-8. The three methods used here were the ones described in section 2.1.2.1. The results are not as bad as expected, but are still hardly usable. Moreover, second order derivatives computed for different kinds of curves show much more important error ranges. Note that the pseudo-periodical trend obtained with method 3 is obvious on this plot.

The second order derivative seems to be very noisy, as soon as small random noises appear on the initial data. This means that using it in the program is a very risky method, and should be avoided if possible. Since the shapes recognition algorithms are based on this

second order derivative, new algorithms have to be proposed to overcome this difficulty. Next section describes these algorithms.

2.1.3. SHAPES RECOGNITION

In the current state of the system, two main sorts of shapes are looked for: humps and straight lines (See 1.2.3.3.). The concept of hump is heavily based on the second order derivative, since humps describe the curvature of the curves. In the other hand, humps could be represented in first approximation by two straight lines, one going up followed by another one going down (for a hill hump).

The advantage of using only straight lines is that straight lines can be computed without the need of the second order derivative: least-squares methods (or linear regression) give good results using only points, and not the slopes at those points. These methods have been studied with the real data sets presently in use in the system, and the results are given in the following sections.

2.1.3.1. Algorithms

The algorithm described above explains how the straight lines are computed with the least-squares approach. Basically, two different methods can be used: the first is the real least-squares method, where distances from points to lines are computed using the euclidian way. The second is a linear regression method, where distances are considered on the y axis only. This difference does not appear in the main algorithm, since only values are modified. More details about the difference between these two methods are given in Appendix B. In the description of the algorithm, some formal functions are used: *interval()* returns the interval *I* defined by the first points on both sides of the point of interest which are at least at *length* log cycle from this point. *Line()* returns the values of the slope, the intercept and the quality factor (which is the length of the line over the standard deviation) of the computed straight line. This straight line is computed from all the points that are in the interval *I*.

The values of the straight line characteristics is the only part of the algorithm which is dependant on the method (least-squares or linear regression). The notations are [ ] for an interval, and {a, b, q} for a straight line, where a is the slope, b the intercept and q the quality factor. i is the initial point for the computation of each straight line.

```
        i = 1
1       if i = end then stop
        I = interval(i, length) (= [tii, tfi])
        line = line(I) (= {a, b, q})
2       I1 = [tii, tfi+1]
        line1 = line(I1)
        if q1 > q then I = I1 (i.e. fi = fi+1)
              line = line1 (i.e. a = a1 ; b = b1 ; q = q1)
        I2 = [tii-1, tfi]
        line2 = line(I2)
        if q2 > q then I = I2
              line =line2
        if q > q1 and q > q2 then
              return line
              i = fi
              goto 1
        else goto 2
```

Basically, this algorithm selects an interval (I) of a given length and computes the straight line corresponding to this interval. Then it extends the interval to the right (I_1) and to the left (I_2) and compares the new quality factors with the old one. While at least one of them is better, the algorithm repeats this step. When the quality factor decreases for both sides, the program returns the lines. The new point of interest, that is, the central point of the new interval, is defined as the last point of the preceding interval. This means that two consecutive straight lines are defined on overlapping intervals: this characteristic gives a good continuity to the straight lines computed by this method.

Once the lines are computed, the graphic representation is obtained by computing the intersection points for each group of two consecutive straight lines. Figure 2-9 shows examples of the results.

Advantages of this method are:

- No use of the second order derivative.
- No use of arbitrary error bounds.
- Smoothing of the curve (least-squares methods are well-known for their smoothing effects).
- Obtainment of a quality factor for each straight line. This quality factor might be used in the following of the analysis.

The main drawback is that this method does not allow to describe curves in terms of humps. Thus a new knowledge base will have to be built to deal with the new description of the curve.

2.1.3.2. Results

Two main aspects are studied in this section: one is the selection of the best method (either real least-squares or linear regression), the other is the influence of the computation interval length (*length* in the algorithm above) on the results. The combination of derivative smoothing and least-squares smoothing is also described.

Figure 2-9a shows the straight lines that have been computed for a real data set. The derivative is here extremely noisy, but it shows that the smoothing effect of the linear regression method is by far more important than the one obtained with real least-squares. This is still true with less extreme cases.

In the data set in use here, the late part of the curve can be considered as bad data (the pressure derivative becomes positive, what is theoretically impossible). The linear regression method gives for the few last straight lines very bad quality factors: this is a very interesting point and could probably be used in the analysis to give less importance to that part of the data. Note that the difference between quality factors for good and bad data is less important with the real least-squares method.

Figure 2-9b and 2-9c show the influence of the length of the computation interval on the straight lines. This influence is obvious when the curve is very noisy (Figure 2-9b). In that particular case

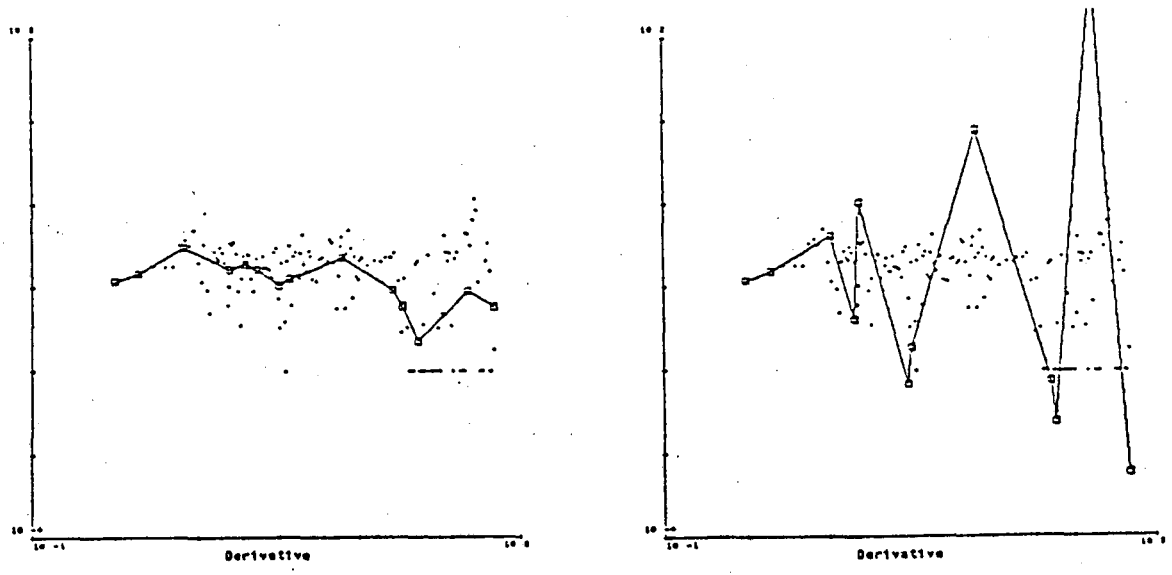


Figure 2-9a: Difference between least-squares (right) and linear regression (left).

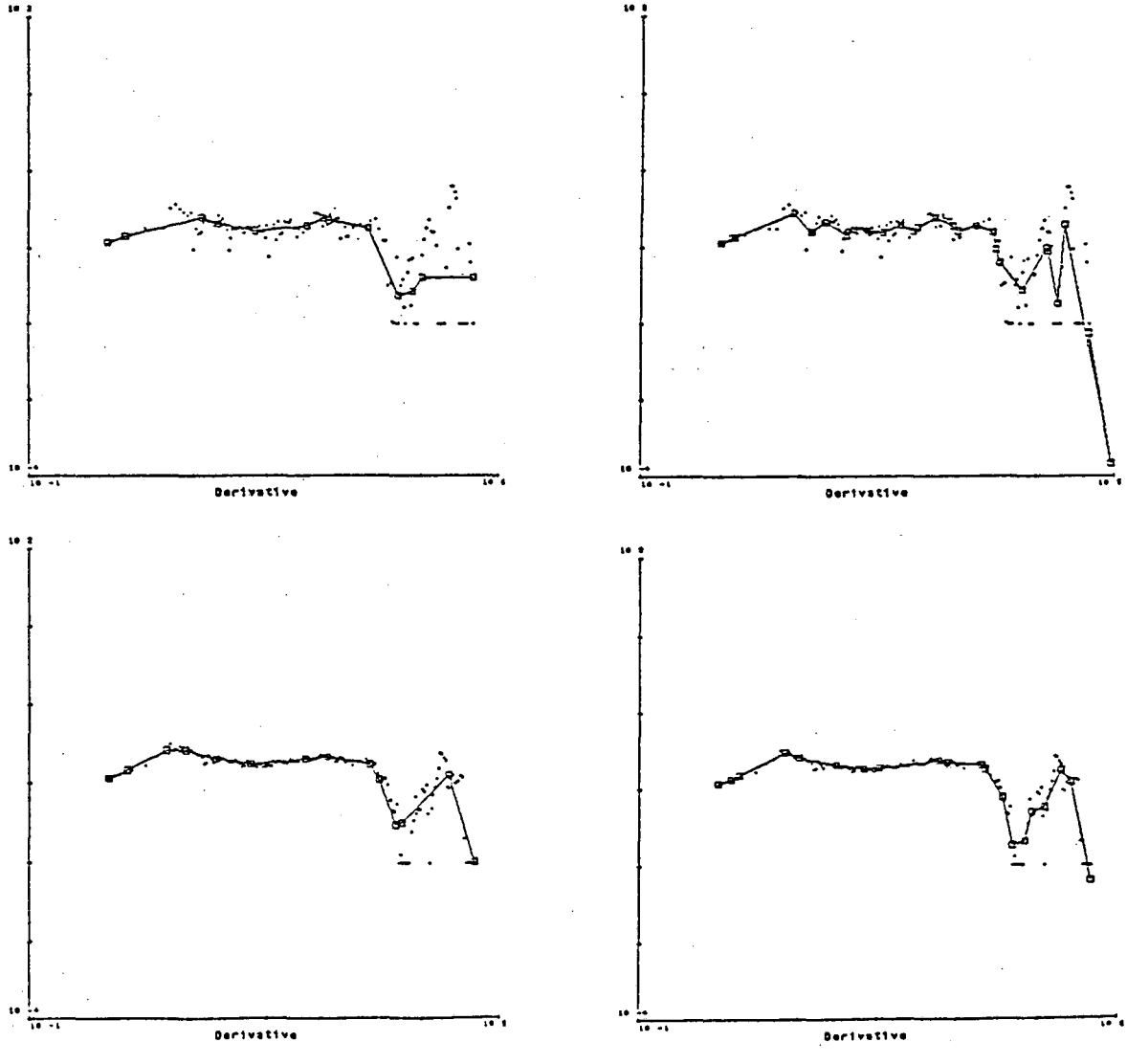


Figure 2-9b and c: Influence of the computation interval. Left: 0.2. Right: 0.1. Top: Noisy data. Bottom: Smoothed data.

the derivative was computed with a small interval (0.05 decimal log cycle). When the same derivative is computed with an interval of 0.2, Figure 2-9c is obtained. It shows that the interval length has by far less importance when the curve is regular enough. More curves about this problem are given in Appendix A.

The interval length for straight lines computation seems to have less influence than the derivation interval length. Obviously, too large intervals can lead to a loss of information, and too small intervals give too numerous straight lines that are hardly usable. An average length of 0.2 decimal log cycle (on each side of the initial point) seems to give good results for the data sets currently used by the program.

2.1.3.3. Possible Use Of Expertise In Shapes Recognition

Until now, all that has been described in the field of shapes recognition has been purely numerical. This numerical part is essential, since the initial data is numerical too. The method described above returns a small number of straight lines (usually between ten and twenty lines). An expert system can deal much more easily with these "symbolic" informations than with a data set of two hundreds points. Moreover, the linear regression gives a quality factor for each straight line. This quality factor could be used by the system to give more or less importance to such and such straight line.

Nothing has been really done in that field for the moment. Moreover, use of expertise in shapes recognition must be in relation with the real expert part of the system. Basically, once a part of the analysis has been done (for instance, for the early part of the data), the system might be able to decide if a specific straight line (found either in the same part or in the other part of the curve) is compatible or not with the analysis already done.

2.2. WELL-TEST ANALYSIS IMPROVEMENT

2.2.1. INTRODUCTION

This part has been almost not developed during the last two months (i.e. the period of my internship), since a lot of things had to be fixed in the numerical field (see 2.1.) and since part of the system has been translated in C (see 2.3.) during this period. Thus this chapter describes only type curves computation and some problems that appear with the data sets currently in use by the program.

This part is the one that is directly relevant to expert systems. In that way, this is probably the most interesting part of the whole project. It will probably become in the future the most important part of the system. The following sections describe some "tools" that will be useful for improving the analysis.

2.2.2. TYPE CURVES GENERATION

Type curves are used by human experts to refine their analysis, once they have found a first estimate of the reservoir model. Since this step is not already done in the present state of the system (the system gives sometimes the right result, but this is not always true), type curves will not be useful in the near future. However, two sets of type curves have been generated, using two different ways. The methods and results are described in the following sections. A third section will describe an other way to compute type curves.

The different tables and equations used in type curves generation have been found in different papers on well-test analysis, most of them published in the "Journal of Petroleum Technology" and in "World Oil".

2.2.2.1. Wellbore Storage Type Curves - Use Of Tables

The wellbore storage type curves are probably the most useful type curves used in well-test analysis, since wellbore storage appears in the large majority of the test. The curves actually generated depend on only one parameter, the wellbore storage group, $C_D e^{2S}$.

Exact solution for these curves is known in the Laplace domain, but there is no valid approximation in the real domain. Therefore the curves have been generated from tables found in a technical paper[6]. Since this paper has been published in 1970, the derivative type curves are not described in it. The discrete method to compute these derivative curves is method 5, with the notations of 2.1.2.1. It gives the most regular results for the curves (Method 1, which is usually used to compute derivatives, gives small periodical oscillations at the beginning of the curves).

The tables used yield, for a given value of the skin factor S, the dimensionless pressure p_D as a function of the dimensionless time t_D and of the wellbore storage constant C_D. It means that the tables depend on two parameters, S and C_D. The theory shows that these two parameters can be grouped in only one, the wellbore storage group $C_D e^{2S}$. Thus a new table has been computed from the five initial tables, and yields the dimensionless pressure as a function of t_D/C_D and $C_D e^{2S}$. Since the values of the wellbore storage group were not round values, interpolation has been used to obtain $C_D e^{2S}$ as powers of ten.

The table finally contains pressure versus time values for thirteen values of $C_D e^{2S}$. The main drawback is the too small number of time values, that is, the points are not numerous enough to give a regular aspect to the curves. The same problem appears for the derivative type curves. Polynomial interpolation has been used to smooth these curves.

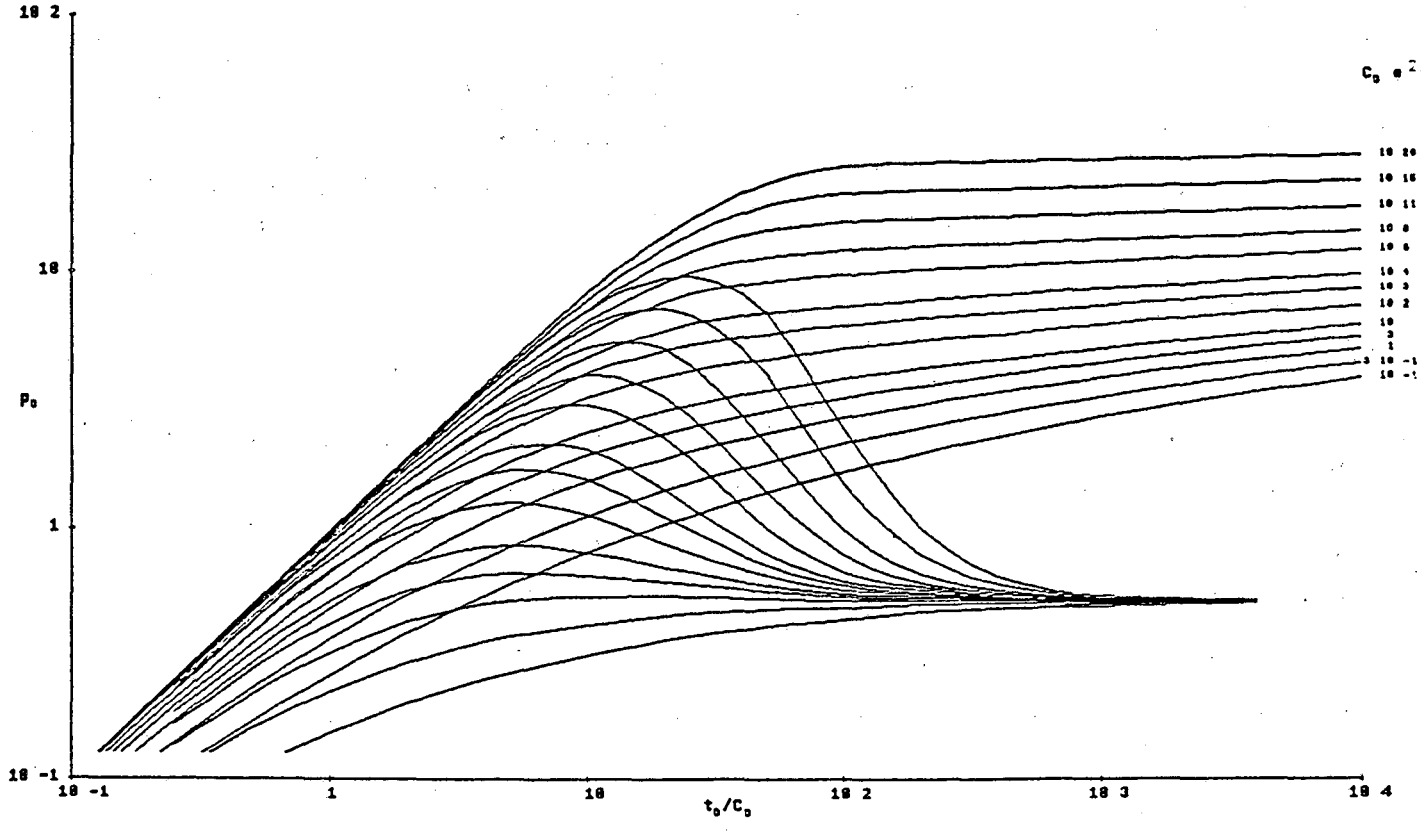


Figure 2-10: Combined derivative and pressure type curves.

Figure 2-10 shows the combined log-log and derivative plots of the type curves.

2.2.2.2. Double Porosity Type Curves - Use Of The Asymptotic Solution

The generation of double porosity type curves uses an other technic and is based on the asymptotic solution given for these curves[5]. Here again, the exact solution is known in the Laplace domain, but in the present case the real approximation is valid on a large interval. The asymptotic solution is given by

$$p(\tau) = 1/2\ \{ln\tau + 0.80908 + E_i[-\lambda\tau/\omega(1-\omega)] - E_i[-\lambda\tau/(1-\omega)]\}$$

where p is the dimensionless pressure, τ the dimensionless time, λ the interporosity flow coefficient, and ω the storativity ratio. E_i is given by

$$-E_i(-x) = \int_x^{\infty} \frac{e^{-u}}{u} du$$

In this case, the analytical solution for the derivative type curves is easily obtained from the asymptotic solution. However, the type curves shown in Figure 2-11b have been computed from a discrete set of points on the semilog type curves using Method 1 (see notations in 2.1.2.1.). It seems better to use the same method to compute real and type curves, to reproduce the systematical error that appears using discrete derivation. This is true only if the systematical error always gives the same distortion. Analytical type curves could be very easily computed if needed.

Figure 2-11a shows the semilog type curves (in this case, the log-log type curves have no characteristic shapes) and Figure 2-11b the log-log plot of the derivative type curves.

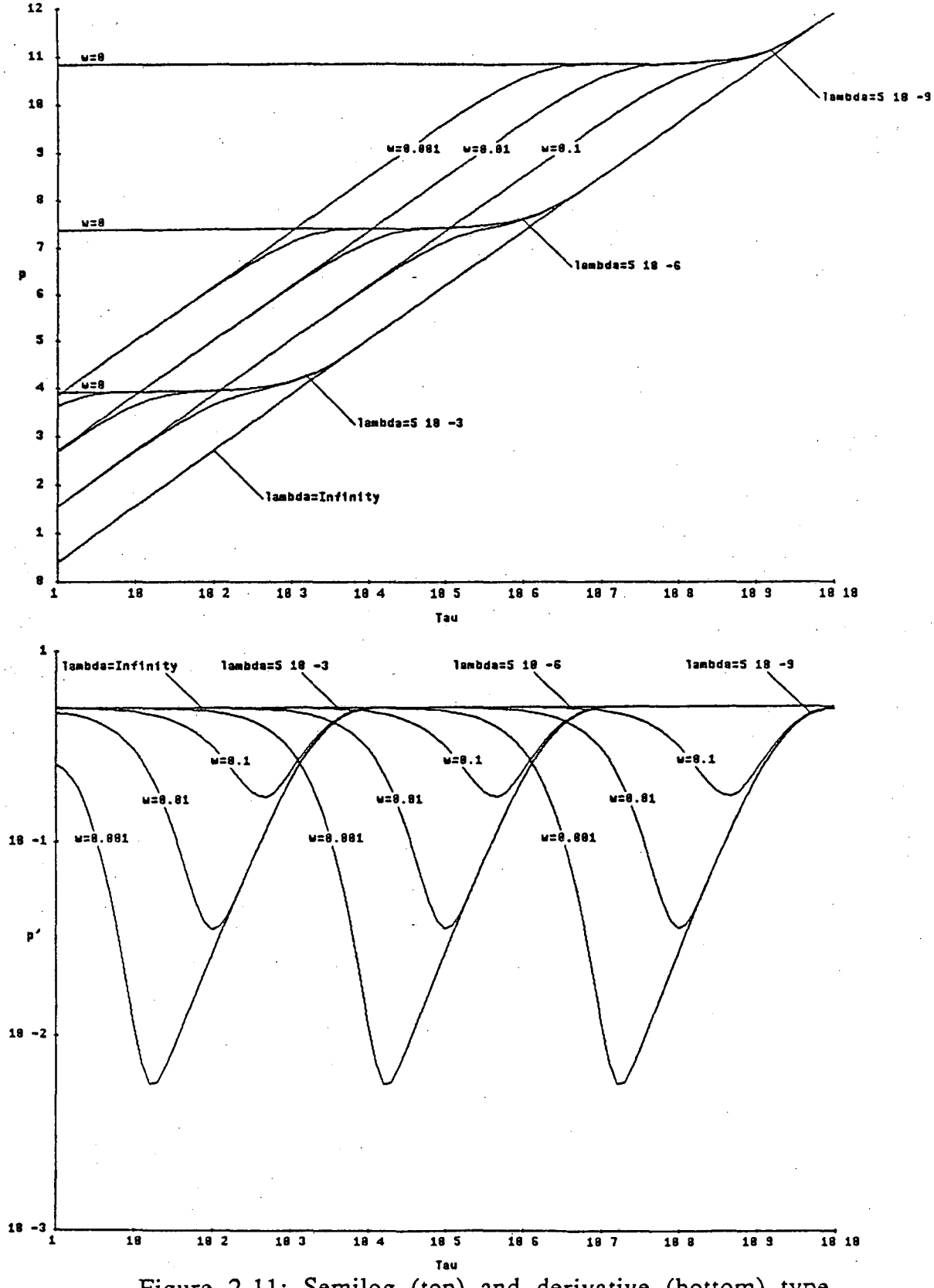


Figure 2-11: Semilog (top) and derivative (bottom) type curves for the double porosity model.

2.2.2.3. Use Of The Laplace Inversion

As seen above, results given by use of tables are not excellent. Since the exact solution is known for a large majority of type curves in the Laplace domain, use of Laplace inversion algorithms should be considered. This has not been done yet, but the algorithm that will be probably used in the future is the one described by H. Stehfest[12], which is based on a probabilistic approach. A more complicated one, using complex numbers, will also be studied.

This method is certainly time-consuming, but will probably lead to better results. Moreover, some modifications could be added in the Laplace equations to model different variations from the basic model, such as buildup analysis or atmospheric pressure changes. Using the inversion algorithm, type curves corresponding to these variations might be easily computed.

2.2.3. REAL DATA SETS

The characteristics of the data sets used by the program during its development have a considerable importance: the rules are created and modified to give the right results for at least the facts contained in the database. In the case of well-test analysis, some of the initial data sets must have at least enough characteristic shapes to be easily recognize by an expert.

This is not true for the data sets used by the system until now. Many of the well-tests show huge variations from the basic model to which they belong: geothermal effects, atmospheric pressure changes, bad data, etc... often appear for most of the well-tests. Thus recognition needs a lot of very specific rules to be achieved. This is probably not the right method to build a knowledge base: it is widely accepted that expert systems should be developed going from the general to the specific.

Therefore a new method is proposed: first, generation of ideal solutions for the different models, using type curves equations or tables. A first very simple knowledge base will be created using these ideal solutions. Second, superposition of different kinds of noise on the ideal solutions, to model the different problems that can be encountered during a well-test (random noise, flow rate changes, atmospheric pressure changes, geothermal effects, tidal effects, etc...). The knowledge base will be modified to take care of these different noises. Third and last, real data sets will be introduced to test the validity of the knowledge base. The different noises that will appear on these well-tests might be diagnosed, and perhaps corrected, with the help of the preceding step.

This method has not been used yet. The very next step that will be done in the development of the system will be the generation of the ideal curves for the different models. This needs a lot of bibliography research, and therefore takes a lot of time.

Few things have been done in the field of well-test analysis, which is the real expert part of the system. The main reason is that most of the time has been spent to improve the numerical algorithms and to translate the graphic interface from ART and LISP to C. This particular part is described in the next section.

2.3. CONVERSION OF THE PROGRAM

2.3.1. INTRODUCTION

The problem of the translation of the system from the ART language to a lower level language such as LISP or C was first considered in the long term. But the limits of the expert system shell has appeared sooner than expected: the system is dramatically slowed down when too many facts are present in the database. Since most of the facts used by WES are not directly relevant to the expert part of the system, one can think of keeping the memory space for facts used in the analysis and using other means to program the user interface and the numerical part of the program.

The main problem is to realize a good interface between the ART shell and the program that will be used for the user interface (since the user must have access to the facts and results contained in the knowledge base). The first possibility is to use LISP, since ART and LISP are very close to each other (ART is written in LISP). In the other hand, ART provides the possibility of running a program through the Unix shell. Moreover, a data stream can be opened between the ART shell and the Unix program, so both programs can exchange informations, while they are running in parallel.

This part of the project represents at least two thirds of the work that has already been done, but is very difficult to comment without getting into programming details. Therefore only a quick overview of the program will be given in the next sections.

2.3.2. THE C PROGRAM

C has been chosen for two main reasons: first it is faster than LISP for computation and screen management, second window-

oriented C libraries are provided for the SUN (WES is run on a SUN workstation). The main drawback of C is that it is more difficult to interface with ART than LISP.

The program is written to reproduce exactly the behavior of the version of WES that has been described in section 1.2. Therefore it still uses the concept of hump and the second order derivative (the conversion of the program is the very first thing that has been done). It can be divided in three parts: the first one is the graphic-oriented user interface, the second one deals with the numerical computations (curves and shapes), and the last one is the interface between the C program and the ART shell.

Graphics represent the main part of the program. As for the ART version, one window is associated with each well under study. In each window, five different curves can be plotted (initial, filtered, semilog, log-log and derivative plots). The user asks for the one he wants through a menu. An other menu is used to select the wells that will be analyzed. This part of the program has been written with the low-level functions provided by the Sunview package.

An other important part of the program is the computation of the curves, the derivatives and the characteristic shapes. This part was written in LISP in the original program, and therefore was very slow. The translation to C has improved widely the speed of the program. Shapes are still computed by a numerical way, since the program reproduces exactly the behavior of the original one. It has been explained in the preceding sections that the method currently in use is probably not the best. Therefore this part is likely to change within the few next weeks.

Interface with the ART expert system shell is provided in the last part of the program. The functions contained in this part send to ART all the data it needs to achieve the expertise: straight lines and humps, values of slopes at the beginning and at the end of the curves, etc... In the present state, the stream is only directed from

the C program to ART. If needed, data could easily be sent from ART to C.

No more details will be given about this program: this part is of no interest from the expert system point of view. It needed to be done, and has been done. Listings are given in Appendix C. Figure 2-12 shows the screen obtained during the run.

2.3.3. THE REMAINING ART PROGRAM

The existing program has been rather shortened, since more than half of it has been rewritten in C. The only parts which remain written in ART are the definition of the objects (see 1.2.2) and the expert part, which extracts the model from the patterns of the curves. These parts are almost unchanged, only a few rules have been added to read the data on the stream which links ART and the C program.

As the Unix environment on the SUN allows to run programs in parallel, the ART part of the system is able to make the analysis of a well while the C part is computing the shapes for an other one. Therefore none of the system possibilities to analyze more than one well simultaneously has been lost.

This part will also probably change in the future, since most of the rules are based on humps and values of slopes. The existing rules can probably be easily modified to use the new representation of the curves in terms of straight lines: a hump can be represented in first approximation by two straight lines of different directions. It can be described more precisely by a group of straight lines whose slopes increase or decrease.

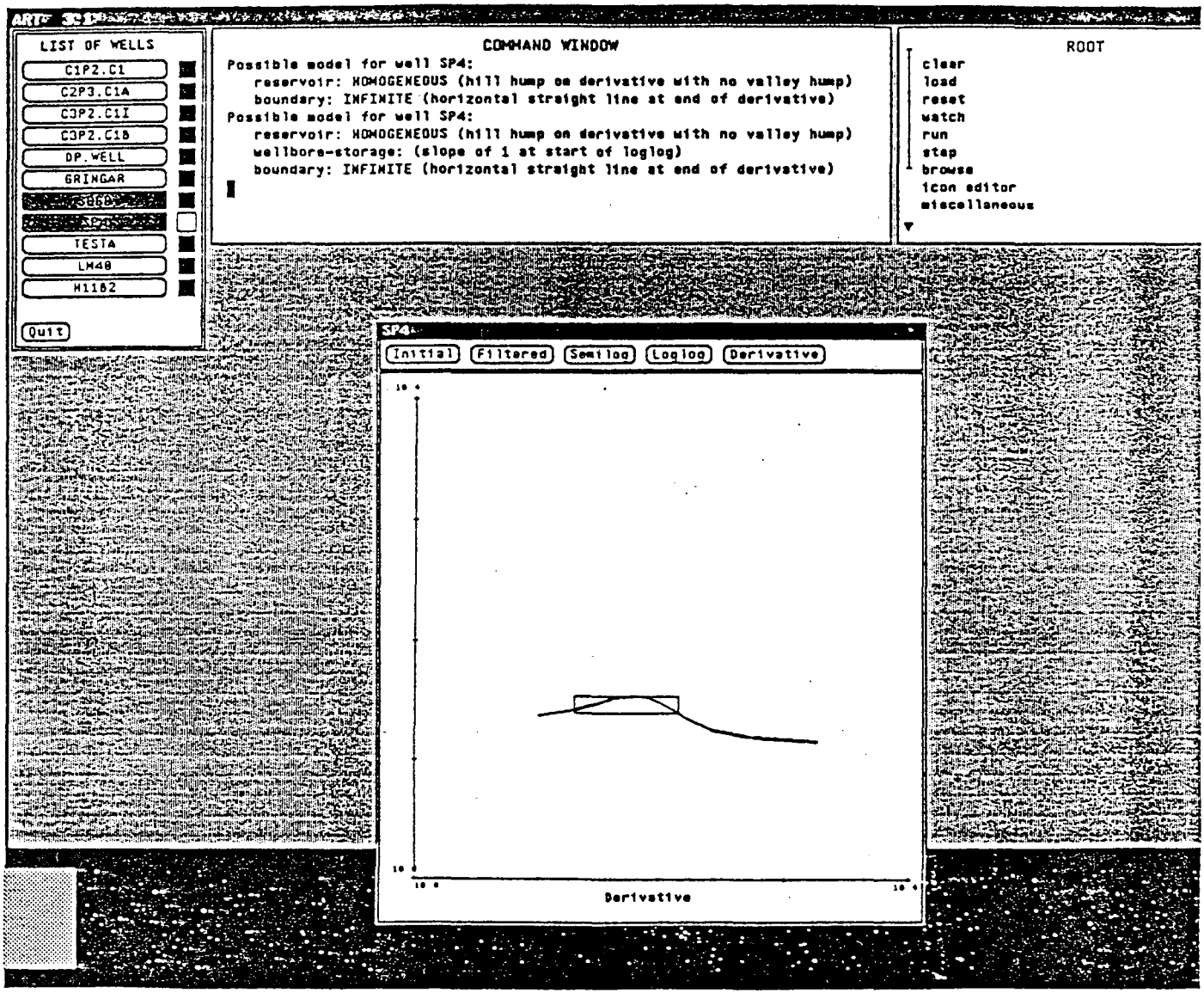


Figure 2-12: Screen obtained during the run. The windows "List of wells" and "Sp4" belong to the C program. The other ones art part of the ART shell.

PART III - CONCLUSIONS AND EXTENSIONS

3.1. CONCLUSIONS

This section describes the conclusions that can be drawn from the results given in the above sections. Since research has been done in very narrow and separated fields, this synthesis is difficult to do. Moreover, almost none of those fields is related to Expert Systems, which were supposed to be the subject of this project. One have to consider this work as a preliminary or as a realization of tools useful for the following of the whole project. I would like to insist on the importance of the program conversion,which has taken a lot of time and was really necessary: the ART expert system shell was no longer able to support the huge number of facts used by the graphic interface. This part is certainly not spectacular, since it consists of the copy of an existing program, but still has two important effects: the execution speed has been multiplied by about five times, and more than 70% of the memory space has been freed in the expert system shell.

The next sections draw conclusions for the different parts of the research that has been done. Since none of the methods described above are currently in use, these conclusions are only expectations. They will be probably modified in the future, when new difficulties will appear. However, they represent a new step in the realization of the whole project.

3.1.1. NUMERICAL ALGORITHMS

Results on derivative and straight lines computation have been obtained. About derivatives, two algorithms have been shown as the best among the ones that have been tested: methods 1 and 3, with the notations used in 2.1.2.1. Method 1 has been preferred because it gives better results at the extremities of the curves, which are of great importance in the analysis. If method 3 can be improved to

give better results for those parts of the curves, it can probably be used successfully too. This method gives more continuous results than method 1, and this characteristic can probably be useful in some cases. In the other hand, it can lead to analysis errors, since continuity gives a pseudo-periodical trend to the curves. In the short term, method 1 will be used to compute derivatives in the program.

An other interesting result about derivatives has been given, concerning the second order derivative: it has been shown that the error range on this derivative can reach large values, even for a small range of noise on the initial curve. Since this error range is sometimes twice as big as the error bounds used in the existing program for shapes recognition, the validity of the method currently in use is doubtful. Therefore a new method for shapes recognition that no longer uses the second order derivative has been proposed.

This method is basically a least-squares algorithm, or more precisely a variation of it, called linear regression. It gives good results for the data sets currently used by the program, but an important question remains: the new representation of the curves in terms of straight lines can not be used by the rules already written. Therefore its possibility of use in well-test analysis has not been tested yet.

A new knowledge base will be written to accommodate this new representation. The main difference between this database and the existing one will be the disappearance of the hump concept, which is based on the second order derivative, Since this concept is a very important one, it will be replaced in the new representation by a group of two straight lines, one going up followed by another going down (in the case of a hill). If it appears that the concept of curvature is needed, this curvature will be represented by a group of straight lines whose slopes increase or decrease. Using these representations, the transformation of the existing database should not be too difficult.

The results described here do not give new possibilities to the system, since derivative computation and patterns recognition were already done in the existing program. They should be considered as improvements of the existing algorithms, whose results were not always reliable enough. A high attention has been given to these numerical problems, since the rest of the analysis is based on this part. New possibilities of improvement will still be considered during the development of the system.

3.1.2. WELL-TEST ANALYSIS

Very few things have been done in that field. The main reason is that other parts of the system needed more urgent work, such as modifications of the numerical algorithms or translation of the graphic interface. This part still is the most important in the program, since it is the one which contains the knowledge base of the system. It will be developed in the future, once all the numerical tools it needs will be created.

Parts of those tools are the type curves for the different models the system uses. Two sets of type curves have been generated, using two different methods. The program does not use these type curves yet, since the results it gives for the model diagnosis are generally too bad to be useful. Otherwise, the type curves generation has had important "side effects", since it has leaded to the study of the derivative problems and to the translation of the program.

The two ways used to compute type curves are use of tables and asymptotic solutions. The tables usually give poor results, but are often the only simple way to generate type curves, since a large majority of them have no real solutions, and even no asymptotic approximations. If the system requires more accurate results (this problem has not appeared yet, since the type curves have never been used by the program), an other method will be tested: the exact solution for many of the type curves is known in the Laplace domain, therefore a Laplace inversion algorithm can probably give good

results in type curves generation. Some algorithms have been found in papers, but none of them have been tested yet.

Analytical data sets are also tools that seem to be needed by the program: the real data sets currently in use show too many specific characteristics and therefore can not be successfully analyzed. The generation of these new data sets has not been done yet, but is strongly related to the type curves problem. Here again, Laplace inversion algorithm will probably be useful.

3.1.3. PROGRAM CONVERSION

As explained above, this part takes most of the time I have spent on the project. There is no conclusions to draw from it, since it consists of the exact translation of the original program. It will now be modified to contain the new algorithms described above. These modifications should be easy to do, since graphics and computations have been kept separate in the C program.

This might be considered as a first step in the final conversion of the system to a low level language. This translation will have to be done if the system must be used in real time, that is, if the analysis of a well must be done during the test. In this case, ART and the SUN workstation would probably be considered as too expensive to be used on each well. From this point of view, the main drawback of the current C program is that it is absolutely not standard: all the graphic functions use the Sunview package, which is specific to the SUN workstation.

3.2. EXTENSIONS

3.2.1 SHORT TERM DEVELOPMENTS

3.2.1.1. Use Of Theoretical Data

The very next step in the development of the system will consist of putting together the solutions that have been given in this report. This step will be achieved by:

- generating new analytical data sets and corresponding type curves.
- replacing the numerical algorithms in the current C program by the new ones.
- writing the corresponding knowledge base to extract the model from the new straight lines representation.
- using type curves to estimate the parameters of the different data sets.

The problems of curves generation have been described several times in this report. This part needs a lot of bibliographical researches to find the equations corresponding to the models in use in the program. Since most of those equations are given in the Laplace domain, Laplace inversion will probably be computed.

Replacing the old numerical algorithms by the new ones will not be difficult, since all these algorithms have been already written in C to be tested. Moreover, the graphic and computation functions are totally separate in the C program, therefore the modification will consist of replacing some numerical functions in one file by other functions already written.

The knowledge base will need more modifications, since many of its rules currently used the concept of hump. Humps are no longer computed with the new least-squares algorithm. If this concept is really an important one (but there are actually no evidences of that),

there are many ways to represent humps with straight lines (see above for details). Note that more expertise will be used to compute the curve representation, since only "raw" straight lines will be computed in C and immediately send to the ART shell: grouping of straight lines and elimination of the "bad" data (with the help of the quality factor) will be done in ART. Thus the program will have the possibility to modify the curve representation during the expertise.

The use of type curves to extract model parameters will be the only really new possibility of the program, since it has never been done until now. This extraction is done by matching the curve under study, whose model has been obtained, to the corresponding set of type curves. The problem of the graphic match is a very interesting one, and can be achieved using several ways, such as linear or non-linear regression, but also less sophisticated methods such as comparing the high of the peaks on the real and the type curves. None of these methods have been studied until now.

3.2.1.2. Application To Real Data

Once the system will be able to recognize the theoretical models corresponding to the analytical data sets, different kinds of noise will be added to these data sets. The knowledge base will then be modified to accommodate the new data, and recognize the abnormal shapes on the curves. The possibility of expertise during the shapes extraction step will probably be very helpful: for example, the system might have the possibility to consider a strange pattern either as part of a model or as noisy data.

Type curves will also be used, and some of them could be modified to take in consideration one specific noise that has been identified on the "real" curve (the curve can not be considered yet as real, since it is a superposition of an analytical curve and a given noise).

The last step will be to introduce real data sets and to test the results given by the knowledge base. Since this database will be

theoretically able to recognize the effects of a specific noise on a given model, noises will be discarded, to keep only the meaningful response of the reservoir. An other way to proceed is to modify rules to take in consideration the effects of each different noise for a given model. In that case, the noise is kept during the analysis, and it is the basic knowledge base that is modified. From this point of view, type curves must also be transformed to take the different noises in consideration.

Once this step will be reached, the system will be considered as a real expert system, even if its possibilities will be limited. Other possible developments are described in the next section.

3.2.2. Extensions To A Larger Problem

In its present state, the system uses only a small set of simple models, like homogeneous or double porosity medium and some boundaries configurations. More models need to be inserted as well as a more complete description of the existing ones. This can be done either during the step of development described above, or once the system will be able to recognize correctly well-tests relevant to the existing models.

In the same respect, the system currently analyzes only single well-tests with a constant flow rate and a drawdown phase. Modifications should be added to accomodate buildups and more generally multiple rate pumping tests. The system should be able to reason about these different phases, compare results between them, decide which phase is more informative than the others, and so on. Another step in the development of the system will be to allow more than one observation well for a single pumping well.

In the long term, the possibility of grouping informations obtained by different methods of analysis could be considered. These methods might involved hydrogeologic informations (this is the object of WES), but also geothermal and geochemical results, and so

on. This can lead to the development of a multi-expert system, using different methods of analysis to solve the same problem. A multi-expert system has the possibility of comparing results obtained by several different ways with more objectivity than a human expert, since a human expert will have more confidence in his own analysis.

An other extension in the long term will be the complete translation of the program to a low level language such as LISP or C. The translation of an expert system involves two problems: first, programming of the inference engine, second, development of a rule compiler, which must be able to translate the rules written in some kind of natural language to the format used by the inference engine. With this method, rules can still be added after the system translation, and thus the program keeps part of the flexibility offered by the expert system shell.

APPENDICES

APPENDIX A - CURVES

TYPE CURVES

- Figure A-1: Combined wellbore storage derivative and pressure type curves.
- Figure A-2: Double porosity semilog type curves.
- Figure A-3: Double porosity derivative type curves.

DERIVATIVE METHODS

Selection Of The Algorithm

- Figure A-4: Methods 1 to 6. Noise = 0%.
- Figure A-5: Methods 1 to 3. Noise = 1%.
- Figure A-6: Methods 1 to 3. Noise = 2%.
- Figure A-7: Methods 1 to 3. Noise = 5%.
- Figure A-8: Methods 1 to 3. Noise = 10%.

Influence Of The Interval Length

- Figure A-9: Method 1. Noise = 0%.
- Figure A-10: Method 1. Noise = 1%.
- Figure A-11: Method 1. Noise = 2%.
- Figure A-12: Method 1. Noise = 5%.
- Figure A-13: Method 1. Noise = 10%.

- Figure A-14: Method 3. Noise = 0%.
- Figure A-15: Method 3. Noise = 1%.
- Figure A-16: Method 3. Noise = 2%.
- Figure A-17: Method 3. Noise = 5%.
- Figure A-18: Method 3. Noise = 10%.

Computation At Extremities

- Figure A-19: Two methods to compute derivatives at curve extremities.

Effect Of The Superposition Function

- Figure A-20: Effect of the superposition function.

Second Order Derivative

- Figure A-21: Methods 1 to 3. Noise = 1%.
- Figure A-22: Methods 1 to 3. Noise =2%.

Real Data Sets

- Figure A-23: Left: Semilog curve. Right: Derivative curve (I=0).
- Figure A-24: Derivative curves. Left: I=0.1. Right: I=0.2.
- Figure A-25: Derivative curves. Left: I=0.5. Right: I=1.

STRAIGHT LINES

- Figure A-26: Difference between real least-squares method (right) and linear regression (left).
- Figure A-27: Combined effects of derivative and least-squares smoothing. Top: Derivative interval length = 0.05. Bottom: Derivative interval length = 0.2. Left: Computation interval length = 0.2. Right: Computation interval length = 0.1.

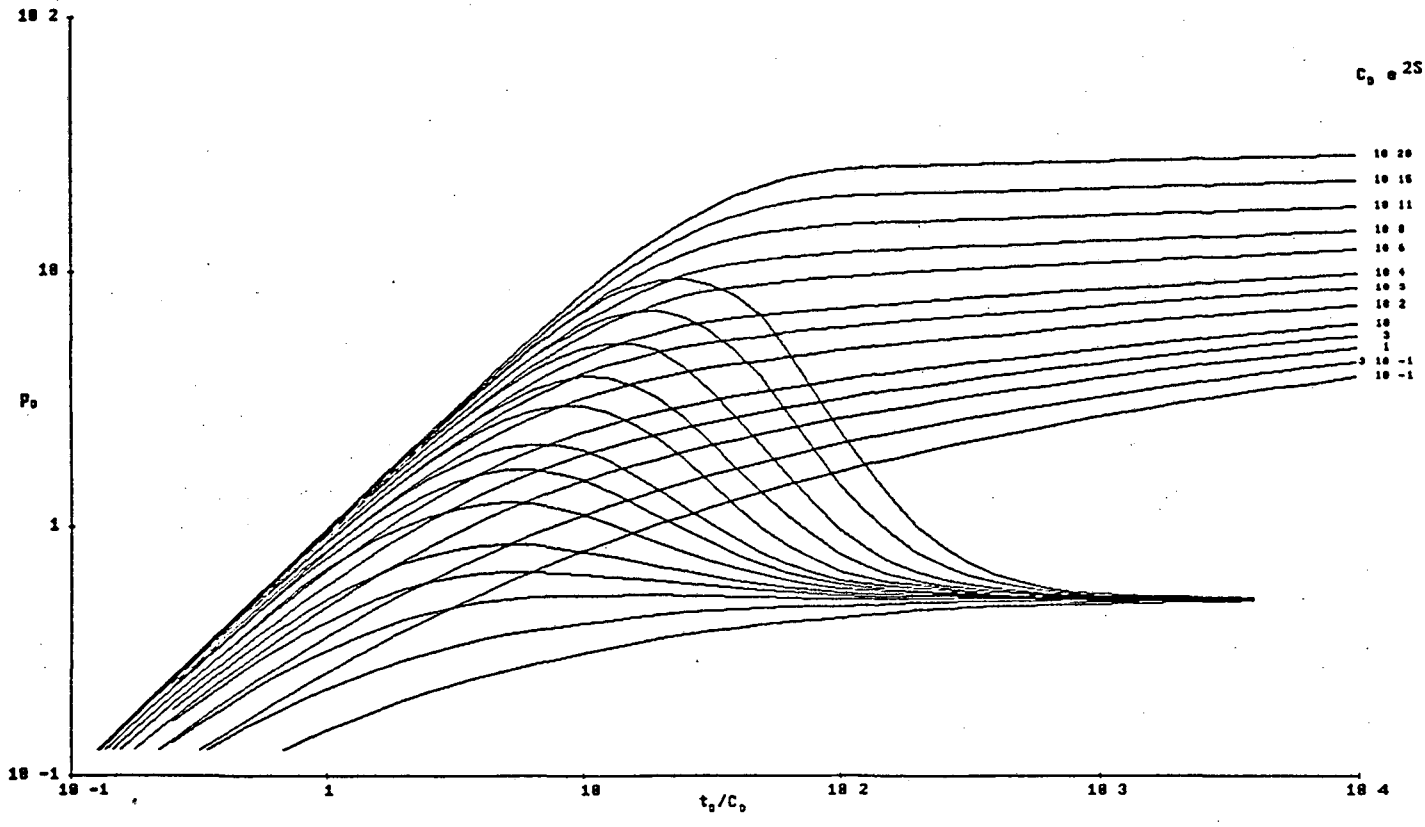


Figure A-1: Combined wellbore storage derivative and pressure type curves.

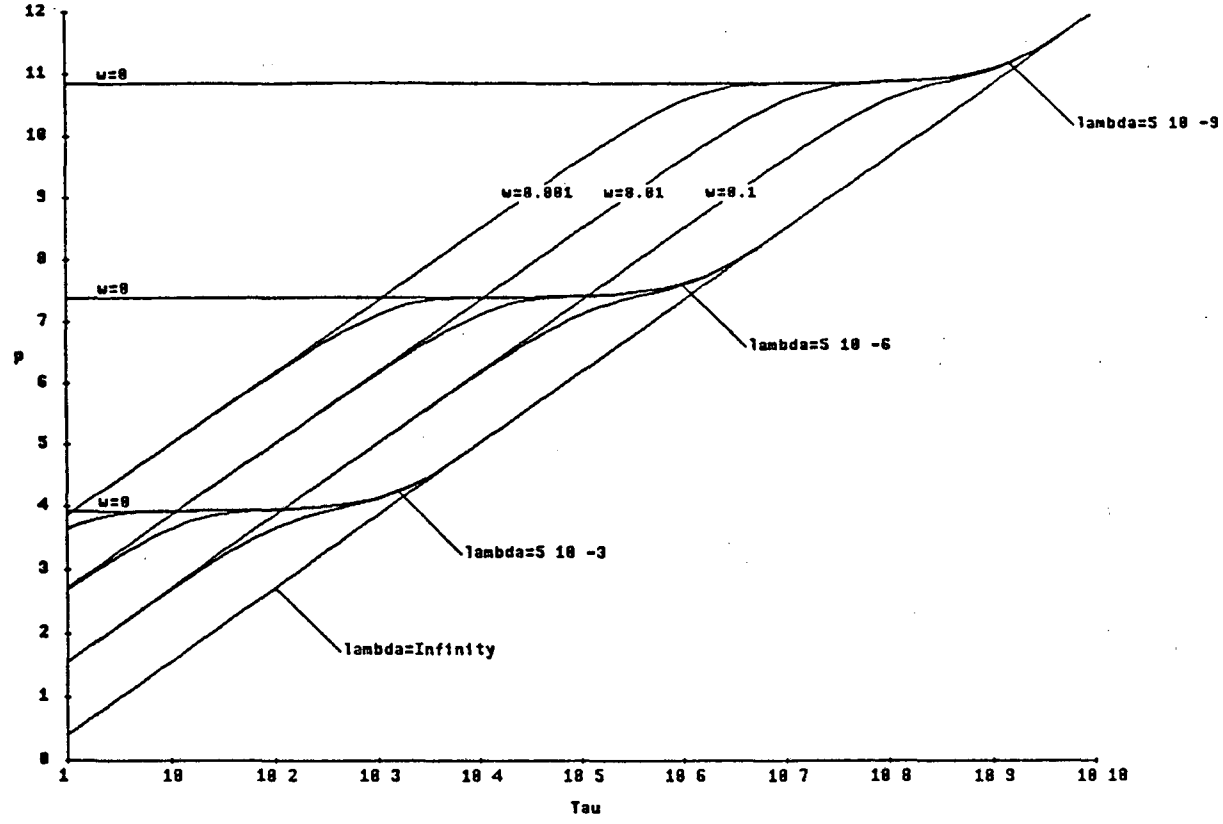


Figure A-2: Double porosity semilog type curves.

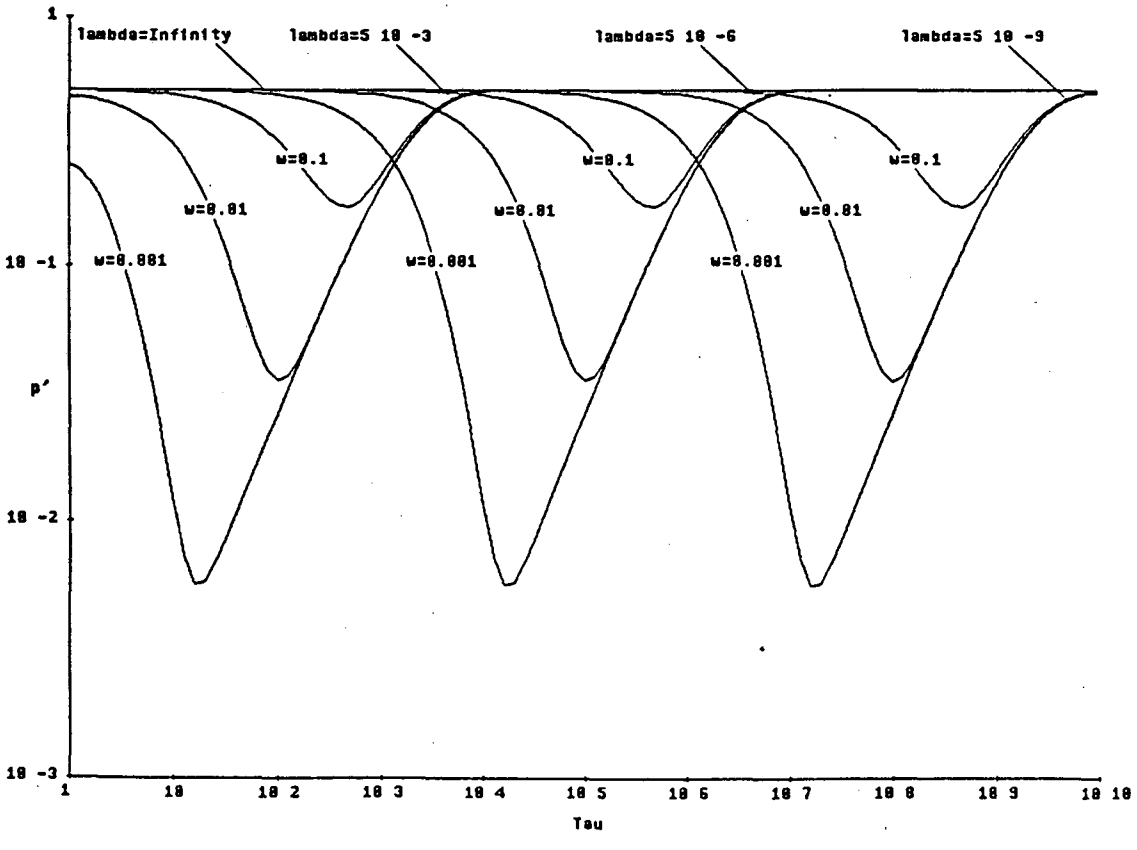


Figure A-3: Double porosity derivative type curves.

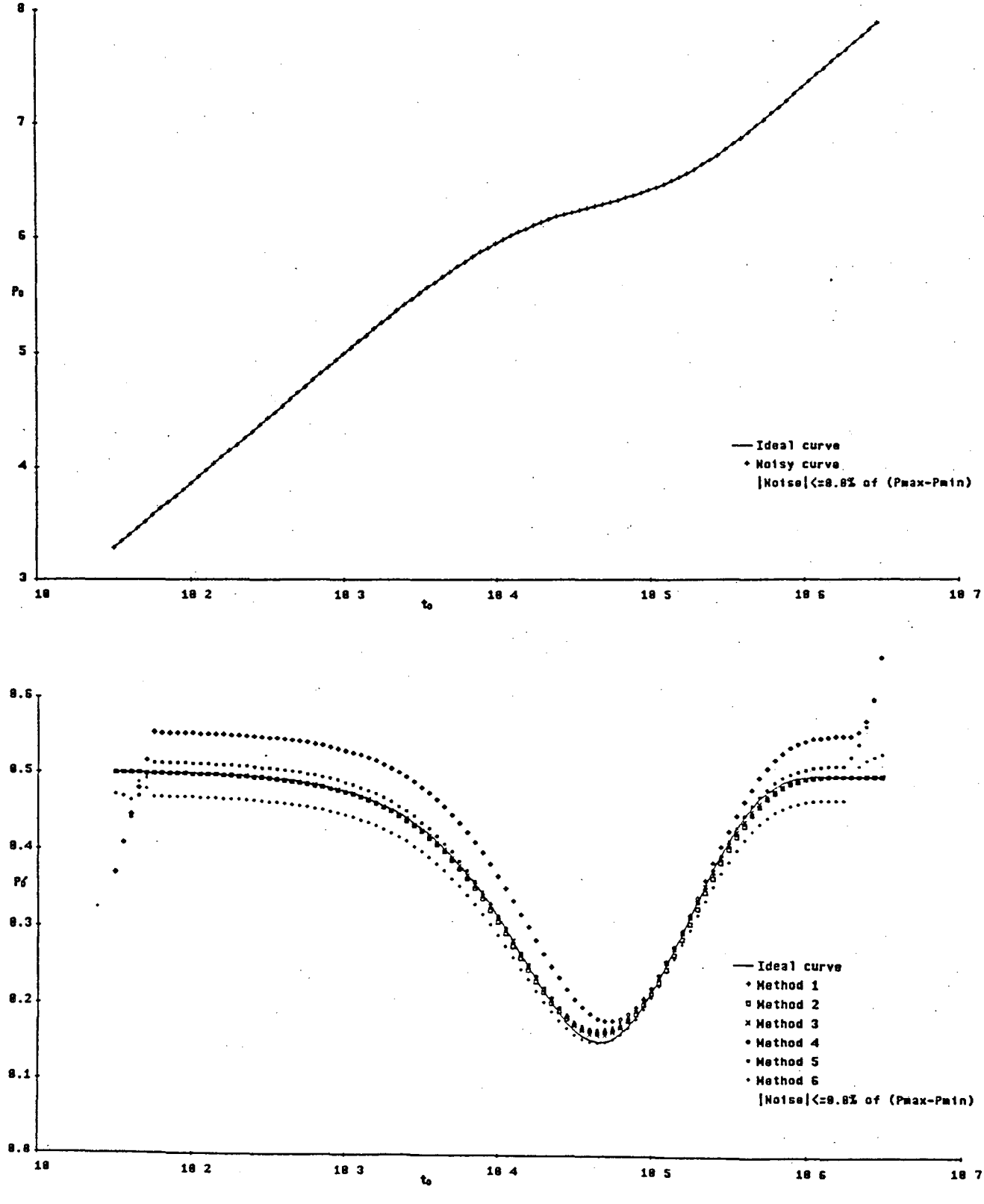


Figure A-4: Methods 1 to 6. Noise = 0%.

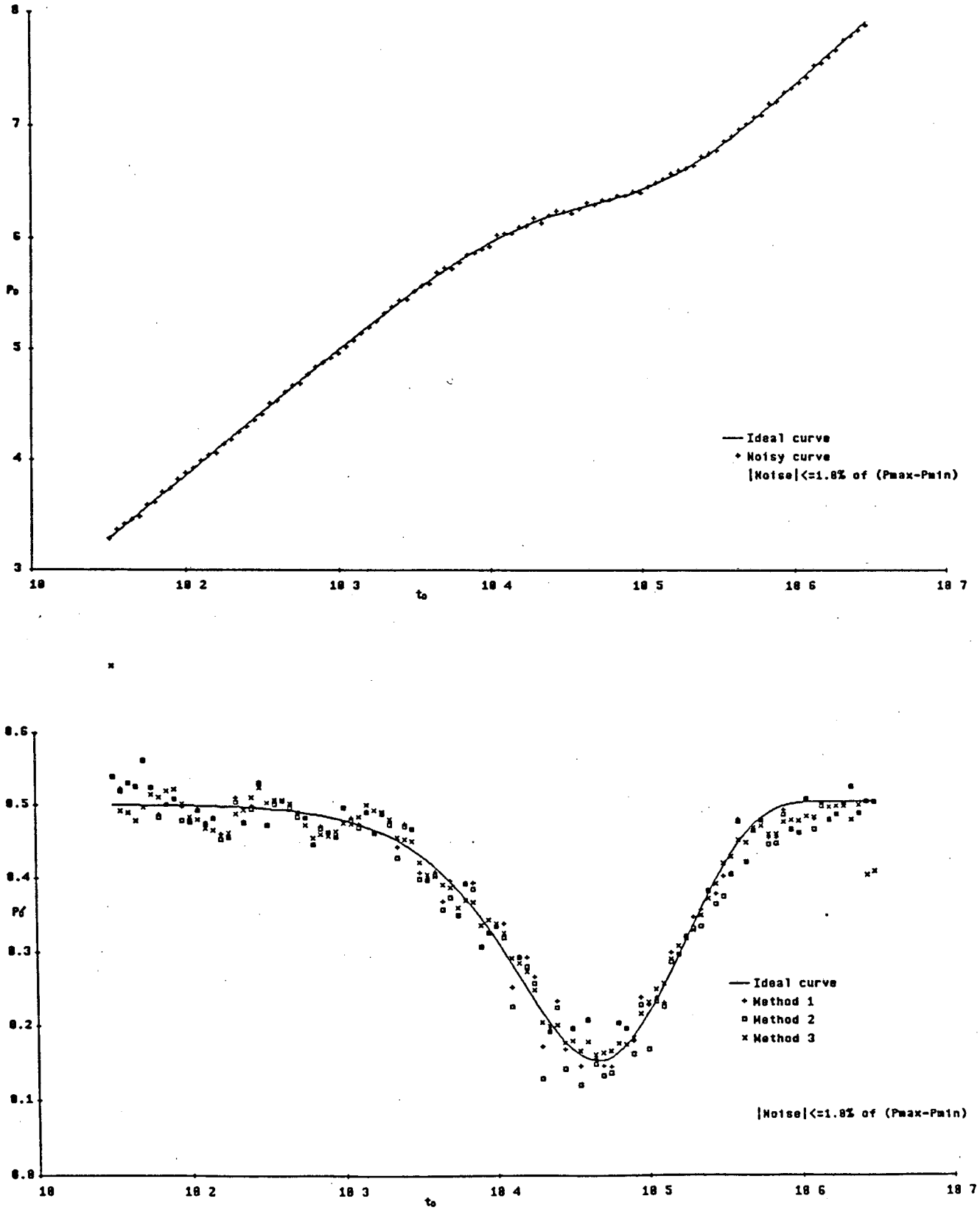


Figure A-5: Methods 1 to 3. Noise = 1%.

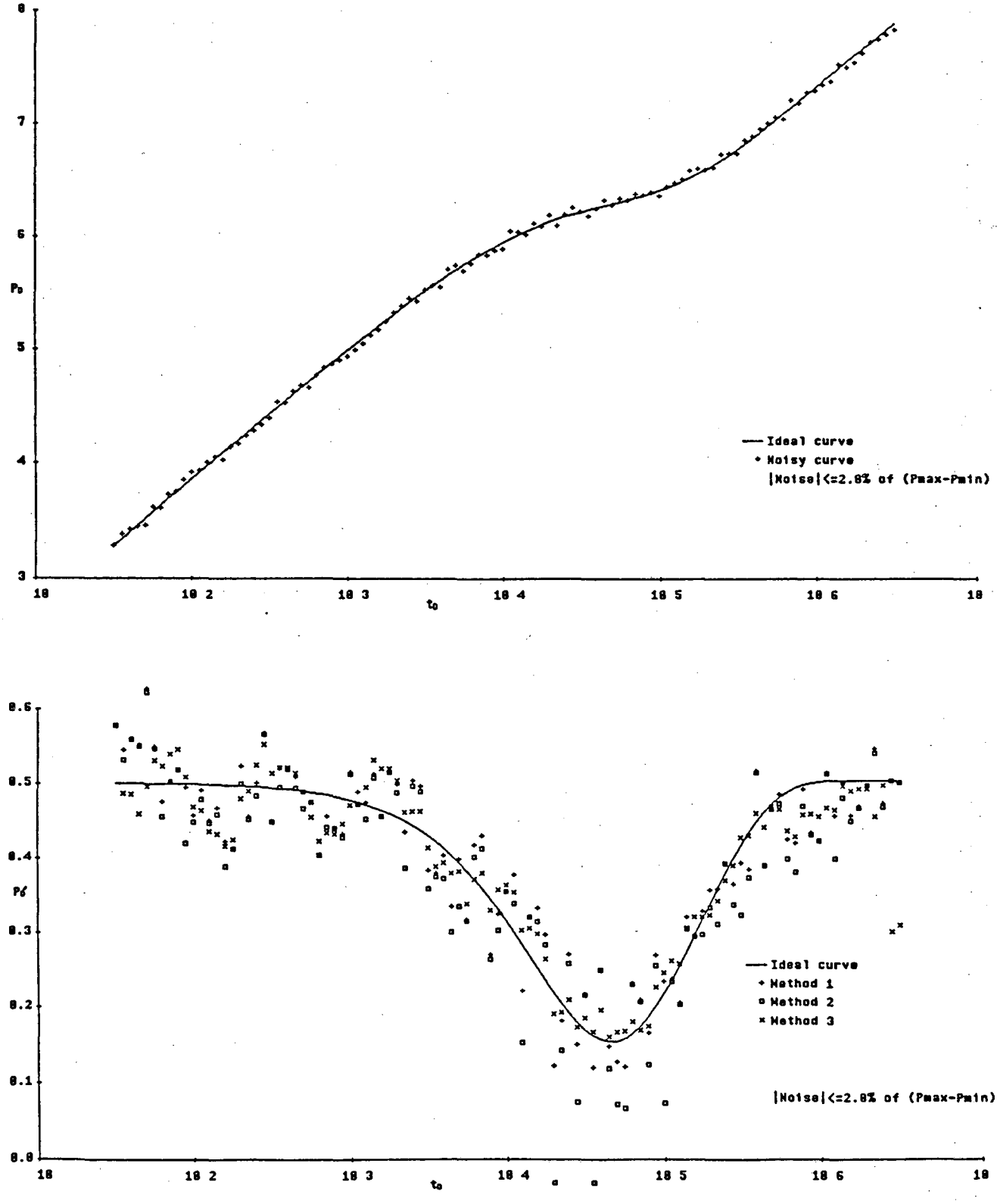


Figure A-6: Methods 1 to 3. Noise = 2%.

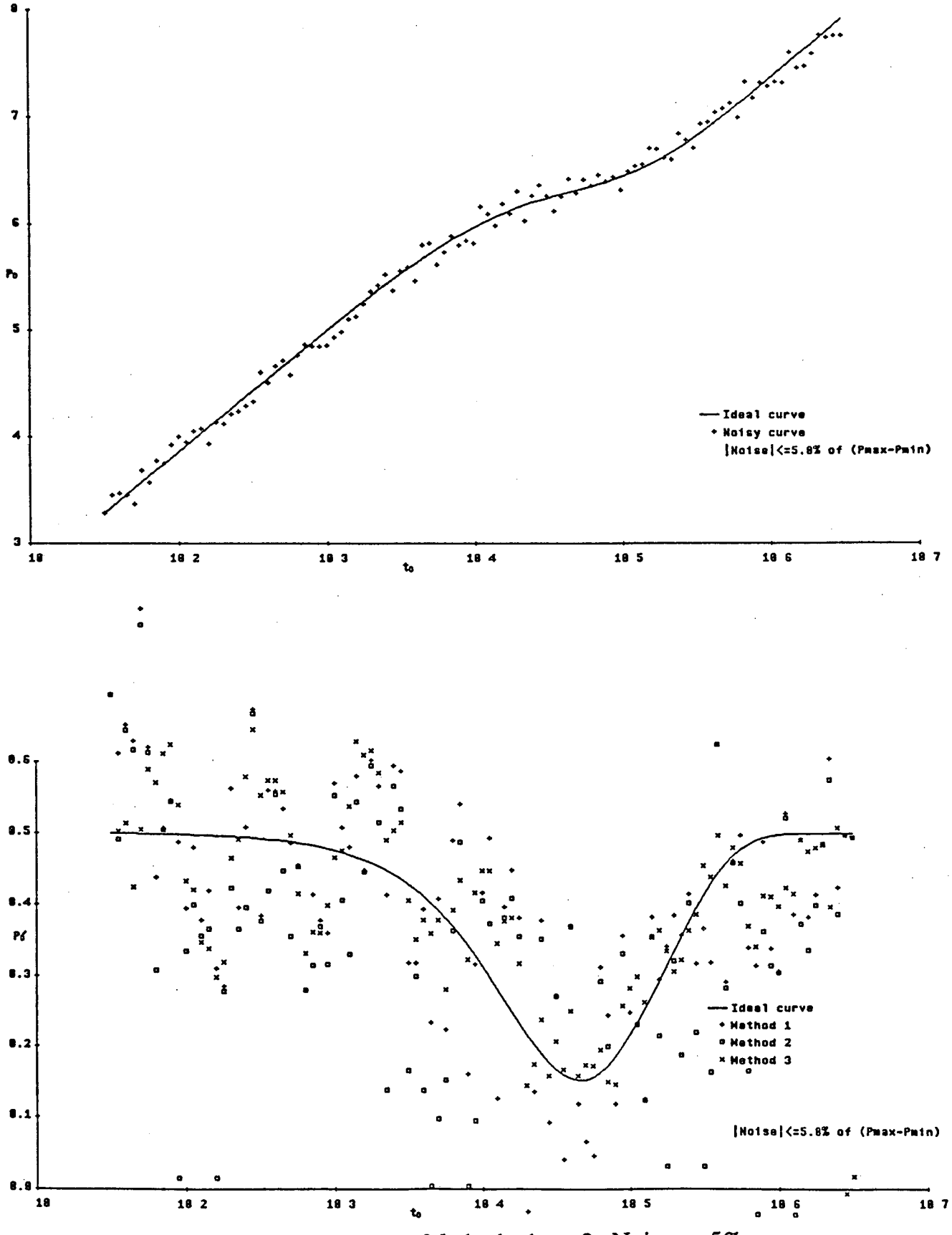


Figure A-7: Methods 1 to 3. Noise = 5%.

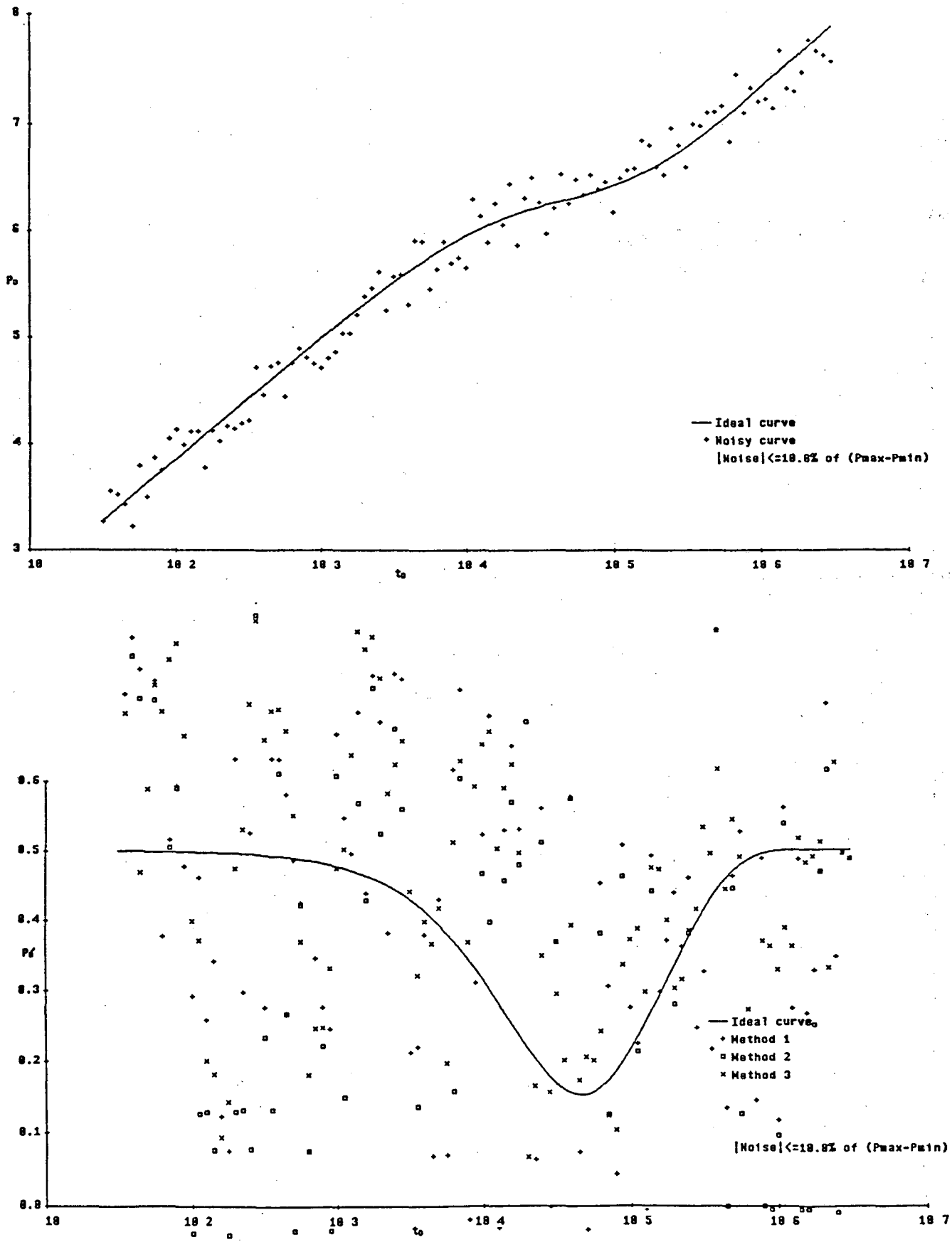


Figure A-8: Methods 1 to 3. Noise = 10%.

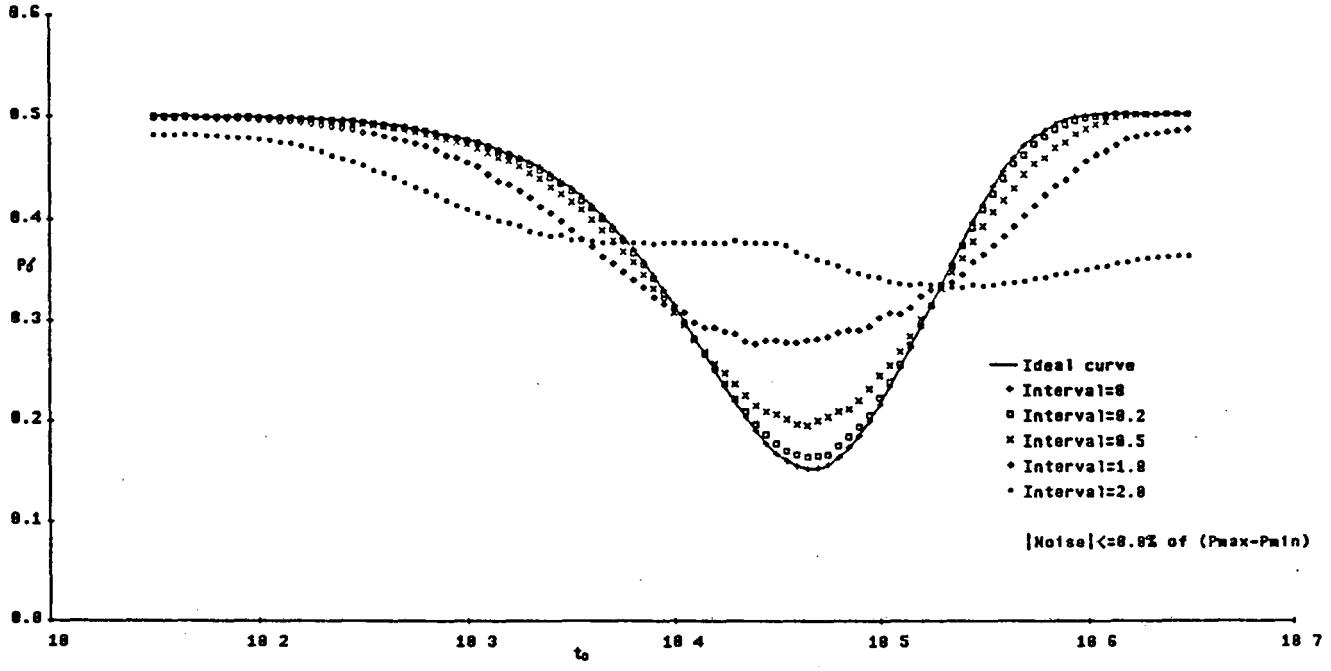


Figure A-9: Method 1. Noise = 0%.

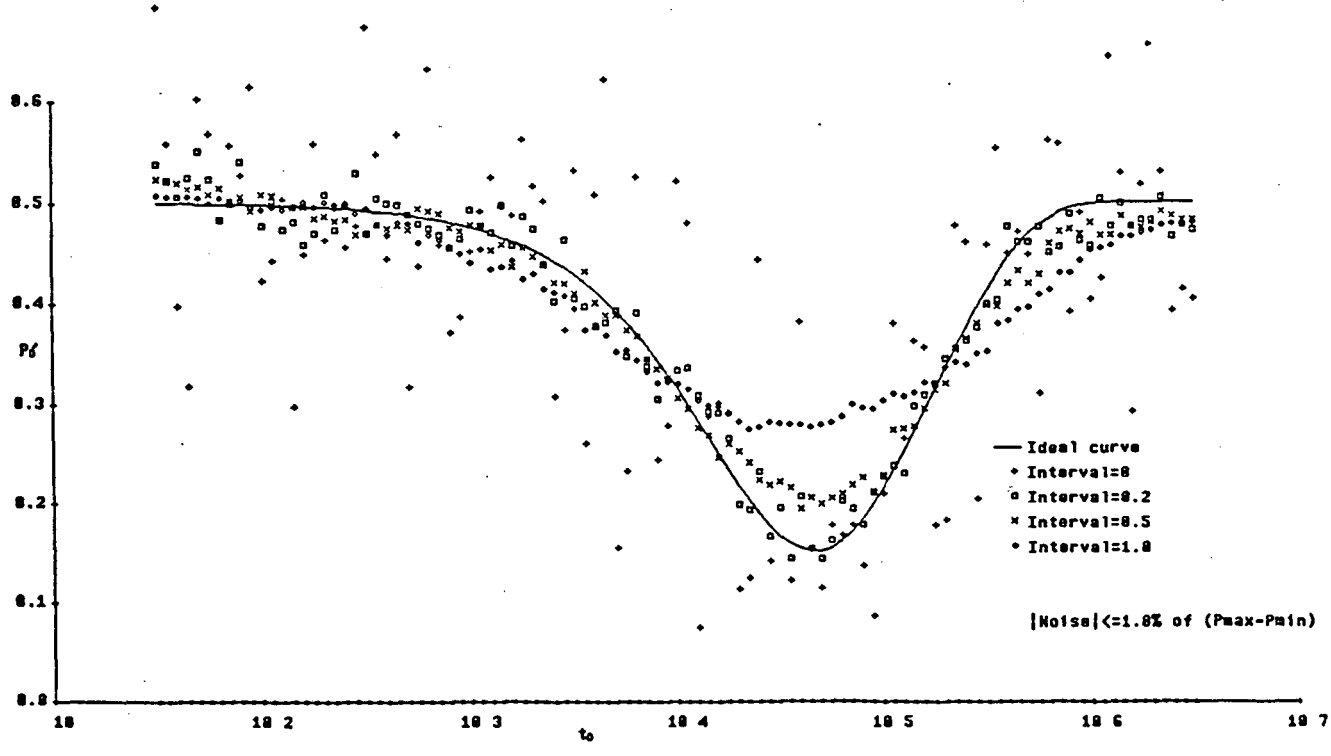


Figure A-10: Method 1. Noise = 1%.

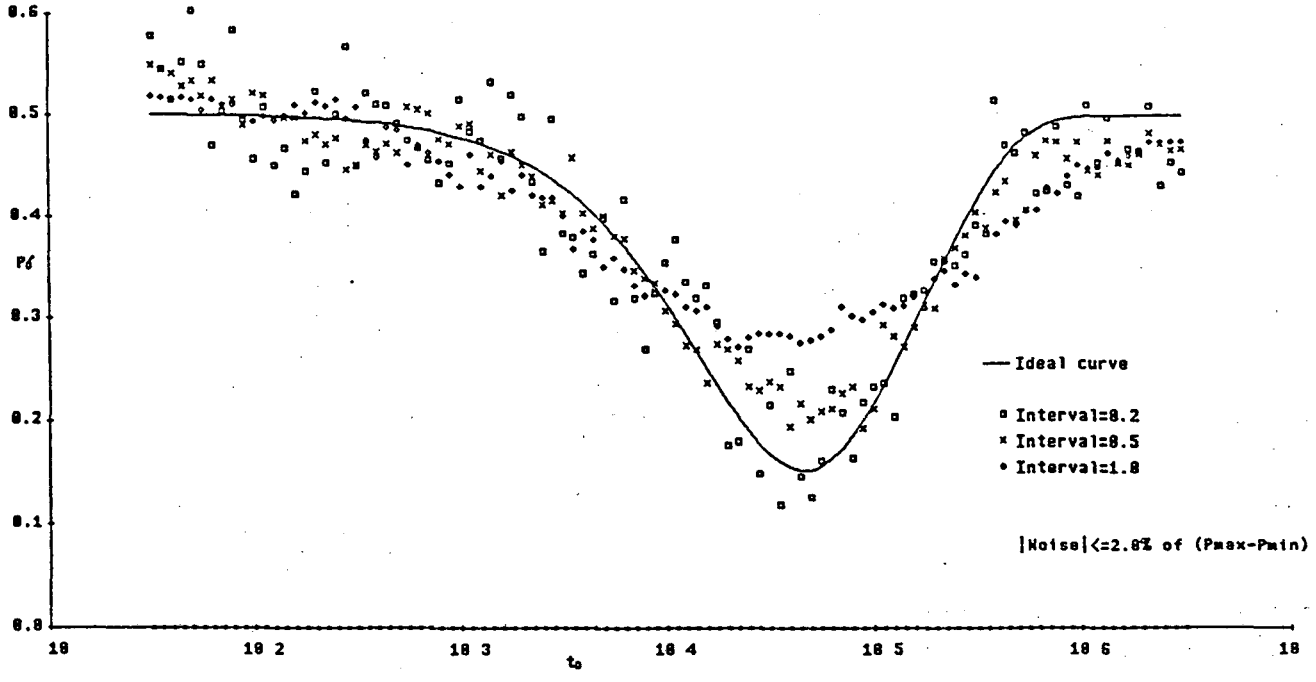


Figure A-11: Method 1. Noise = 2%.

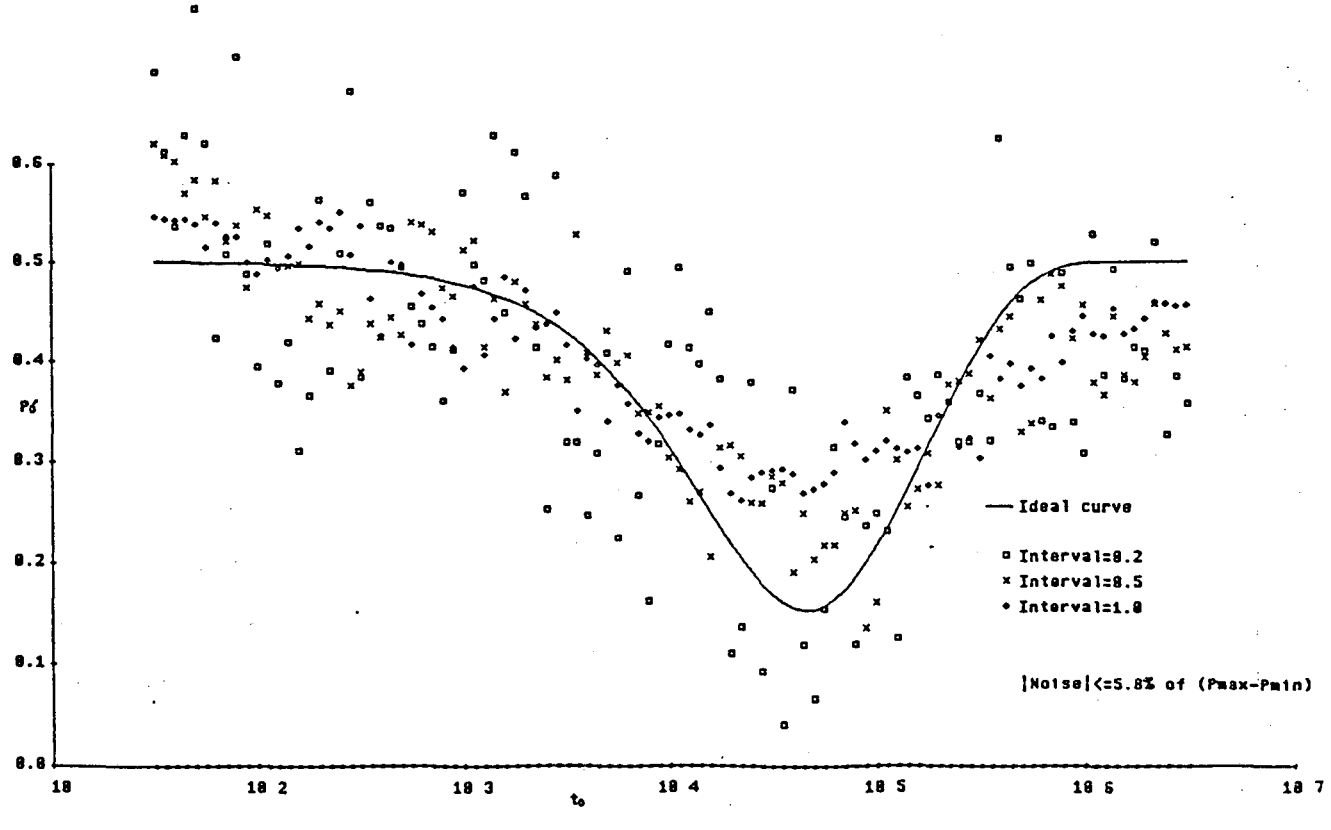


Figure A-12: Method 1. Noise = 5%.

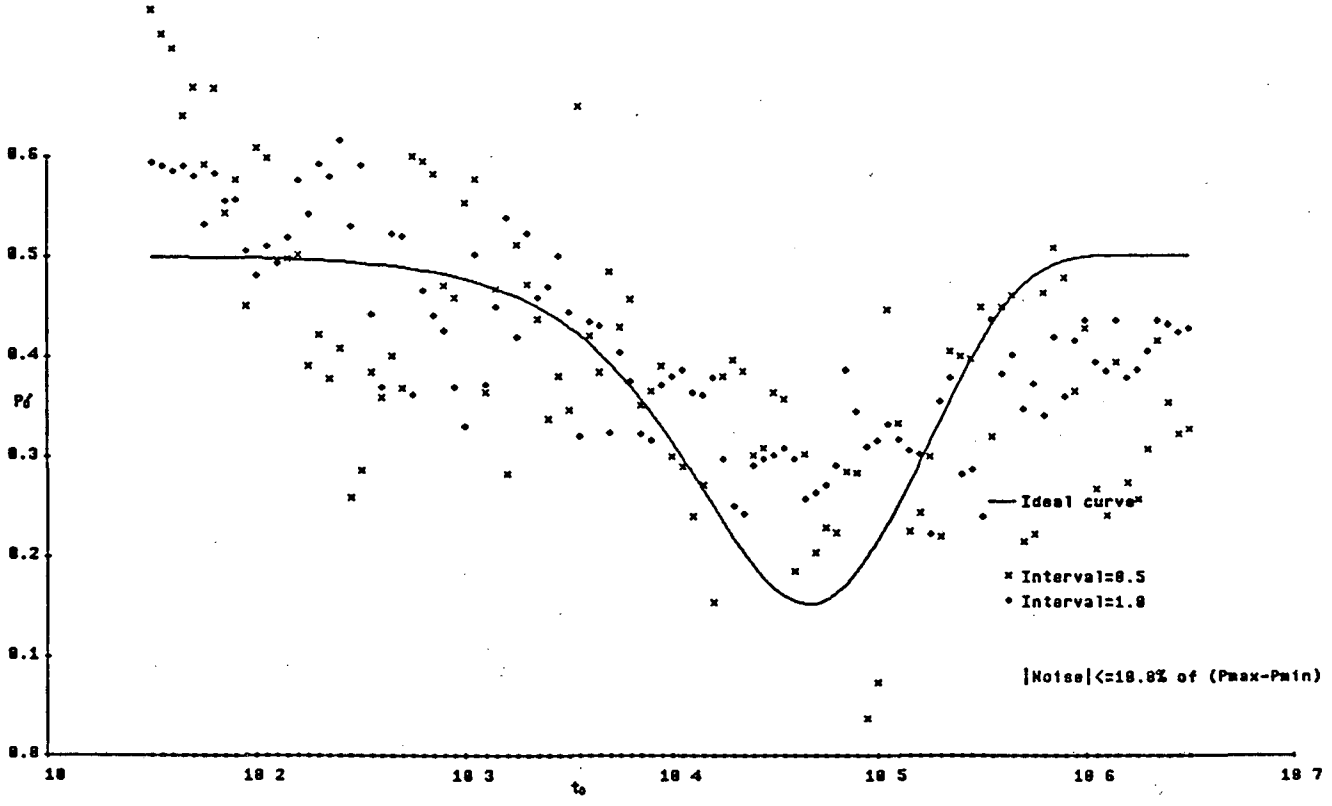


Figure A-13: Method 1. Noise = 10%.

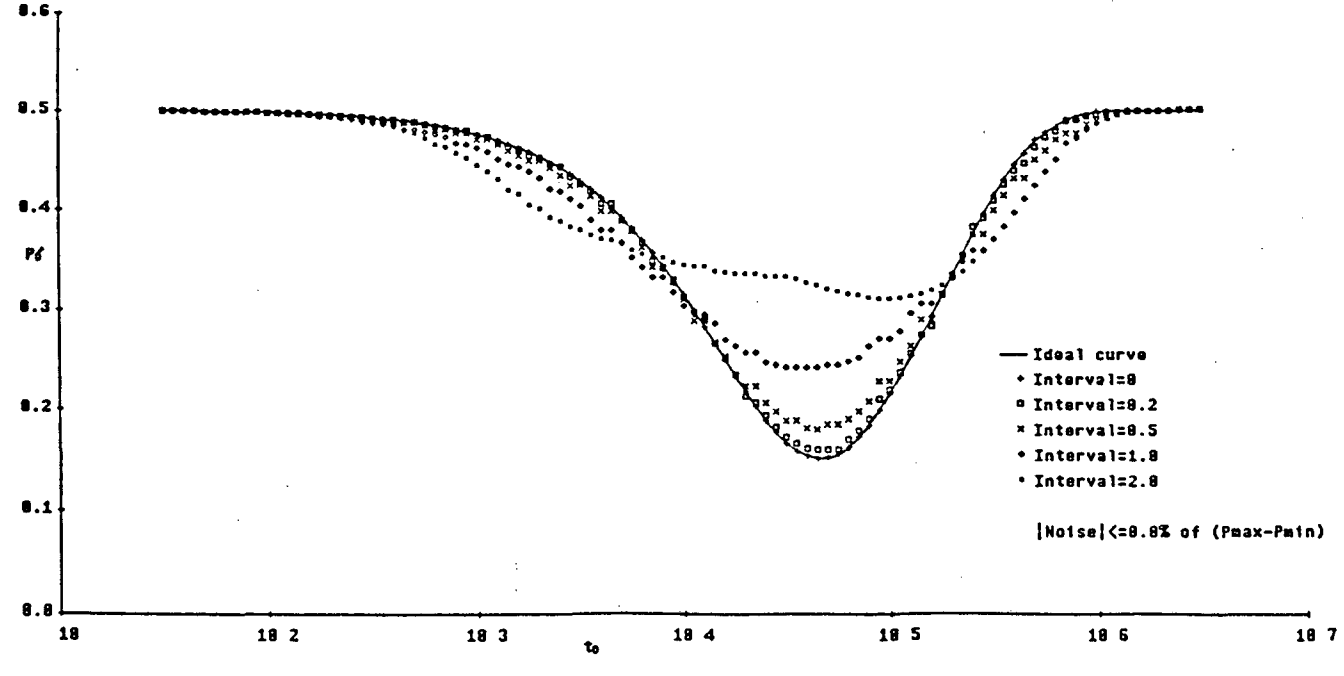


Figure A-14: Method 3. Noise = 0%.

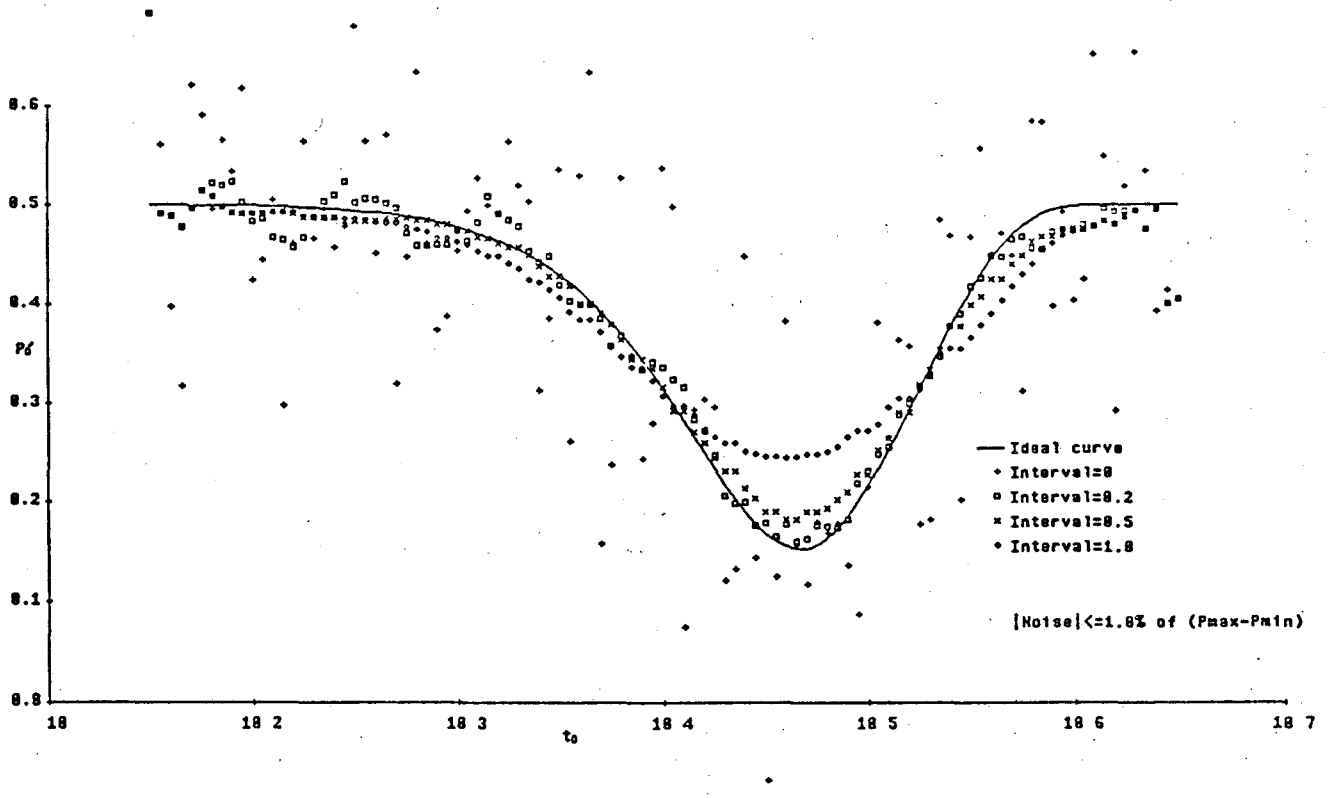


Figure A-15: Method 3. Noise = 1%.

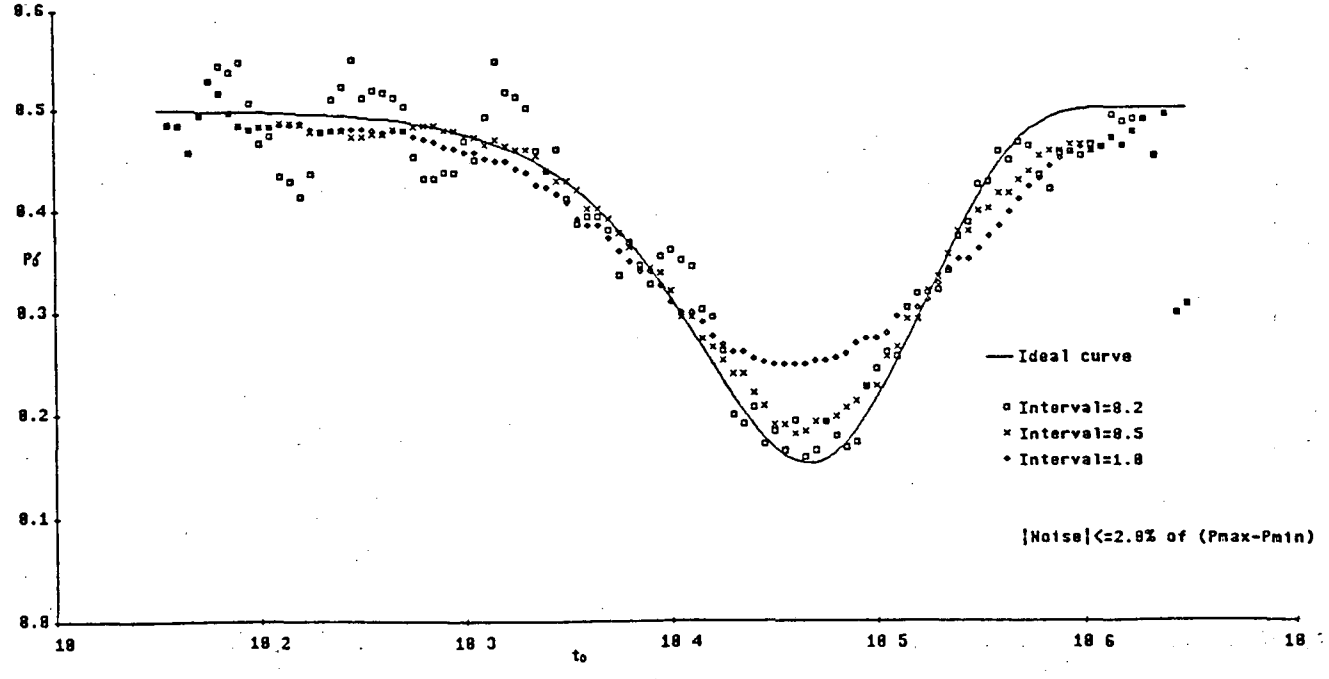


Figure A-16: Method 3. Noise = 2%.

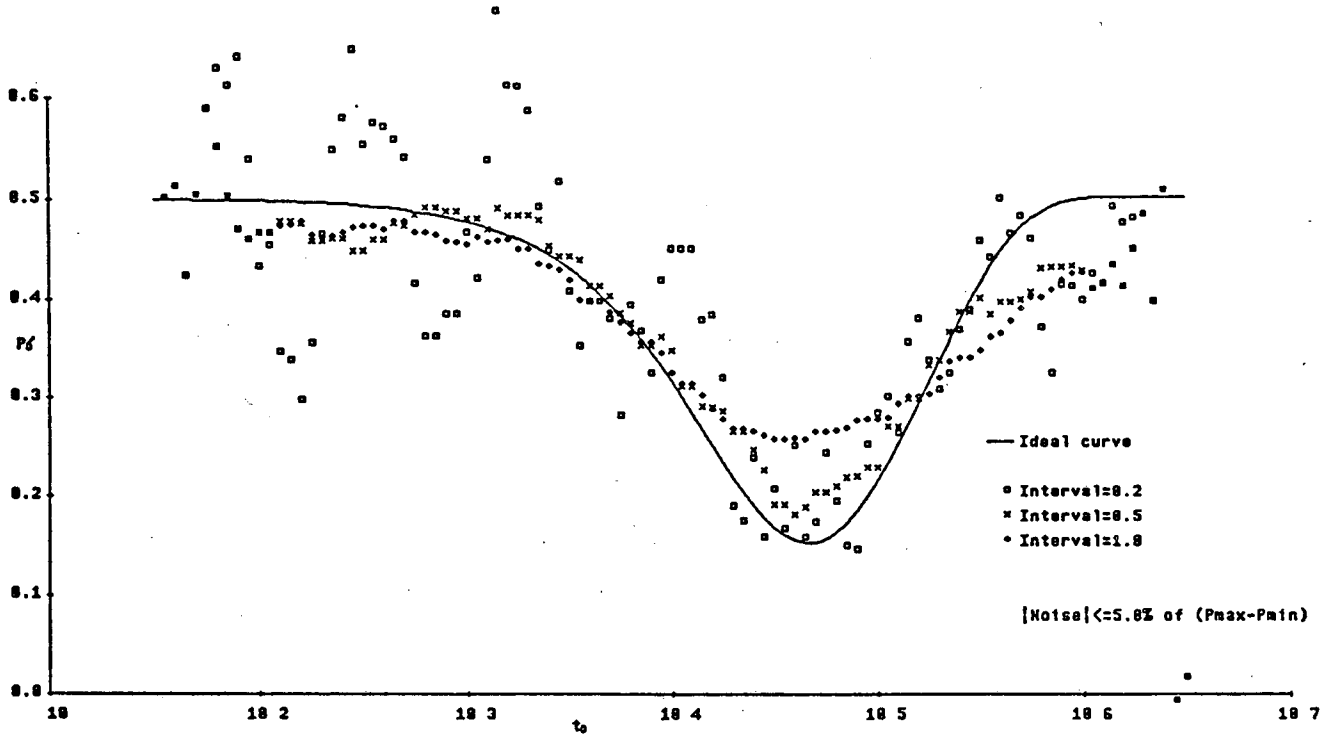


Figure A-17: Method 3. Noise = 5%.

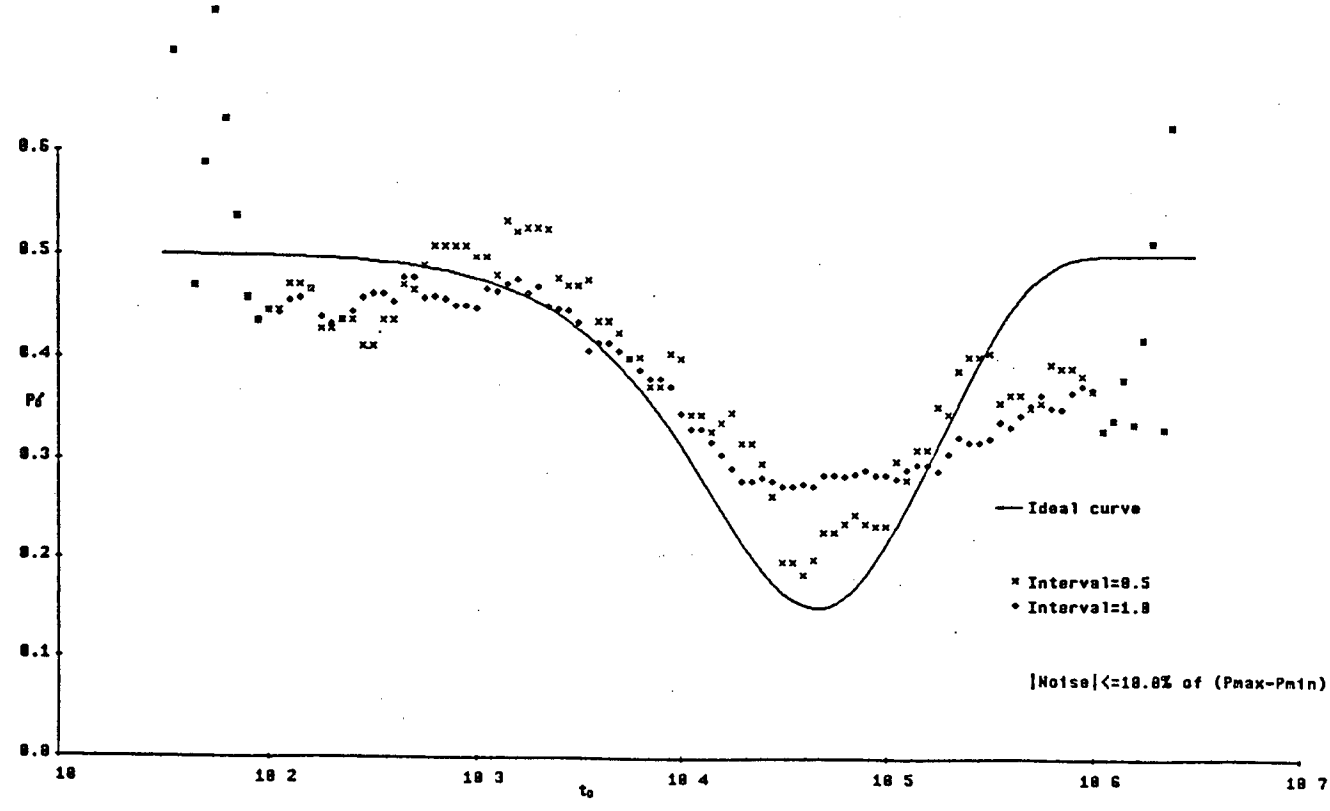


Figure A-18: Method 3. Noise = 10%.

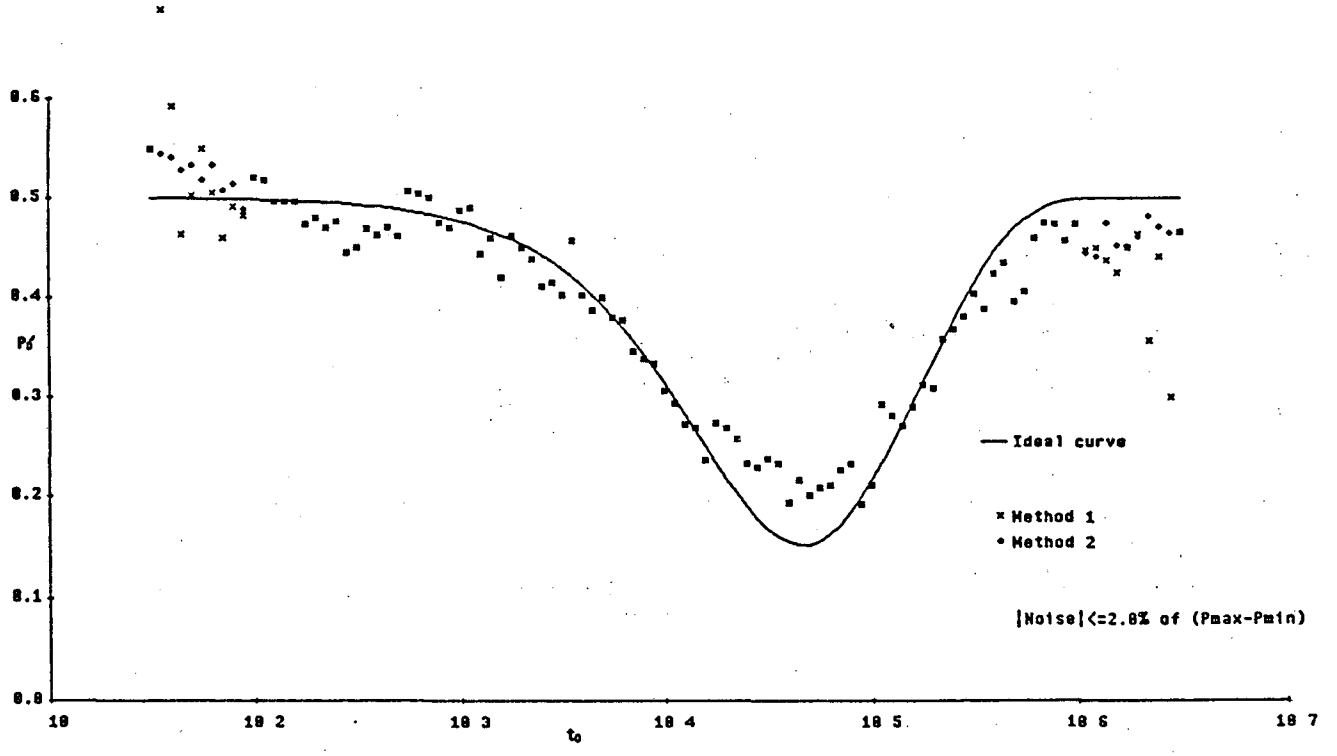


Figure A-19: Two methods to compute derivatives at curve extremities.

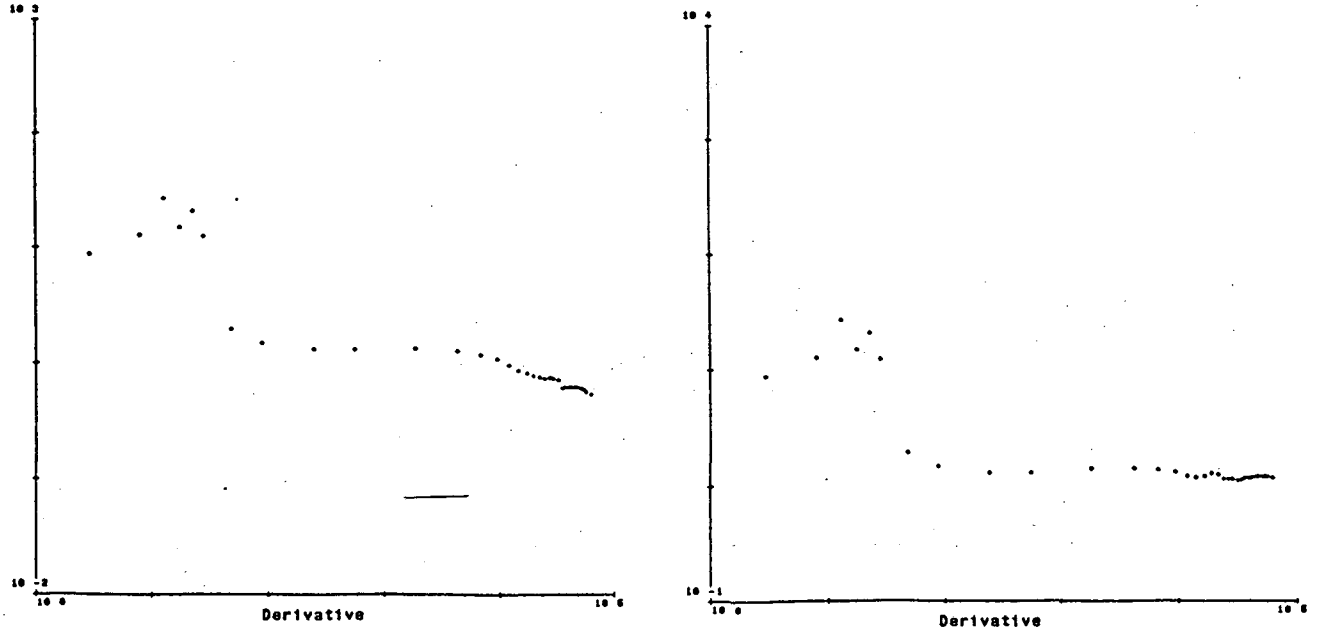


Figure A-20: Effect of the superposition function.

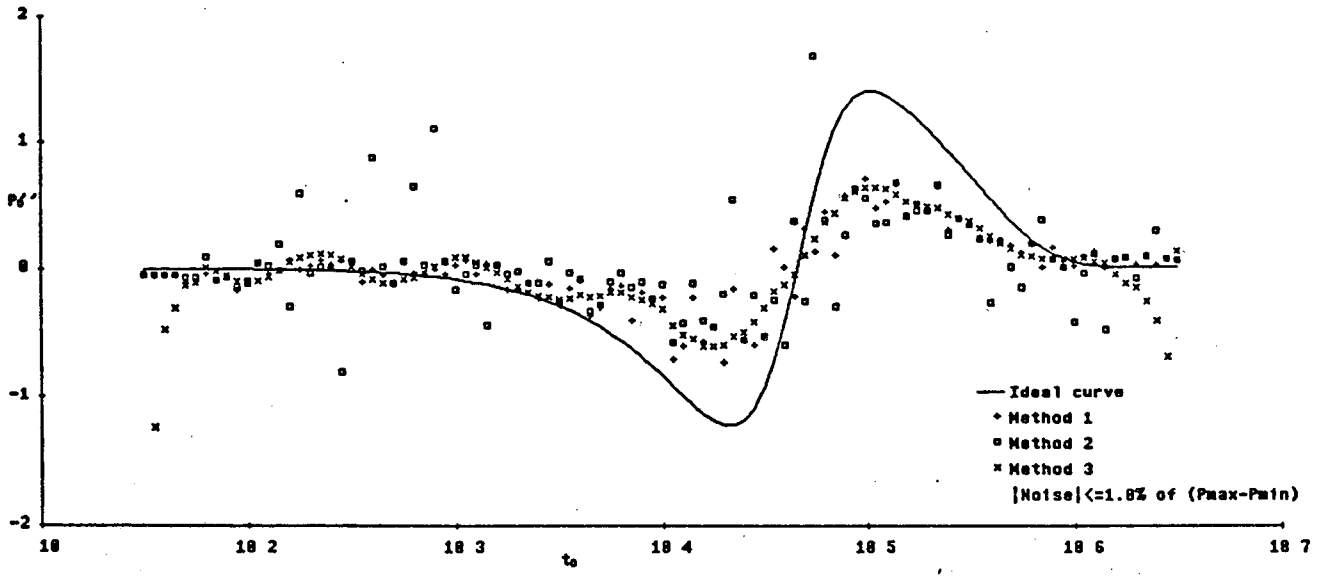


Figure A-21: Methods 1 to 3. Noise = 1%.

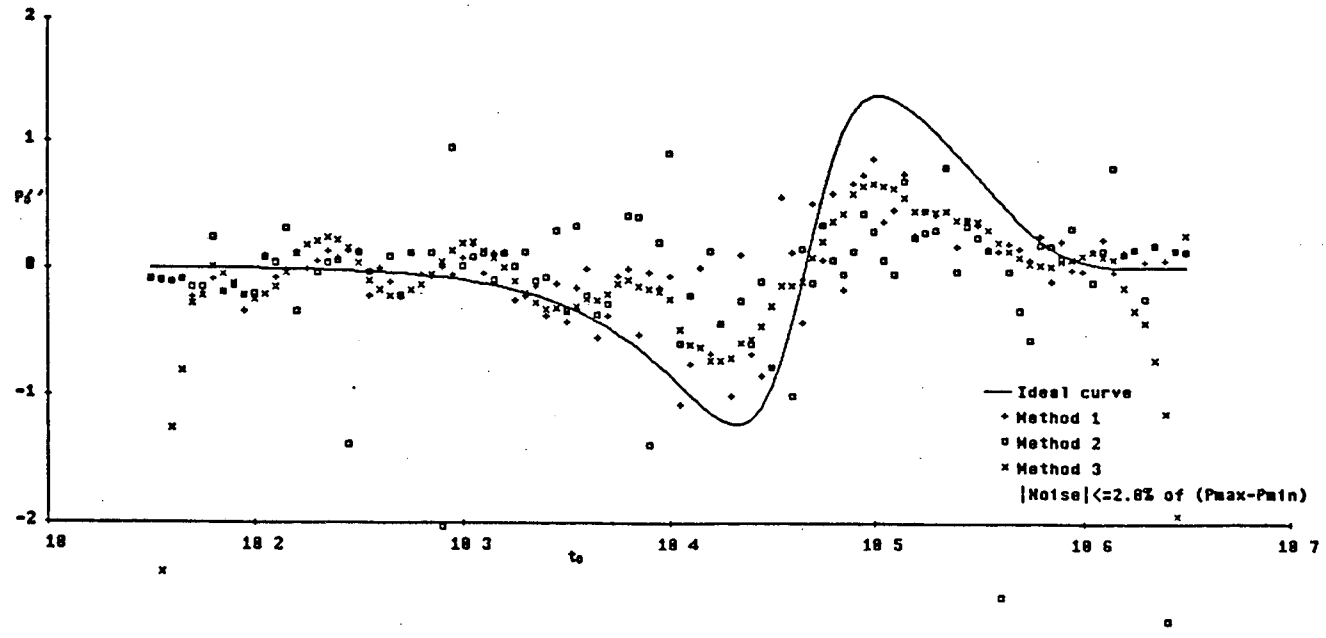


Figure A-22: Methods 1 to 3. Noise =2%.

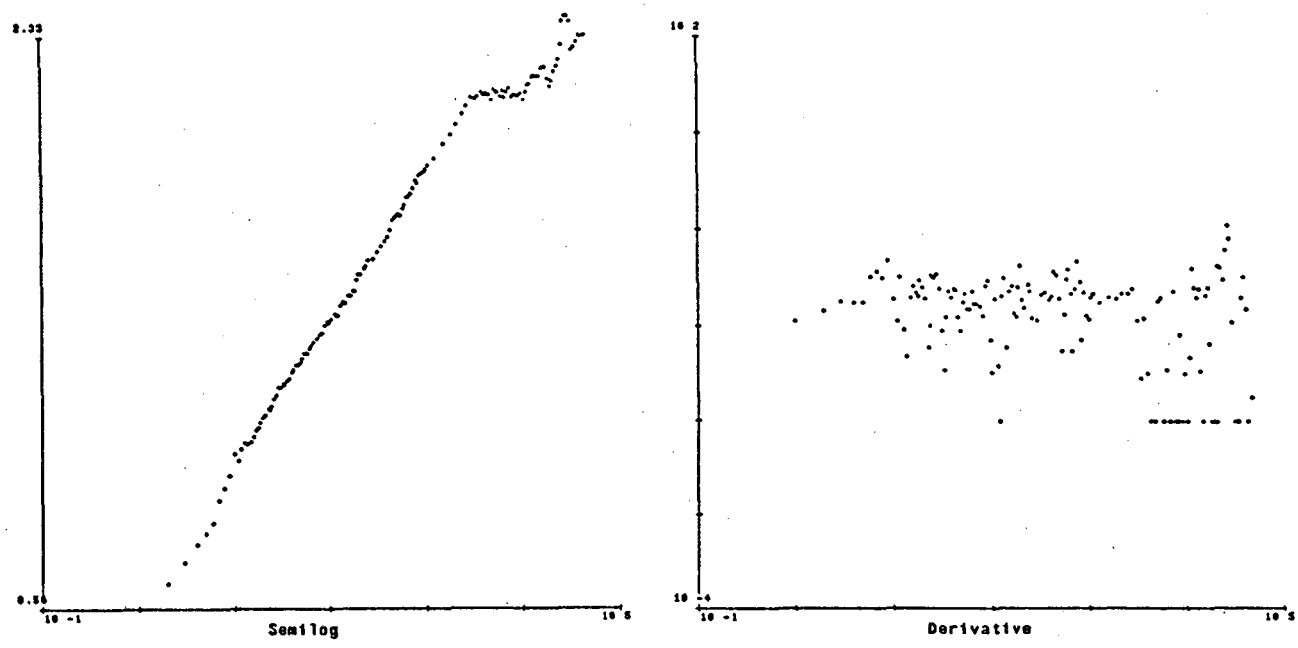


Figure A-23: Left: Semilog curve. Right: Derivative curve (I=0).

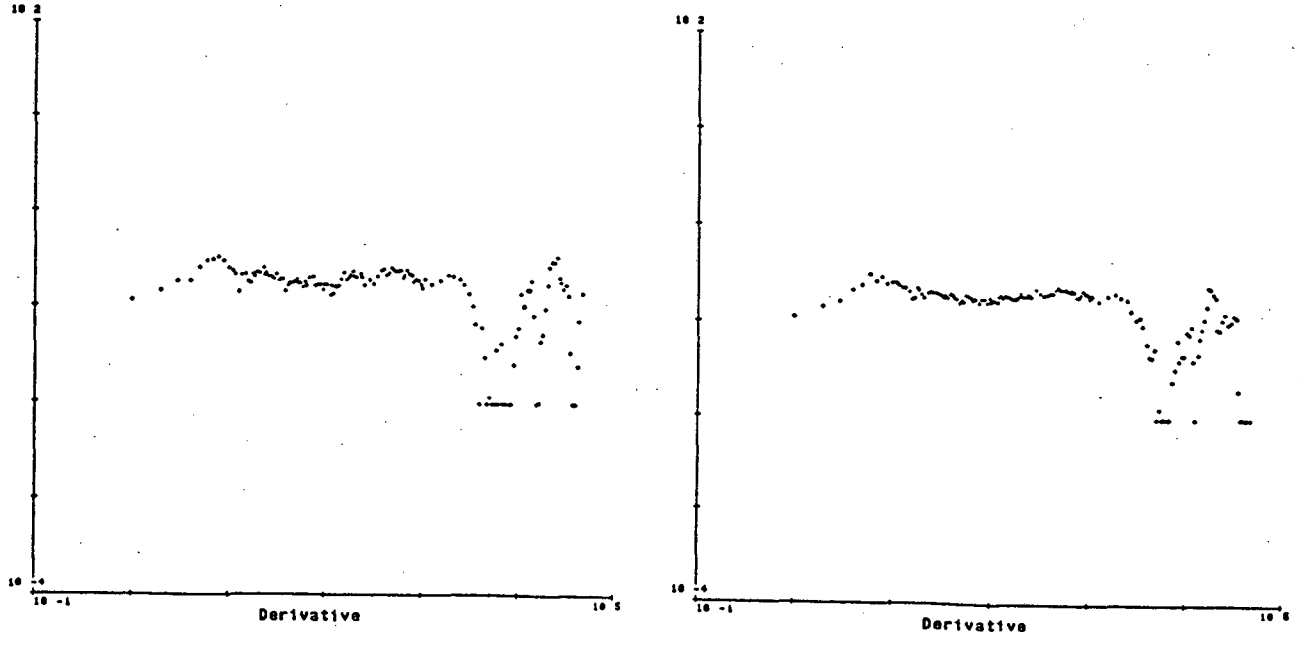


Figure A-24: Derivative curves. Left: I=0.1. Right: I=0.2.

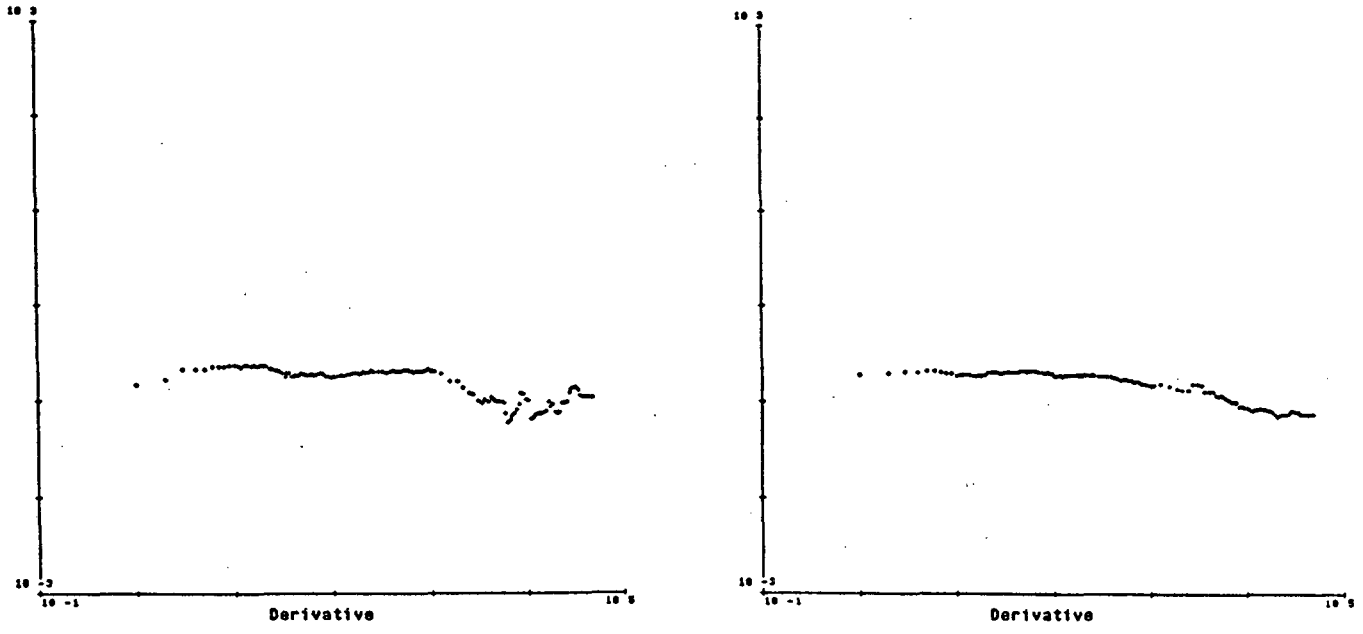


Figure A-25: Derivative curves. Left: I=0.5. Right: I=1.

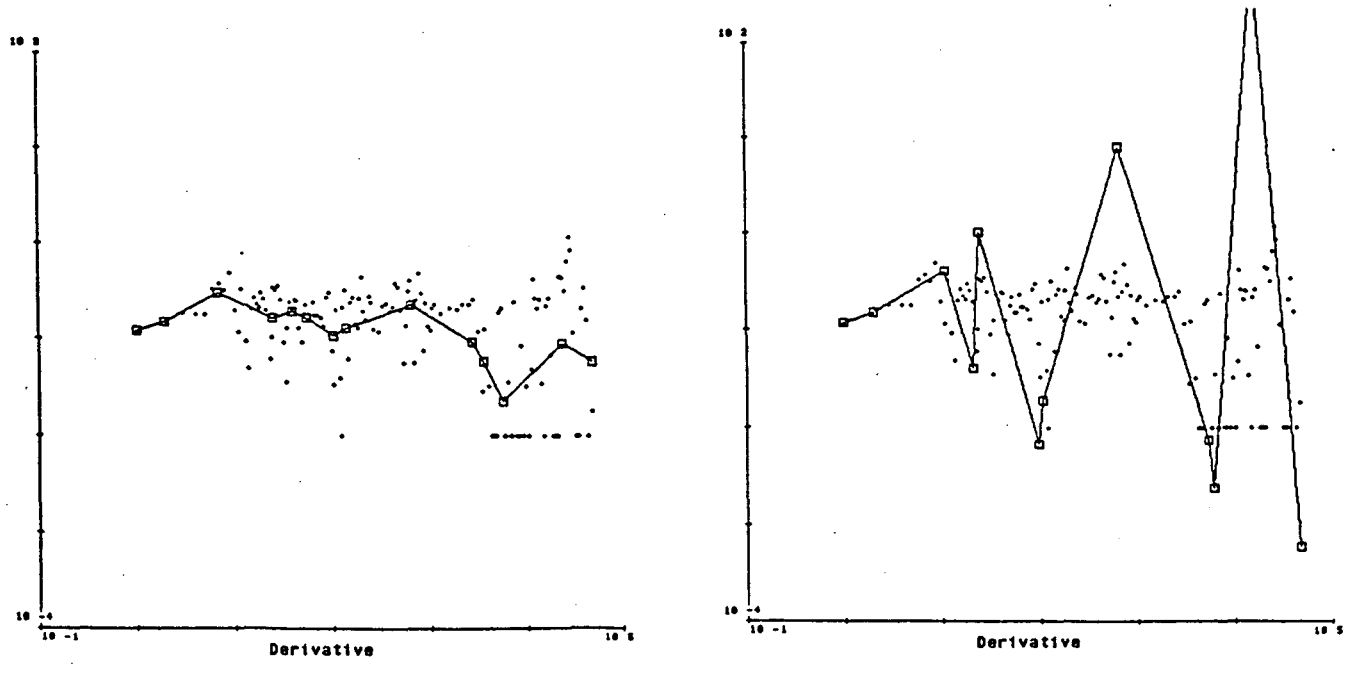


Figure A-26: Difference between real least-squares method (right) and linear regression (left).

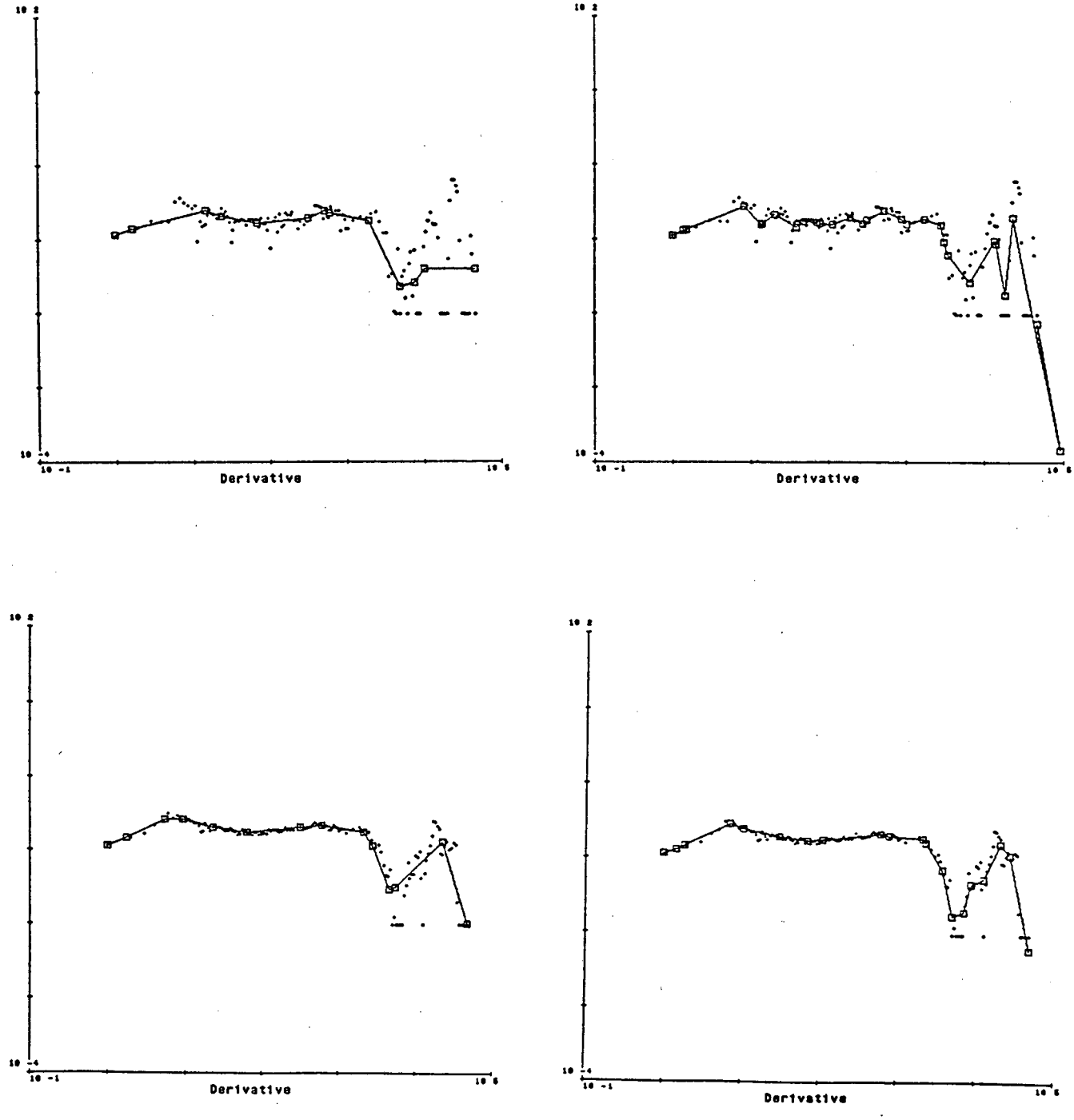


Figure A-27: Combined effects of derivative and least-squares smoothing. Top: Derivative interval length = 0.05. Bottom: Derivative interval length = 0.2. Left: Computation interval length = 0.2. Right: Computation interval length = 0.1.

APPENDIX B - DIFFERENCE BETWEEN LEAST-SQUARES AND LINEAR REGRESSION

LEAST-SQUARES

The function to be minimized with the least-squares method is

$$f(P_1, P_2, ..., P_n) = \sum_{i=1}^{n} \frac{(ax_i - y_i + b)^2}{1 + a^2}$$

where $P_1(x_1,y_1)$, $P_2(x_2,y_2)$, $P_n(x_n,y_n)$ are the points used to compute the straight line, a the slope and b the intercept. f is based on the euclidian distance. The values of a and b are given by

$$a = \frac{-B + \sqrt{B^2 + 4A^2}}{2A} \quad \text{and} \quad b = \bar{y} - a\bar{x}$$

$$\text{where } A = \sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y} \quad \text{and} \quad B = \sum_{i=1}^{n} (x_i^2 - y_i^2) - n(\bar{x}^2 - \bar{y}^2)$$

LINEAR REGRESSION

In this case, the function is given by

$$f(P_1, P_2, ..., P_n) = \sum_{i=1}^{n} (ax_i - y_i + b)^2$$

with the same notations. f is based here on the vertical distance between each point P_i and the straight line. It means that the value of f increases faster with this method than with the least-squares one when a increases. The values of a and b are given by

$$a = \frac{A}{B} \quad \text{and} \quad b = \bar{y} - a\bar{x}$$

$$\text{where } A = \sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y} \quad \text{and} \quad B = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2$$

APPENDIX C - LISTINGS

C PROGRAM

The listings of the following files are given:

- "cmp.h": Declarations for computation functions.
- "cmp_data.c": Read the data, filter it and compute derivatives and curves.
- "cmp_main.c": Call the different functions used for well numerical analysis.
- "cmp_pattern.c": Compute patterns from the curves.
- "cmp_send.c": Interface between C and ART.

- "env.h": Set the environment variables.

- "gph.h": Declarations for graphic functions.
- "gph_main.c": Drawing functions for the curves.
- "gph_util.c": Low-level graphic functions.

- "macro.h": macro function definitions.

- "main.c": function main().

- "pat.h": Declarations for patterns.

- "win.h": Declarations for windows.
- "win_bar.c": Create the explanation line at the bottom of the screen.
- "win_base.c": Create the main window (list of wells).
- "win_closer.c": Ask the user for closing windows when too many of them are opened.
- "win_confirmer.c": Confirm quit.
- "win_events.c": Handle mouse events (clicks, moves, etc...).
- "win_init.c": Initializations for windows.
- "win_main.c": Create all the windows.
- "win_util.c": Utilities for windows.
- "win_wells.c": Create the windows used for curve drawing.

ART PROGRAM

Two files are used in the current version

- "init.art": Set the object architecture, functions used to interface C and ART.
- "model.art": Rules used to extract the models from the patterns.

```
/*****************************************************************************
/* "cmp.h" : global constants and variables for computations.                *
/*****************************************************************************

#ifdef CMP_MAIN
#define EXTERN
#else
#define EXTERN extern
#endif


/*****************************************************************************
/* wells                                                                     *
/*****************************************************************************

#define MAX_NBR_WELLS 20
#define MAX_NBR_INITIAL_PTS 1000
#define MAX_NBR_PTS 61
#define INTERV 3
#define EPSILON 1e-6

EXTERN float initial_time[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];
EXTERN float initial_pressure[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];

EXTERN float time[MAX_NBR_WELLS][MAX_NBR_PTS];
EXTERN float pressure[MAX_NBR_WELLS][MAX_NBR_PTS];

EXTERN float p_derivative[MAX_NBR_WELLS][MAX_NBR_PTS];
EXTERN float p_d_derivative[MAX_NBR_WELLS][MAX_NBR_PTS];
EXTERN float log_derivative[MAX_NBR_WELLS][MAX_NBR_PTS];


EXTERN float first_pressure[MAX_NBR_WELLS];
EXTERN float i_last_time[MAX_NBR_WELLS];
EXTERN float i_last_pressure[MAX_NBR_WELLS];
EXTERN float last_time[MAX_NBR_WELLS];
EXTERN float last_pressure[MAX_NBR_WELLS];
EXTERN float min_p_derivative[MAX_NBR_WELLS];
EXTERN float max_p_derivative[MAX_NBR_WELLS];
EXTERN float min_p_d_derivative[MAX_NBR_WELLS];
EXTERN float max_p_d_derivative[MAX_NBR_WELLS];
EXTERN float local_max_time[MAX_NBR_WELLS][10];
EXTERN int i_min_p_derivative[MAX_NBR_WELLS];
EXTERN int i_max_p_derivative[MAX_NBR_WELLS];
EXTERN int i_min_p_d_derivative[MAX_NBR_WELLS];
EXTERN int i_max_p_d_derivative[MAX_NBR_WELLS];



/*****************************************************************************
/* curves                                                                    *
/*****************************************************************************

#define MAX_WIDTH 450
#define MAX_HEIGHT 450
#define X_ORIGIN 25
```

```
#define Y_ORIGIN 475


EXTERN short initial_curve_x[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];
EXTERN short initial_curve_y[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];

EXTERN short curve_x[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];
EXTERN short curve_y[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];
EXTERN short curve_y1[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];

EXTERN short curve_log_x[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];

EXTERN short curve_log_y[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];

EXTERN short derivative_log_y[MAX_NBR_WELLS][MAX_NBR_INITIAL_PTS];

EXTERN short log_x_min[MAX_NBR_WELLS];
EXTERN short log_y_min[MAX_NBR_WELLS];
EXTERN short log_cycles[MAX_NBR_WELLS];


EXTERN struct axis_mark {int pos; char text[10]};

EXTERN struct axis_mark initial_x_axis[MAX_NBR_WELLS][10];
EXTERN struct axis_mark initial_y_axis[MAX_NBR_WELLS][10];

EXTERN struct axis_mark x_axis[MAX_NBR_WELLS][10];
EXTERN struct axis_mark y_axis[MAX_NBR_WELLS][10];
EXTERN struct axis_mark y1_axis[MAX_NBR_WELLS][10];

EXTERN struct axis_mark log_x_axis[MAX_NBR_WELLS][10];
EXTERN struct axis_mark log_y_axis[MAX_NBR_WELLS][10];
```

```
/*****************************************************************************
/* "cmp_data.c" : compute filtered data, derivatives and curves.             *
/*****************************************************************************
#include <math.h>
#include <stdio.h>
#include "cmp.h"
#include "env.h"
#include "macro.h"


/*****************************************************************************
/* references                                                                *
/*****************************************************************************


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern int read_well();
  static float hms_to_s();
extern void filter_data();
  static float *derivative_func();
extern void compute_p_derivative();
extern void examine_p_derivative();
extern void compute_p_d_derivative();
extern void compute_extrema();
extern void compute_log_derivative();

  static int c_c();
extern void compute_initial_curve();
extern void compute_filtered_curve();
extern void compute_log_scale();
extern void compute_semilog_curve();
extern void compute_loglog_curve();
extern void compute_derivative_curve();

/*****************************************************************************
/* read initial data for well n                                              *
/*****************************************************************************
extern int read_well(n,name)
short n;
char *name;
{ int i;
  float hour,minute,sec,p;
  float time_unit,pressure_unit,first_time;
  char filename[80];
  FILE *in_file;

  strcpy(filename,PATH);
  strcat(filename,name);
  if (!(in_file = fopen(filename,"r"))) return 0 ;

  (void) fscanf(in_file,"(%f %f)",&time_unit,&pressure_unit);
  while (fgetc(in_file)!='\n');

  (void) fscanf(in_file,"(%f %f %f %f)\n",&hour,&minute,&sec,&p);
```

```
  first_time = time_unit*hms_to_s(hour,minute,sec);
  initial_time[n][0]=0.0;
  first_pressure[n] = initial_pressure[n][0] = pressure_unit*p;

  for (i=1;!feof(in_file);i++) {
    (void) fscanf(in_file,"(%f %f %f %f)\n",&hour,&minute,&sec,&p);
    initial_time[n][i] = time_unit*hms_to_s(hour,minute,sec)-first_time;
    initial_pressure[n][i] = pressure_unit*p;
    }

  i_last_time[n] = initial_time[n][i-1];
  i_last_pressure[n]=initial_pressure[n][i-1];

  fclose(in_file);
  return i;
}


/*****************************************************************************
/* convert hours, minutes, seconds to seconds                                *
/*****************************************************************************
static float hms_to_s(h,m,s)
float h,m,s;
{ return((float)(3600*h+60*m+s));
}

/*****************************************************************************
/* compute filtered data for well n                                          *
/*****************************************************************************
extern void filter_data(n)
short n;
{ int i,j,k,last_j;
  double coeff;

  coeff = log10(i_last_time[n])/(MAX_NBR_PTS-1);

  time[n][0] = 0;
  pressure[n][0] = first_pressure[n];
  for (i=1,j=1,k=1;i<=MAX_NBR_PTS-1;i++) {
    for (last_j=j;
         initial_time[n][j]<=(fl0(coeff*i)+EPSILON)
         && initial_time[n][j];
         j++) {
           time[n][k] += initial_time[n][j];
           pressure[n][k] += initial_pressure[n][j];
           }
    if (j>last_j) {
      time[n][k] /= (j-last_j);
      pressure[n][k++] /= (j-last_j);
      }
  }
  last_time[n]=time[n][k-1];
  last_pressure[n]=pressure[n][k-1];
}
```

```
/*****************************************************************************
/* derivative function                                                       *
/*****************************************************************************
static float *derivative_func(len,x,y,interv)
int len;
float x[],y[];
int interv;
{ static float d[MAX_NBR_PTS];
  float ix,x0,fx,iy,y0,fy;
  int i,ii,fi;

  for (i=0;i<MAX_NBR_PTS;i++) d[i]=0;

  for (i=1;i<len-1;i++) {
    switch(i) {
      case 1: ii=0;fi=2;break;
      case 2: ii=0;fi=4;break;
      default: switch(len-i) {
                 case 2: ii=len-3;fi=len-1;break;
                 case 3: ii=len-5;fi=len-1;break;
                 default: ii=i-interv;fi=i+interv;break;
                 }
      break;
      }
    ix=x[ii];x0=x[i];fx=x[fi];
    iy=y[ii];y0=y[i];fy=y[fi];
    d[i]=((fx-x0)*(y0-iy)/(x0-ix)+(x0-ix)*(fy-y0)/(fx-x0))/(fx-ix);
    }
  d[0]=(y[1]-y[0])/(x[1]-x[0]);
  d[len-1]=(y[len-1]-y[len-2])/(x[len-1]-x[len-2]);

  return(d);
}



/*****************************************************************************
/* compute pressure derivative for well n ( = dp/dln(t))                     *
/*****************************************************************************
extern void compute_p_derivative(n,interv)
short n,interv;
{ int i;
  float log_t[MAX_NBR_PTS],*deriv;

  for (i=1;time[n][i];i++) log_t[i]=log(time[n][i]);

  deriv=derivative_func(i-1,&log_t[1],&pressure[n][1],interv);

  for (i=1;time[n][i];i++) {
    p_derivative[n][i]= *(deriv+i-1);
    }
}



/*****************************************************************************
```

```
/* examine pressure derivative and change values greater than -0.01 to -0.01   *
/*****************************************************************************
extern void examine_p_derivative(n)
short n;
{ int i;

  for (i=1;time[n][i];i++)
    if (p_derivative[n][i]> -0.01) p_derivative[n][i]= -0.01;
}


/*****************************************************************************
/* compute pressure second derivative for well n ( = dln(dp/dln(t))/dln(t))   */
/*****************************************************************************
extern void compute_p_d_derivative(n,interv)
short n,interv;
{ int i;
  float log_t[MAX_NBR_PTS],log_p_d[MAX_NBR_PTS],*deriv;

  for (i=1;time[n][i];i++) {
    log_t[i]=log(time[n][i]);
    log_p_d[i]=log(-p_derivative[n][i]);
    }

  deriv=derivative_func(i-1,&log_t[1],&log_p_d[1],interv);

  for (i=1;time[n][i];i++)
    p_d_derivative[n][i]= *(deriv+i-1);
}


/*****************************************************************************
/* compute extrema of pressure derivatives for well n                        *
/*****************************************************************************
extern void compute_extrema(n)
short n;
{ int i,j,imax,imin;
  float x,min,max;

  min=max=p_derivative[n][1];
  imin=imax=1;
  for (i=2;time[n][i];i++) {
    if ((x=p_derivative[n][i])<min) {
      min=x;
      imin=i;
      }
    if (x>max) {
      max=x;
      imax=i;
      }
    }
  min_p_derivative[n]=max;
  max_p_derivative[n]=min;
  i_min_p_derivative[n]=imax;
  i_max_p_derivative[n]=imin;
```

```
  min=max=p_d_derivative[n][1];
  imin=imax=1;
  for (i=2;time[n][i];i++) {
    if ((x=p_d_derivative[n][i])<min) {
      min=x;
      imin=i;
      }
    if (x>max) {
      max=x;
      imax=i;
      }
    }
  min_p_d_derivative[n]=max;
  max_p_d_derivative[n]=min;
  i_min_p_d_derivative[n]=imax;
  i_max_p_d_derivative[n]=imin;

  for (i=2,j=0;time[n][i]<1000;i++)
    if (p_d_derivative[n][i-1]>0
     && p_d_derivative[n][i]<0)
       local_max_time[n][j++]=time[n][i];
}



/*************************************************************************
/* compute log of pressure derivative for well n ( = dln(p)/dln(t))      *
/*************************************************************************
extern void compute_log_derivative(n,interv)
short n,interv;
{ int i;
  float log_t[MAX_NBR_PTS],log_p[MAX_NBR_PTS],*deriv;

  for (i=1;time[n][i];i++) {
    log_t[i]=log(time[n][i]);
    log_p[i]=log(first_pressure[n]-pressure[n][i]);
    }

  deriv=derivative_func(i-1,&log_t[1],&log_p[1],interv);

  for (i=1;log_derivative[n][i]= *(deriv+i-1);i++);
}




/*************************************************************************
/* curves                                                                *
/*************************************************************************


/*************************************************************************
/* coordinates computation function                                      *
/*************************************************************************
```

```
static int c_c(x,x0,scale,origin)
float x,x0,scale;
int origin;
{ return (origin+(int)(scale*(x-x0))); }


/*****************************************************************************
/* initial curve for well n                                                  *
/*****************************************************************************
extern void compute_initial_curve(n)
short n;
{ int i;
  float x_scale,y_scale;

  x_scale=MAX_WIDTH/i_last_time[n];
  y_scale=MAX_HEIGHT/(i_last_pressure[n]-first_pressure[n]);

  for (i=0;!i || initial_time[n][i];i++) {
    initial_curve_x[n][i]=c_c(initial_time[n][i],0.0,
                              x_scale,X_ORIGIN);
    initial_curve_y[n][i]=c_c(initial_pressure[n][i],i_last_pressure[n],
                              y_scale,Y_ORIGIN);
    }

  initial_x_axis[n][0].pos=X_ORIGIN;
  initial_x_axis[n][1].pos=X_ORIGIN+MAX_WIDTH;
  sprintf(initial_x_axis[n][0].text,"%.2f",0.0);
  sprintf(initial_x_axis[n][1].text,"%.2f",i_last_time[n]);

  initial_y_axis[n][0].pos=Y_ORIGIN;
  initial_y_axis[n][1].pos=Y_ORIGIN-MAX_HEIGHT;
  sprintf(initial_y_axis[n][0].text,"%.2f",i_last_pressure[n]);
  sprintf(initial_y_axis[n][1].text,"%.2f",first_pressure[n]);
}


/*****************************************************************************
/* filtered curve for well n                                                 *
/*****************************************************************************
extern void compute_filtered_curve(n)
short n;
{ int i;
  float x_scale,y_scale;

  x_scale=MAX_WIDTH/last_time[n];
  y_scale=MAX_HEIGHT/(last_pressure[n]-first_pressure[n]);

  for (i=0;!i || time[n][i];i++) {
    curve_x[n][i]=c_c(time[n][i],0.0,x_scale,X_ORIGIN);
    curve_y[n][i]=c_c(pressure[n][i],last_pressure[n],y_scale,Y_ORIGIN);
    }

  x_axis[n][0].pos=X_ORIGIN;
  x_axis[n][1].pos=X_ORIGIN+MAX_WIDTH;
  sprintf(x_axis[n][0].text,"%.2f",0.0);
  sprintf(x_axis[n][1].text,"%.2f",last_time[n]);
```

```
  y_axis[n][0].pos=Y_ORIGIN;
  y_axis[n][1].pos=Y_ORIGIN-MAX_HEIGHT;
  sprintf(y_axis[n][0].text,"%.2f",last_pressure[n]);
  sprintf(y_axis[n][1].text,"%.2f",first_pressure[n]);
}

/*****************************************************************************
/* compute log scales for well n                                             *
/*****************************************************************************
extern void compute_log_scales(n)
short n;
{ int i,tmax,ymin,ymax;
  float x,min,max;

  log_x_min[n]=(int)(log10(0.98*time[n][1])+10)-10;
  tmax=(int)(log10(last_time[n])+10)-9;
  log_cycles[n]=tmax-log_x_min[n];

  min=max=first_pressure[n]-pressure[n][1];
  for (i=2;time[n][i];i++) {
    if ((x=first_pressure[n]-pressure[n][i])<min) min=x;
    if (x>max) max=x;
    }
  ymin=inf((int)(log10(min)+10)-10,(int)(log10(-min_p_derivative[n])+10)-10);
  ymax=sup((int)(log10(max)+10)-9,(int)(log10(-max_p_derivative[n])+10)-9);

  switch (log_cycles[n]-(ymax-ymin)) {
    case 0: log_y_min[n]=ymin;break;
    case 1: log_y_min[n]=ymin;break;
    case 2: log_y_min[n]=ymin-1;break;
    case 3: log_y_min[n]=ymin-1;break;
    case 4: log_y_min[n]=ymin-2;break;
    case 5: log_y_min[n]=ymin-2;break;
    case 6: log_y_min[n]=ymin-3;break;
    default: log_y_min[n]=ymin-4;break;
    }
}


/*****************************************************************************
/* semilog curve for well n                                                  *
/*****************************************************************************
extern void compute_semilog_curve(n)
short n;
{ int i;
  float x_scale,y_scale,p0;

  p0=first_pressure[n]-pressure[n][1];
  x_scale=MAX_WIDTH/(log_cycles[n]);
  y_scale=MAX_HEIGHT/(last_pressure[n]-pressure[n][1]);

  for (i=1;time[n][i];i++) {
    curve_log_x[n][i-1]=c_c(log10(time[n][i]),
                            (float)log_x_min[n],
                            x_scale,X_ORIGIN);
```

```
    curve_y1[n][i-1]=c_c(first_pressure[n]-pressure[n][i],p0,y_scale,Y_ORIGIN);
    }

  for (i=0;i<=log_cycles[n];i++) {
    log_x_axis[n][i].pos=X_ORIGIN+i*x_scale;
    }
  sprintf(log_x_axis[n][0].text,"10 %d",log_x_min[n]);
  sprintf(log_x_axis[n][log_cycles[n]].text,"    10 %d",
                                          log_x_min[n]+log_cycles[n]);

  y1_axis[n][0].pos=Y_ORIGIN;
  y1_axis[n][1].pos=Y_ORIGIN-MAX_HEIGHT;
  sprintf(y1_axis[n][0].text,"%.2f",p0);
  sprintf(y1_axis[n][1].text,"%.2f",first_pressure[n]-last_pressure[n]);
}


/*****************************************************************************
/* loglog curve for well n                                                   *
/*****************************************************************************
extern void compute_loglog_curve(n)
short n;
{ int i;
  float y_scale;

  y_scale=MAX_HEIGHT/(log_cycles[n]);

  for (i=1;time[n][i];i++)
    curve_log_y[n][i-1]=c_c(log10(first_pressure[n]-pressure[n][i]),
                            (float)log_y_min[n],
                            -y_scale,Y_ORIGIN);

  for (i=0;i<=log_cycles[n];i++) {
    log_y_axis[n][i].pos=Y_ORIGIN-i*y_scale;
    }
  sprintf(log_y_axis[n][0].text,"10 %d",log_y_min[n]);
  sprintf(log_y_axis[n][log_cycles[n]].text,"10 %d",
                                          log_y_min[n]+log_cycles[n]);
}


/*****************************************************************************
/* derivative curve for well n                                               *
/*****************************************************************************
extern void compute_derivative_curve(n)
short n;
{ int i;
  float y_scale;

  y_scale=MAX_HEIGHT/(log_cycles[n]);

  for (i=1;time[n][i];i++)
    derivative_log_y[n][i-1]=c_c(log10(-p_derivative[n][i]),
                                 (float)log_y_min[n],
                                 -y_scale,Y_ORIGIN);
```

```
}
```

```
/*****************************************************************************
/* "cmp_main.c" : main procedures for computations                            *
/*****************************************************************************
#define CMP_MAIN

#include "cmp.h"

/*****************************************************************************
/* functions references                                                      *
/*****************************************************************************
extern int read_well();
extern void filter_data();
extern void compute_p_derivative();
extern void examine_p_derivative();
extern void compute_p_d_derivative();
extern void compute_extrema();
extern void compute_log_derivative();
extern void compute_initial_curve();
extern void compute_filtered_curve();
extern void compute_log_scales();
extern void compute_semilog_curve();
extern void compute_loglog_curve();
extern void compute_derivative_curve();

extern void semilog_straight_lines();
extern void loglog_straight_lines();
extern void derivative_straight_lines();
extern void seek_humps();

extern void send_number();
extern void send_initial_data();
extern void send_semilog_staight_lines();
extern void send_loglog_straight_lines();
extern void send_derivative_straight_lines();
extern void send_humps();
extern void send_local_max_time();
extern void send_quit();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void do_well_analysis();


/*****************************************************************************
/* begin the analysis for well n                                             *
/*****************************************************************************
extern void do_well_analysis(n,name)
int n;
char *name;
{
  send_number(n,name);

  (void) read_well(n,name);
```

```
    filter_data(n);
    compute_p_derivative(n,INTERV);
    examine_p_derivative(n);
    compute_p_d_derivative(n,INTERV);
    compute_extrema(n);
    compute_log_derivative(n,INTERV);
    send_initial_data(n);
    send_local_max_time(n);

    semilog_straight_lines(n);
    send_semilog_straight_lines(n);
    loglog_straight_lines(n);
    send_loglog_straight_lines(n);
    derivative_straight_lines(n);
    send_derivative_straight_lines(n);
    seek_humps(n);
    send_humps(n);

    compute_initial_curve(n);
    compute_filtered_curve(n);
    compute_log_scales(n);
    compute_semilog_curve(n);
    compute_loglog_curve(n);
    compute_derivative_curve(n);
}
```

```
/*****************************************************************************
/* "cmp_pattern.c" : look for pattern in semilog, loglog and derivative curves.*
/*****************************************************************************

#define PAT_MAIN /* set flag for pattern declarations */

#include <stdio.h>
#include <math.h>
#include "cmp.h"
#include "pat.h"
#include "macro.h"


/*****************************************************************************
/* references                                                                *
/*****************************************************************************


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern float slope();
  static struct s_line *straight_lines();
  static void group_s_s_lines();
  static void check_s_s_lines();
extern void semilog_straight_lines();
extern void loglog_straight_lines();
  static int expand_d_s_lines_left();
  static int group_d_s_lines();
  static void modify_d_s_lines();
extern void derivative_straight_lines();

  static void modify_extrema();
extern void seek_humps();



/*****************************************************************************
/* staight lines                                                             *
/*****************************************************************************

/*****************************************************************************
/* compute the slope of a straight line                                      *
/*****************************************************************************
extern float slope(c,n,pos)
char c;
int n,pos;
{ int ii,fi;
  if (c=='s') {
    ii=semilog_s_lines[n][pos].i;
    fi=semilog_s_lines[n][pos].f;
    return (pressure[n][ii]-pressure[n][fi])
           /(log10(time[n][ii])-log10(time[n][fi]));
    }

  if (c=='l') {
```

```
    ii=loglog_s_lines[n][pos].i;
    fi=loglog_s_lines[n][pos].f;
    return (log10(abs(pressure[n][fi]))
            -log10(abs(pressure[n][ii])))
           /(log10(time[n][fi])-log10(time[n][ii]));
    }

  if (c=='d') {
    ii=derivative_s_lines[n][pos].i;
    fi=derivative_s_lines[n][pos].f;
    return (log10(-p_derivative[n][fi])-log10(-p_derivative[n][ii]))
           /(log10(time[n][fi])-log10(time[n][ii]));
    }
}


/*****************************************************************************
/* look for straight lines                                                   *
/*****************************************************************************
static struct s_line *straight_lines(n,deriv,sig_length,error)
int n;
float deriv[],sig_length,error;
{ static struct s_line sl[MAX_NBR_S_LINES];
  int i,j,k;
  float sum,average,current,first;
  for (i=1,k=0;time[n][i];i=j) {
    first=deriv[i];
    sum=first;
    average=first;

    for (j=i+1;time[n][j] && abs((current=deriv[j])-average)<error
                                  && abs(average-first)<error;
         j++) {
           sum += current;
           average = sum/(j-i+1);
           }

    if (log10(time[n][j-1])-log10(time[n][i])>sig_length) {
      sl[k].i=i;
      sl[k++].f=j-1;
      }
    }
  sl[k].i=0;


  return(sl);
}

/*****************************************************************************
/* group straight lines in the semilog curve for well n                      *
/*****************************************************************************
static void group_s_s_lines(n)
int n;
{ int i,ii1,fi1,ii2,fi2,mi,sl_number;
  float x=0;
```

```
  for (i=1;time[n][i];i++) x += p_derivative[n][i];
  x /= i-1;

  for (i=0;semilog_s_lines[n][i].i;i++);
  sl_number=i;

  for (i=sl_number-1;i>0;i--) {
    ii1=semilog_s_lines[n][i-1].i;
    fi1=semilog_s_lines[n][i-1].f;
    ii2=semilog_s_lines[n][i].i;
    fi2=semilog_s_lines[n][i].f;
    mi=(int)((ii1+fi2)/2);
    if (abs(slope('s',n,i)-slope('s',n,i-1))<ABS_ERR
    && (log10(time[n][ii2])-log10(time[n][fi1]))<0.25
    && abs(pressure[n][mi]-(pressure[n][ii1]
                           +(pressure[n][fi2]-pressure[n][ii1])
                            *(log10(time[n][mi])-log10(time[n][ii1]))
                            /(log10(time[n][fi2])-log10(time[n][ii1]))))
       < abs(x*REL_ERR)) {
      semilog_s_lines[n][i-1].f=fi2;
      semilog_s_lines[n][i].i=0;
      }
    }
}


/*****************************************************************************
/* check straight lines in the semilog curve for well n                      *
/*****************************************************************************
static void check_s_s_lines(n)
int n;
{ int i,j,ii,fi,mi,sl_number;
  float x=0;

  for (i=1;time[n][i];i++) x += p_derivative[n][i];
  x /= i-1;

  for (i=0;semilog_s_lines[n][i].i;i++);
  sl_number=i;

  for (i=0;ii=semilog_s_lines[n][i].i;i++) {
    fi=semilog_s_lines[n][i].f;
    mi=(int)((ii+fi)/2);
    while ((fi-ii)>2
         && (log10(time[n][fi])-log10(time[n][ii]))>2*SIG_LENGTH
         && abs(pressure[n][mi]-(pressure[n][ii]
                                 + slope('s',n,i)
                                 *(log10(time[n][mi])-log10(time[n][ii]))))
            > abs(x*REL_ERR)) {
      for (j=sl_number-1;j>i;j--) {
        semilog_s_lines[n][j+1].i=semilog_s_lines[n][j].i;
        semilog_s_lines[n][j+1].f=semilog_s_lines[n][j].f;
        }
      sl_number++;
      semilog_s_lines[n][i+1].i=mi;
      semilog_s_lines[n][i+1].f=fi;
```

```
      fi=semilog_s_lines[n][i].f=mi;
      mi=(int)((ii+fi)/2);
      }
  }
}

/*****************************************************************************
/* straight lines in the semilog curve for well n                            *
/*****************************************************************************
extern void semilog_straight_lines(n)
int n;
{ struct s_line *sl;
  int i,ii,fi;
  float x=0;

  for (i=1;time[n][i];i++) x += p_derivative[n][i];
  x /= i-1;

  sl=straight_lines(n,&p_derivative[n][0],(float) SIG_LENGTH,
                    (float) sup(abs(REL_ERR*x),ABS_ERR));

  for (i=0;ii=semilog_s_lines[n][i].i=sl[i].i;i++) {
    fi=semilog_s_lines[n][i].f=sl[i].f;
    }
  group_s_s_lines(n);
  check_s_s_lines(n);
}



/*****************************************************************************
/* straight lines in the loglog curve for well n                             */
/*****************************************************************************
extern void loglog_straight_lines(n)
int n;
{ struct s_line *sl;
  int i,ii,fi;
  float x=0;

  for (i=1;time[n][i];i++) x += log_derivative[n][i];
  x /= i-1;

  sl=straight_lines(n,&log_derivative[n][0],(float) SIG_LENGTH,
                    (float) sup(abs(REL_ERR*x),ABS_ERR));

  for (i=0;ii=loglog_s_lines[n][i].i=sl[i].i;i++) {
    fi=loglog_s_lines[n][i].f=sl[i].f;
    }
}


/*****************************************************************************
/* expand derivative straight lines to the left                              *
/*****************************************************************************
static int expand_d_s_lines_left(n)
int n;
```

```
{ int i,ii,fi1,flag=0;

  ii=derivative_s_lines[n][0].i;
  while (ii>1
         && abs(slope('d',n,0)-p_d_derivative[n][ii-1])<ABS_ERR) {
    derivative_s_lines[n][0].i= --ii;
    flag=1;
    }

  for (i=1;ii=derivative_s_lines[n][i].i;i++) {
    fi1=derivative_s_lines[n][i-1].f;
    while ((log10(time[n][ii])-log10(time[n][fi1]))>0.2
        && abs(slope('d',n,i)-p_d_derivative[n][ii-1])<ABS_ERR) {
      derivative_s_lines[n][i].i= --ii;
      flag=1;
      }
    }
  return flag;
}


/*************************************************************************
/* group derivative straight lines                                       *
/*************************************************************************
static int group_d_s_lines(n)
int n;
{ int i,fi1,ii2,sl_number,flag=0;

  for (i=0;derivative_s_lines[n][i].i;i++);
  sl_number=i;

  for (i=sl_number-1;i>0;i--) {
    fi1=derivative_s_lines[n][i-1].f;
    ii2=derivative_s_lines[n][i].i;
    if (abs(slope('d',n,i)-slope('d',n,i-1))<ABS_ERR
    && (log10(time[n][ii2])-log10(time[n][fi1]))<0.25
    && abs(p_derivative[n][ii2]-p_derivative[n][fi1])
     <(abs(max_p_derivative[n]/10))) {
      derivative_s_lines[n][i-1].f=derivative_s_lines[n][i].f;
      derivative_s_lines[n][i].i=0;
      flag=1;
      }
    }
  return flag;
}


/*************************************************************************
/* modify derivative straight lines                                      *
/*************************************************************************
static void modify_d_s_lines(n)
int n;
{ if (group_d_s_lines(n))
    if (expand_d_s_lines_left(n)) modify_d_s_lines(n);
}
```

```
/*****************************************************************************
/* straight lines in the derivative curve for well n                         *
/*****************************************************************************
extern void derivative_straight_lines(n)
int n;
{ struct s_line *sl;
  int i,ii,fi;
  float x=0;

  for (i=1;time[n][i];i++) x += p_d_derivative[n][i];
  x /= i-1;

  sl=straight_lines(n,&p_d_derivative[n][0],(float) SIG_LENGTH,
                    (float) (2.3*sup(0.43*abs(REL_ERR*x),ABS_ERR)));

  for (i=0;ii=derivative_s_lines[n][i].i=sl[i].i;i++) {
    fi=derivative_s_lines[n][i].f=sl[i].f;
    }
  (void) expand_d_s_lines_left(n);
  modify_d_s_lines(n);
}


/*****************************************************************************
/* modify extrema                                                            *
/*****************************************************************************
static void modify_extrema(n)
int n;
{ int i,ii,ti;
  float diff;

  for (i=0;ii=humps[n][i].i;i++) {
    ti=humps[n][i].t;
    diff=p_derivative[n][ii]-p_derivative[n][ti];
    if (diff>0)
      while (p_derivative[n][ti-1]<p_derivative[n][ti]
          || p_derivative[n][ti+1]<p_derivative[n][ti]) {
        if (p_derivative[n][ti-1]<p_derivative[n][ti]) humps[n][i].t= --ti;
        else humps[n][i].t= ++ti;
       }
    else
      while (p_derivative[n][ti-1]>p_derivative[n][ti]
          || p_derivative[n][ti+1]>p_derivative[n][ti]) {
        if (p_derivative[n][ti-1]>p_derivative[n][ti]) humps[n][i].t= --ti;
        else humps[n][i].t= ++ti;
       }
  }
}


/*****************************************************************************
/* humps                                                                     *
/*****************************************************************************
extern void seek_humps(n)
int n;
```

```
{ int i,ii,top,j,k,inoise,fnoise;
  float first,current,next,bef_noise;

  for (i=1,k=0;time[n][i];i++) {
    ii=i;
    first=bef_noise=p_d_derivative[n][i];
    inoise=fnoise=i;

    for (j=i,top=0;time[n][j+1]
             && (log10(time[n][fnoise])-log10(time[n][inoise]))<SIG_LENGTH;
         j++) {
      if ((first>0
        && (current=p_d_derivative[n][j])<bef_noise
        && (next=p_d_derivative[n][j+1])<current)
      ||
          (first<0
        && (current=p_d_derivative[n][j])>bef_noise
        && (next=p_d_derivative[n][j+1])>current)
         ) {
        bef_noise=current;
        inoise=fnoise=j;
        }
      else
        fnoise=j;

      if (!top && signum(current)!=signum(first)) top=j;

      if ((log10(time[n][fnoise])-log10(time[n][inoise]))<=(SIG_LENGTH/2)
       && (!top || (log10(time[n][j])-log10(time[n][top]))<=SIG_LENGTH))
        i=j;
      }

   if (top && (log10(time[n][j-1])-log10(time[n][ii]))>=(3*SIG_LENGTH)
           && (log10(time[n][j-1])-log10(time[n][top]))>=SIG_LENGTH
           && (log10(time[n][top])-log10(time[n][ii]))>=SIG_LENGTH
           && sup(abs(log10(-p_derivative[n][j-1])-log10(-p_derivative[n][top]))
                  abs(log10(-p_derivative[n][ii])-log10(-p_derivative[n][top])))
              >=(SIG_LENGTH/2)) {
     humps[n][k].i=ii;
     humps[n][k].t=top;
     humps[n][k++].f=inoise+1;
     }
  }
  modify_extrema(n);
}
```

```
/*****************************************************************************
/* "cmp_send.c" : functions for interface between C & ART.                   *
/*****************************************************************************
#include <stdio.h>
#include <string.h>
#include "cmp.h"
#include "pat.h"

/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern float slope();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
static void send();
extern void send_number();
extern void send_initial_data();
extern void send_semilog_straight_lines();
extern void send_loglog_straight_lines();
extern void send_derivative_straight_lines();
extern void send_humps();
extern void send_quit();


/*****************************************************************************
/* send string to ART                                                        *
/*****************************************************************************
static void send(buf)
char *buf;
{ write(1,buf,strlen(buf));
}


/*****************************************************************************
/* send well number                                                          *
/*****************************************************************************
extern void send_number(n,name)
int n;
char *name;
{ char buf[80];

  sprintf(buf,"(1 %d %s)",n,name);
  send(buf);
}


/*****************************************************************************
/* send initial data                                                         *
/*****************************************************************************
extern void send_initial_data(n)
int n;
{ char buf[200];
  int last,imax=i_max_p_derivative[n];
```

```
  for (last=1;time[n][last];last++);

  sprintf(buf,"(2 %d %f (%f %f) (%f) (%f %f %f %f) (%f %f))",
              n,
              last_time[n],
              log_derivative[n][1],log_derivative[n][2],
              time[n][1],
              p_d_derivative[n][1],p_d_derivative[n][2],
              p_d_derivative[n][last-2],p_d_derivative[n][last-1],
              time[n][imax],max_p_derivative[n]);
  send(buf);
}


/*****************************************************************************
/* send semilog straight lines                                               *
/*****************************************************************************
extern void send_semilog_straight_lines(n)
int n;
{ int sl_number,i,ii,fi;
  char buf[1000],aux[200];

  for (sl_number=0;semilog_s_lines[n][sl_number].i;sl_number++);

  if (sl_number) {
    sprintf(buf,"(3 %d %d",n,sl_number);
    for (i=0;i<sl_number;i++) {
      ii=semilog_s_lines[n][i].i;
      fi=semilog_s_lines[n][i].f;
      sprintf(aux," ((%f %f) (%f %f) %f)",
                time[n][ii],pressure[n][ii],
                time[n][fi],pressure[n][fi],
                slope('s',n,i));
      strcat(buf,aux);
      }
    strcat(buf,")");
    send(buf);
    }
}


/*****************************************************************************
/* send loglog straight lines                                                */
/*****************************************************************************
extern void send_loglog_straight_lines(n)
int n;
{ int sl_number,i,ii,fi;
  char buf[1000],aux[200];

  for (sl_number=0;loglog_s_lines[n][sl_number].i;sl_number++);

  if (sl_number) {
    sprintf(buf,"(4 %d %d",n,sl_number);
    for (i=0;i<sl_number;i++) {
      ii=loglog_s_lines[n][i].i;
```

```
      fi=loglog_s_lines[n][i].f;
      sprintf(aux," ((%f %f) (%f %f) %f)",
                time[n][ii],pressure[n][ii],
                time[n][fi],pressure[n][fi],
                slope('l',n,i));
      strcat(buf,aux);
      }
    strcat(buf,")");
    send(buf);
    }
}


/*****************************************************************************
/* send derivative straight lines                                            *
/*****************************************************************************
extern void send_derivative_straight_lines(n)
int n;
{ int sl_number,i,ii,fi;
  char buf[1000],aux[200];

  for (sl_number=0;derivative_s_lines[n][sl_number].i;sl_number++);

  if (sl_number) {
    sprintf(buf,"(5 %d %d",n,sl_number);
    for (i=0;i<sl_number;i++) {
      ii=derivative_s_lines[n][i].i;
      fi=derivative_s_lines[n][i].f;
      sprintf(aux," ((%f %f) (%f %f) %f)",
                time[n][ii],p_derivative[n][ii],
                time[n][fi],p_derivative[n][fi],
                slope('d',n,i));
      strcat(buf,aux);
      }
    strcat(buf,")");
    send(buf);
    }
}


/*****************************************************************************
/* send humps                                                                *
/*****************************************************************************
extern void send_humps(n)
int n;
{ int h_number,i,ii,ti,fi;
  char buf[1000],aux[200];

  for (h_number=0;humps[n][h_number].i;h_number++);

  if (h_number) {
    sprintf(buf,"(6 %d %d",n,h_number);
    for (i=0;i<h_number;i++) {
      ii=humps[n][i].i;
      ti=humps[n][i].t;
      fi=humps[n][i].f;
```

```
      if (p_d_derivative[n][ii]>0) {
        sprintf(aux," (hill (%f %f) (%f %f) (%f %f))",
                  time[n][ii],p_derivative[n][ii],
                  time[n][fi],p_derivative[n][fi],
                  time[n][ti],p_derivative[n][ti]);
        }
      else {
        sprintf(aux," (valley (%f %f) (%f %f) (%f %f))",
                  time[n][ii],p_derivative[n][ii],
                  time[n][fi],p_derivative[n][fi],
                  time[n][ti],p_derivative[n][ti]);
        }
      strcat(buf,aux);
      }
    strcat(buf,")");
    send(buf);
    }
}


/*****************************************************************************
/* send local maximum of pressure derivative                                 *
/*****************************************************************************
extern void send_local_max_time(n)
int n;
{ int number,i;
  char buf[1000],aux[200];

  for (number=0;local_max_time[n][number];number++);

  if (number) {
    sprintf(buf,"(7 %d (%f",n,local_max_time[n][0]);
    for (i=1;i<number;i++) {
      sprintf(aux," %f",local_max_time[n][i]);
      strcat(buf,aux);
      }
    strcat(buf,"))");
    send(buf);
    }
}


/*****************************************************************************
/* send end of execution                                                     *
/*****************************************************************************
extern void send_quit()
{ char buf[20];

  sprintf(buf,"(8)");
  send(buf);
}
```

```
/*****************************************************************************
/* "env.h" : Set environment variables                                       *
/*****************************************************************************

#define PATH "/usr/lola/dm/antoine/Wes/Data/"
#define WELLS_LIST "/usr/lola/dm/antoine/Wes/Data/wells.list"
```

```
/*****************************************************************************
/* "gph.h" : Declarations for graphics                                       */
/*****************************************************************************
#include <suntool/sunview.h>
#include <suntool/canvas.h>

#ifdef GPH_MAIN
#define EXTGPH
#else
#define EXTGPH extern
#endif

#define VERTICAL_SIZE 500
#define HORIZONTAL_SIZE 500
#define X_ORIGIN 25
#define Y_ORIGIN 475

EXTGPH Pixwin *pw[MAX_NBR_WINDOWS];
```

```
/*****************************************************************************
/* "gph_main.c" : Curves drawing.                                            *
/*****************************************************************************
#include "cmp.h"
#include "pat.h"


/*****************************************************************************
/* functions references                                                      *
/*****************************************************************************


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void draw_initial_curve();
extern void draw_filtered_curve();
extern void draw_semilog_curve();
extern void draw_semilog_patterns();
extern void draw_loglog_curve();
extern void draw_loglog_patterns();
extern void draw_derivative_curve();
extern void draw_derivative_patterns();


/*****************************************************************************
/* draw the initial curve                                                    *
/*****************************************************************************
extern void draw_initial_curve(n)
int n;
{

  plot(n,&initial_curve_x[n][0],
         &initial_curve_y[n][0],
         &initial_x_axis[n][0],
         &initial_y_axis[n][0],
         "Initial");
}


/*****************************************************************************
/* draw the filtered curve                                                   *
/*****************************************************************************
extern void draw_filtered_curve(n)
int n;
{

  plot(n,&curve_x[n][0],
         &curve_y[n][0],
         &x_axis[n][0],
         &y_axis[n][0],
         "Filtered");
}


/*****************************************************************************
```

```
/* draw the semilog curve                                                    *
/***************************************************************************
extern void draw_semilog_curve(n)
int n;
{

  plot(n,&curve_log_x[n][0],
         &curve_y1[n][0],
         &log_x_axis[n][0],
         &y1_axis[n][0],
         "Semilog");
}


/***************************************************************************
/* draw the semilog patterns                                                *
/***************************************************************************
extern void draw_semilog_patterns(n)
int n;
{
  plot_straight_lines(n,&semilog_s_lines[n][0],
                        &curve_log_x[n][0],
                        &curve_y1[n][0]);
}


/***************************************************************************
/* draw the loglog curve                                                    *
/***************************************************************************
extern void draw_loglog_curve(n)
int n;
{

  plot(n,&curve_log_x[n][0],
         &curve_log_y[n][0],
         &log_x_axis[n][0],
         &log_y_axis[n][0],
         "Loglog");
}


/***************************************************************************
/* draw the loglog patterns                                                 *
/***************************************************************************
extern void draw_loglog_patterns(n)
int n;
{

  plot_straight_lines(n,&loglog_s_lines[n][0],
                        &curve_log_x[n][0],
                        &curve_log_y[n][0]);
}


/***************************************************************************
/* draw the derivative curve                                                *
```

```
/*****************************************************************************
extern void draw_derivative_curve(n)
int n;
{

  plot(n,&curve_log_x[n][0],
         &derivative_log_y[n][0],
         &log_x_axis[n][0],
         &log_y_axis[n][0],
         "Derivative");
}


/*****************************************************************************
/* draw the derivative patterns                                              *
/*****************************************************************************
extern void draw_derivative_patterns(n)
int n;
{

  plot_straight_lines(n,&derivative_s_lines[n][0],
                        &curve_log_x[n][0],
                        &derivative_log_y[n][0]);
  plot_humps(n,&humps[n][0],
               &curve_log_x[n][0],
               &derivative_log_y[n][0]);
}
```

```
/*****************************************************************************
/* "gph_util.c" : Drawing functions                                          *
/*****************************************************************************
#include "cmp.h"
#include "win.h"
#include "gph.h"
#include "pat.h"

/*****************************************************************************
/* references                                                                *
/*****************************************************************************


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void plot();
extern void plot_straight_lines();
extern void plot_humps();


/*****************************************************************************
/* plot curve y versus x, x axis and y axis                                  *
/*****************************************************************************
extern void plot(w_number,x,y,x_ax,y_ax,title)
int w_number;
short x[],y[];
struct axis_mark x_ax[],y_ax[];
char *title;
{ int i,pos,n=w_window[w_number];

  pw_writebackground(pw[n],0,0,HORIZONTAL_SIZE-1,VERTICAL_SIZE-1,PIX_SRC);

  for (i=1;x[i];i++)
    pw_vector(pw[n],x[i-1],y[i-1],x[i],y[i],PIX_SRC,1);

  pw_vector(pw[n],X_ORIGIN,Y_ORIGIN,X_ORIGIN+MAX_WIDTH,Y_ORIGIN,PIX_SRC,1);
  for (i=0;(pos=x_ax[i].pos);i++) {
    pw_vector(pw[n],pos,Y_ORIGIN-2,pos,Y_ORIGIN+2,PIX_SRC,1);
    }
  pw_text(pw[n],X_ORIGIN,Y_ORIGIN+10,PIX_SRC,scale_font,x_ax[0].text);
  pw_text(pw[n],X_ORIGIN+MAX_WIDTH-40,Y_ORIGIN+10,
            PIX_SRC,scale_font,x_ax[i-1].text);

  pw_vector(pw[n],X_ORIGIN,Y_ORIGIN,X_ORIGIN,Y_ORIGIN-MAX_HEIGHT,PIX_SRC,1);
  for (i=0;(pos=y_ax[i].pos);i++) {
    pw_vector(pw[n],X_ORIGIN-2,pos,X_ORIGIN+2,pos,PIX_SRC,1);
    }
  pw_text(pw[n],X_ORIGIN-20,Y_ORIGIN-5,PIX_SRC,scale_font,y_ax[0].text);
  pw_text(pw[n],X_ORIGIN-20,Y_ORIGIN-MAX_HEIGHT-5,
            PIX_SRC,scale_font,y_ax[i-1].text);

  pw_text(pw[n],200,Y_ORIGIN+20,PIX_SRC,0,title);
}
```

```
/*****************************************************************************
/* draw straight lines                                                       *
/*****************************************************************************
extern void plot_straight_lines(w_number,sl,x,y)
int w_number;
struct s_line sl[];
short x[],y[];
{ int i,ii,fi,n=w_window[w_number];

  for (i=0;(ii=sl[i].i-1)!=-1;i++) {
    fi=sl[i].f-1;
    pw_vector(pw[n],x[ii],y[ii],x[fi],y[fi],PIX_SRC,1);
    pw_vector(pw[n],x[ii],y[ii]-1,x[fi],y[fi]-1,PIX_SRC,1);
    pw_vector(pw[n],x[ii]-1,y[ii],x[fi]-1,y[fi],PIX_SRC,1);
    pw_vector(pw[n],x[ii],y[ii]+1,x[fi],y[fi]+1,PIX_SRC,1);
    pw_vector(pw[n],x[ii]+1,y[ii],x[fi]+1,y[fi],PIX_SRC,1);
    }
}


/*****************************************************************************
/* draw humps                                                                *
/*****************************************************************************
extern void plot_humps(w_number,h,x,y)
int w_number;
struct hump h[];
short x[],y[];
{ int i,ii,ti,fi,n=w_window[w_number];

  for (i=0;(ii=h[i].i-1)!=-1;i++) {
    ti=h[i].t-1;
    fi=h[i].f-1;
    if (abs(y[ii]-y[ti])>abs(y[fi]-y[ti])) {
      pw_vector(pw[n],x[ii],y[ii],x[fi],y[ii],PIX_SRC,1);
      pw_vector(pw[n],x[ii],y[ti],x[fi],y[ti],PIX_SRC,1);
      pw_vector(pw[n],x[ii],y[ii],x[ii],y[ti],PIX_SRC,1);
      pw_vector(pw[n],x[fi],y[ii],x[fi],y[ti],PIX_SRC,1);
      }
    else {
      pw_vector(pw[n],x[ii],y[fi],x[fi],y[fi],PIX_SRC,1);
      pw_vector(pw[n],x[ii],y[ti],x[fi],y[ti],PIX_SRC,1);
      pw_vector(pw[n],x[ii],y[fi],x[ii],y[ti],PIX_SRC,1);
      pw_vector(pw[n],x[fi],y[fi],x[fi],y[ti],PIX_SRC,1);
      }
  }
}
```

```
/*****************************************************************************
/* "macro.h" : macro-functions definitions. Date : 05/09/88                  *
/*****************************************************************************


#define abs(x) (((x)>0) ? (x) : -(x))
#define sup(a,b) (((a)>(b)) ? (a) : (b))
#define inf(a,b) (((a)<(b)) ? (a) : (b))
#define signum(x) (((x)>0) ? 1 : (((x)<0) ? -1 : 0))
#define fl0(x) (exp((x)*M_LN10))
```

```
/*****************************************************************************
/* "main.c" : function main().                                               *
/*****************************************************************************

/*****************************************************************************
/* functions references                                                      *
/*****************************************************************************
extern void create_windows();
extern void start_loop();

/*****************************************************************************
/* main function                                                             *
/*****************************************************************************
void main()
{
  create_windows();
  start_loop();

  exit(0);
}
```

```
/*************************************************************************
/* "global_pattern.h" : global declarations for pattern. Date : 05/05/88      *
/*************************************************************************

#ifdef PAT_MAIN
#define EXTPAT
#else
#define EXTPAT extern
#endif


#define SIG_LENGTH 0.25
#define ABS_ERR 0.05
#define REL_ERR 0.2


/*************************************************************************
/* straight lines                                                         *
/*************************************************************************

#define MAX_NBR_S_LINES 10


EXTPAT struct s_line {int i,f;};

EXTPAT struct s_line semilog_s_lines[MAX_NBR_WELLS][MAX_NBR_S_LINES];
EXTPAT struct s_line loglog_s_lines[MAX_NBR_WELLS][MAX_NBR_S_LINES];
EXTPAT struct s_line derivative_s_lines[MAX_NBR_WELLS][MAX_NBR_S_LINES];


/*************************************************************************
/* humps                                                                  *
/*************************************************************************

#define MAX_NBR_HUMPS 10


EXTPAT struct hump {int i,t,f;};

EXTPAT struct hump humps[MAX_NBR_WELLS][MAX_NBR_HUMPS];
```

```
/*****************************************************************************
/* "win.h" : Declarations for windows.                                       *
#include <suntool/sunview.h>
#include <suntool/panel.h>

#ifdef WIN_MAIN
#define EXTWIN
#else
#define EXTWIN extern
#endif

#define MAX_NBR_WELLS 20
#define MAX_NBR_WINDOWS 5
#define MAX_CHAR_NAME 15
#define DIM_SQ 18

EXTWIN Frame base_frame,
             w_frame[MAX_NBR_WINDOWS],
             confirmer,
             closer,
             expl_bar;

EXTWIN Panel_item square[MAX_NBR_WELLS];

EXTWIN struct pixrect *image[MAX_NBR_WELLS][3],
                      *closed_square,
                      *opened_square;

EXTWIN char mess[MAX_NBR_WELLS][3][200];

EXTWIN Pixfont *scale_font;

EXTWIN char w_name[MAX_NBR_WELLS][MAX_CHAR_NAME];
EXTWIN int w_analysed[MAX_NBR_WELLS];
EXTWIN int w_window[MAX_NBR_WELLS];
EXTWIN int nbr_wells;
EXTWIN int bound_frame[MAX_NBR_WINDOWS];
```

```
/*****************************************************************************
/* "win_bar.c" : Explanation bar.                                            *
/*****************************************************************************
#include "win.h"

/*****************************************************************************
/* references                                                                *
/*****************************************************************************


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void create_expl_bar();
extern void expl_mess();


/*****************************************************************************
/* create the explanation bar.                                               *
/*****************************************************************************
extern void create_expl_bar()
{ Rect *r;

  expl_bar = window_create(base_frame,FRAME,
                           FRAME_SHOW_LABEL, TRUE,
                           WIN_SHOW, TRUE,
                           WIN_X,0,
                           WIN_Y,863,
                           WIN_WIDTH,1152,
                           WIN_HEIGHT,18,
                           0);
  r = (Rect *) window_get(expl_bar,WIN_RECT);
  r->r_left = -5;
  window_set(expl_bar,WIN_RECT,r,0);
}


/*****************************************************************************
/* display explanation message                                               *
/*****************************************************************************
extern void expl_mess(message)
char *message;
{ window_set(expl_bar,FRAME_LABEL,message,0);
}
```

```
/*****************************************************************************
/* "win_base.c" : definition of the main window.    Date : 05/26/88          *
/*****************************************************************************
#include "win.h"


/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern void do_well_analysis();
extern void bind_window_with_well();
extern void show_window();
extern void confirm_quit();
extern void destroy_all_windows();
extern void send_quit();

extern void well_button_proc();
extern void quit_button_proc();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void create_base_frame();

  static Menu new_menu();

  static void create_wells_panel();
    static void open_well(); /* procedure for the Notifier */
    static void quit();      /* procedure for the Notifier */

extern void start_loop();


/*****************************************************************************
/* create the main window                                                    *
/*****************************************************************************
void create_base_frame()
{
  base_frame = window_create(NULL,FRAME,
                             FRAME_SHOW_LABEL, FALSE,
                             WIN_X, 5,
                             WIN_Y, 19,
                             0);

  window_set(base_frame,WIN_MENU,new_menu(base_frame),0);

  create_wells_panel(base_frame);

  window_fit(base_frame);
}


/*****************************************************************************
/* set the menu for the main window                                          *
/*****************************************************************************
```

```
static Menu new_menu(frame)
Frame frame;
{ Menu menu;

  menu = (Menu) window_get(frame,WIN_MENU);

  menu_set(menu, MENU_REMOVE,1,
                 MENU_REMOVE,1,
                 MENU_REMOVE,1,
                 MENU_REMOVE,4,
                 0);

  return(menu);
}


/*****************************************************************************
/* create the panel for the list of wells                                    *
/*****************************************************************************
static void create_wells_panel(frame)
Frame frame;
{ Panel panel;
  int i;

  panel = window_create(frame,PANEL,
                  WIN_CONSUME_PICK_EVENTS, WIN_NO_EVENTS,
                                           LOC_MOVE,
                                           LOC_DRAG,
                                           LOC_STILL,
                                           LOC_WINENTER,
                                           LOC_WINEXIT,
                                           LOC_RGNENTER,
                                           LOC_RGNEXIT,
                                           WIN_MOUSE_BUTTONS,
                                           0,
                  0);

  (void) panel_create_item(panel,PANEL_MESSAGE,
                           PANEL_LABEL_STRING, "  LIST OF WELLS",
                           0);

  for (i=0;i<nbr_wells;i++) {
    (void) panel_create_item(panel,PANEL_BUTTON,
                PANEL_CLIENT_DATA, i,
                PANEL_LABEL_IMAGE, image[i][0],
                PANEL_LABEL_X, ATTR_COL(0),
                PANEL_LABEL_Y, ATTR_ROW(i+1),
                PANEL_EVENT_PROC, well_button_proc,
                PANEL_NOTIFY_PROC, open_well,
                0);
    square[i] = panel_create_item(panel,PANEL_BUTTON,
                   PANEL_CLIENT_DATA, i,
                   PANEL_LABEL_IMAGE, closed_square,
                   0);
    }
```

```
  (void) panel_create_item(panel,PANEL_BUTTON,
              PANEL_LABEL_IMAGE, panel_button_image(panel,"Quit",0,0),
              PANEL_LABEL_X, ATTR_COL(0),
              PANEL_LABEL_Y, ATTR_ROW(i+2),
              PANEL_EVENT_PROC, quit_button_proc,
              PANEL_NOTIFY_PROC, quit,
              0);

  window_fit(panel);
}


/************************************************************************
/* open one well                                                        *
/************************************************************************
static void open_well(item,event)
Panel_item item;
Event *event;
{ int w_number=(int) panel_get(item,PANEL_CLIENT_DATA);

  if (!w_analysed[w_number]) {
    w_analysed[w_number]=1;
    do_well_analysis(w_number,w_name[w_number]);
    }

  if (!w_window[w_number])
    bind_window_with_well(w_number);
  show_window(w_number);
}


/************************************************************************
/* end of program                                                       *
/************************************************************************
static void quit()
{ if (confirm_quit()) {
    destroy_all_windows();
    send_quit();
    }
}


/************************************************************************
/* start loop                                                           *
/************************************************************************
extern void start_loop()
{ window_main_loop(base_frame);
}
```

```
/*****************************************************************************
/* "win_closer.c" : Definition of the closer package.                        *
/*****************************************************************************
#include "win.h"

/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern void free_window();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void create_closer();
  static void quit_closer();                /* procedure for the Notifier */

extern void call_closer();
  static void create_closer_items();
  static void close_subframe_from_closer(); /* procedure for the Notifier */


/*****************************************************************************
/* create the closer                                                         *
/*****************************************************************************
extern void create_closer()
{ Panel panel;

  closer = window_create(NULL,FRAME,
                         FRAME_SHOW_LABEL, FALSE,
                         WIN_X, 400,
                         WIN_Y, 400,
                         0);

  panel = window_create(closer,PANEL,0);

  (void) panel_create_item(panel,PANEL_MESSAGE,
                         PANEL_LABEL_STRING, " Too many opened windows. ",
                         0);

  (void) panel_create_item(panel,PANEL_MESSAGE,
                         PANEL_LABEL_STRING, " Please close at least",
                         PANEL_LABEL_X, ATTR_COL(0),
                         PANEL_LABEL_Y, ATTR_ROW(1),
                         0);

  (void) panel_create_item(panel,PANEL_MESSAGE,
                         PANEL_LABEL_STRING, " one of those wells:",
                         PANEL_LABEL_X, ATTR_COL(0),
                         PANEL_LABEL_Y, ATTR_ROW(2),
                         0);

  (void) panel_create_item(panel,PANEL_BUTTON,
                         PANEL_LABEL_IMAGE, panel_button_image(panel,"Ok",0,0),
                         PANEL_LABEL_X, ATTR_COL(0),
```

```
                        PANEL_LABEL_Y, ATTR_ROW(9),
                        PANEL_NOTIFY_PROC, quit_closer,
                        0);

  window_set(closer,WIN_CLIENT_DATA,panel,0);
}


/*************************************************************************
/* quit the closer                                                       *
/*************************************************************************
static void quit_closer(item,event)
Panel_item item;
Event *event;
{ window_return();
}


/*************************************************************************
/* call the closer                                                       *
/*************************************************************************
extern void call_closer()
{ Panel panel;

  panel = (Panel) window_get(closer,WIN_CLIENT_DATA);
  create_closer_items(panel);
  window_loop(closer);
}


/*************************************************************************
/* create the menu of wells                                              *
/*************************************************************************
static void create_closer_items(panel)
Panel panel;
{ int i,w_number;

  for (i=1;i<MAX_NBR_WINDOWS;i++) {
    w_number = (int) window_get(w_frame[i],WIN_CLIENT_DATA);
    (void) panel_create_item(panel,PANEL_BUTTON,
                        PANEL_LABEL_X, ATTR_COL(0),
                        PANEL_LABEL_Y, ATTR_ROW(i+3),
                        PANEL_LABEL_IMAGE, image[w_number][0],
                        PANEL_CLIENT_DATA, i,
                        PANEL_NOTIFY_PROC, close_subframe_from_closer,
                        0);
    }
  window_fit(panel);
  window_fit(closer);
}


/*************************************************************************
/* close a subframe from closer                                          *
/*************************************************************************
static void close_subframe_from_closer(item,event)
```

```
Panel_item item;
Event *event;
{ int f_number = (int) panel_get(item,PANEL_CLIENT_DATA),
      client = (int) window_get(w_frame[f_number],WIN_CLIENT_DATA);

  free_window(f_number);
  panel_set(item,PANEL_LABEL_IMAGE,image[client][1],0);
}
```

```
/*****************************************************************************
/* "win_confirmer.c" : definition of the confirmer window                    *
/*****************************************************************************
#include "win.h"


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void create_confirmer();
  static int yes_no();  /* procedure for the Notifier */

extern int confirm_quit();


/*****************************************************************************
/* create the confirmer                                                      *
/*****************************************************************************
extern void create_confirmer()
{ Panel panel;

  confirmer = window_create(NULL,FRAME,
                                  FRAME_SHOW_LABEL, FALSE,
                                  WIN_X, 400,
                                  WIN_Y, 400,
                                  0);

  panel = window_create(confirmer,PANEL,0);

  (void) panel_create_item(panel,PANEL_MESSAGE,
                                  PANEL_LABEL_STRING,
                                     " Do you really want to quit? ",
                                  0);

  (void) panel_create_item(panel,PANEL_BUTTON,
                                  PANEL_LABEL_X, ATTR_COL(8),
                                  PANEL_LABEL_Y, ATTR_ROW(2),
                                  PANEL_CLIENT_DATA, TRUE,
                                  PANEL_LABEL_IMAGE,
                                    panel_button_image(panel,"Yes",3,0),
                                  PANEL_NOTIFY_PROC, yes_no,
                                  0);

  (void) panel_create_item(panel,PANEL_BUTTON,
                                  PANEL_CLIENT_DATA, FALSE,
                                  PANEL_LABEL_IMAGE,
                                    panel_button_image(panel,"No",3,0),
                                  PANEL_NOTIFY_PROC, yes_no,
                                  0);

  window_fit(panel);
  window_fit(confirmer);
}

/*****************************************************************************
```

```
/* response from the confirmer                                                *
/*****************************************************************************
static int yes_no(item,event)
Panel_item item;
Event *event;
{ window_return((int)panel_get(item,PANEL_CLIENT_DATA));
}

/*****************************************************************************
/* call the confirmer                                                         *
/*****************************************************************************
int confirm_quit()
{ return ((int) window_loop(confirmer));
}
```

```
/*****************************************************************************
/* "event_handle.c" : handle events for panels.                              *
/*****************************************************************************
#include "win.h"

/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern void expl_mess();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void well_button_proc();
extern void quit_button_proc();
extern void opened_button_proc();
extern void closed_button_proc();
extern void initial_button_proc();
extern void filtered_button_proc();
extern void semilog_button_proc();
extern void loglog_button_proc();
extern void derivative_button_proc();


/*****************************************************************************
/* events for list buttons                                                   *
/*****************************************************************************
extern void well_button_proc(item,event)
Panel_item item;
Event *event;
{ int client=(int)panel_get(item,PANEL_CLIENT_DATA);
  static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                      panel_accept_preview(item,event);
                      expl_mess(mess[client][(w_analysed[client])]);
                      panel_begin_preview(item,event);
                      }
                  break;
    case PANEL_EVENT_MOVE_IN: panel_set(item,PANEL_LABEL_IMAGE,
                                  image[client][0],0);
                              panel_begin_preview(item,event);
                              expl_mess(mess[client][(w_analysed[client])]);
                              flag=1;
                              break;
    case PANEL_EVENT_DRAG_IN: panel_set(item,PANEL_LABEL_IMAGE,
                                  image[client][0],0);
                              panel_begin_preview(item,event);
                              expl_mess(mess[client][(w_analysed[client])]);
                              expl_mess(mess[client]);
                              flag=1;
                              break;
    case LOC_STILL: panel_set(item,PANEL_LABEL_IMAGE,
                                  image[client][0],0);
```

```
                    panel_begin_preview(item,event);
                    expl_mess(mess[client][(w_analysed[client])]);
                    flag=1;
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
                                panel_cancel_preview(item,event);
                                panel_set(item,PANEL_LABEL_IMAGE,
                                   image[client][(w_analysed[client])],0);
                                expl_mess("");
                                flag=0;
                                }
                              break;
  case LOC_WINEXIT: if (flag) {
                      panel_cancel_preview(item,event);
                      panel_set(item,PANEL_LABEL_IMAGE,
                         image[client][(w_analysed[client])],0);
                      expl_mess("");
                      flag=0;
                      }
                    break;
  }
}

/*************************************************************************
/* events for quit button                                                *
/*************************************************************************
extern void quit_button_proc(item,event)
Panel_item item;
Event *event;
{ static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                    panel_accept_preview(item,event);
                    flag=0;
                    }
                  break;
    case PANEL_EVENT_MOVE_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess("Back to Art Graphics Studio.");
                                flag=1;
                                }
                              break;
    case PANEL_EVENT_DRAG_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess("Back to Art Graphics Studio.");
                                flag=1;
                                }
                              break;
    case LOC_STILL: if (!flag) {
                      panel_begin_preview(item,event);
                      expl_mess("Back to Art Graphics Studio.");
                      flag=1;
                      }
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
```

```
                                panel_cancel_preview(item,event);
                                expl_mess("");
                                flag=0;
                                }
                              break;
    case LOC_WINEXIT: if (flag) {
                        panel_cancel_preview(item,event);
                        expl_mess("");
                        flag=0;
                        }
                      break;
  }
}

/*************************************************************************
/* events for opened square button                                       *
/*************************************************************************
extern void opened_button_proc(item,event)
Panel_item item;
Event *event;
{ int client = (int) panel_get(item,PANEL_CLIENT_DATA);

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event))
                    panel_accept_preview(item,event);
                  break;
    case PANEL_EVENT_MOVE_IN: expl_mess(mess[client][2]);
                              break;
    case PANEL_EVENT_DRAG_IN: expl_mess(mess[client][2]);
                              break;
    case LOC_STILL: expl_mess(mess[client][2]);
                    break;
    case PANEL_EVENT_CANCEL: expl_mess("");
                             break;
    case LOC_WINEXIT: expl_mess("");
                      break;
  }
}

/*************************************************************************
/* events for closed square button                                       *
/*************************************************************************
extern void closed_button_proc(item,event)
Panel_item item;
Event *event;
{ int client = (int) panel_get(item,PANEL_CLIENT_DATA);

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event))
                    panel_accept_preview(item,event);
                  break;
    case PANEL_EVENT_MOVE_IN: expl_mess(mess[client][1]);
                              break;
    case PANEL_EVENT_DRAG_IN: expl_mess(mess[client][1]);
```

```
                                break;
    case LOC_STILL: expl_mess(mess[client][1]);
                    break;
    case PANEL_EVENT_CANCEL: expl_mess("");
                             break;
    case LOC_WINEXIT: expl_mess("");
                      break;
  }
}


/*****************************************************************************
/* events for initial button                                                 *
/*****************************************************************************
extern void initial_button_proc(item,event)
Panel_item item;
Event *event;
{ static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                    panel_accept_preview(item,event);
                    panel_begin_preview(item,event);
                    }
                  break;
    case PANEL_EVENT_MOVE_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess("Display the initial curve.");
                                flag=1;
                                }
                              break;
    case PANEL_EVENT_DRAG_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess("Display the initial curve.");
                                flag=1;
                                }
                              break;
    case LOC_STILL: if (!flag) {
                      panel_begin_preview(item,event);
                      expl_mess("Display the initial curve.");
                      flag=1;
                      }
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
                               panel_cancel_preview(item,event);
                               expl_mess("");
                               flag=0;
                               }
                             break;
    case LOC_WINEXIT: if (flag) {
                        panel_cancel_preview(item,event);
                        expl_mess("");
                        flag=0;
                        }
                      break;
  }
```

```
}

/*************************************************************************
/* events for filtered button                                            *
/*************************************************************************
extern void filtered_button_proc(item,event)
Panel_item item;
Event *event;
{ static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                    panel_accept_preview(item,event);
                    panel_begin_preview(item,event);
                    }
                  break;
    case PANEL_EVENT_MOVE_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess("Display the filtered curve.");
                                flag=1;
                                }
                              break;
    case PANEL_EVENT_DRAG_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess("Display the filtered curve.");
                                flag=1;
                                }
                              break;
    case LOC_STILL: if (!flag) {
                      panel_begin_preview(item,event);
                      expl_mess("Display the filtered curve.");
                      flag=1;
                      }
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
                               panel_cancel_preview(item,event);
                               expl_mess("");
                               flag=0;
                               }
                             break;
    case LOC_WINEXIT: if (flag) {
                        panel_cancel_preview(item,event);
                        expl_mess("");
                        flag=0;
                        }
                      break;
  }
}

/*************************************************************************
/* events for semilog button                                             *
/*************************************************************************
extern void semilog_button_proc(item,event)
Panel_item item;
```

```
Event *event;
{ static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                      panel_accept_preview(item,event);
                      panel_begin_preview(item,event);
                      }
                  break;
    case MS_RIGHT: if (event_is_down(event)) {
                      panel_accept_preview(item,event);
                      panel_begin_preview(item,event);
                      }
                  break;
    case PANEL_EVENT_MOVE_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess(
                       "Display the semilog curve. L: Alone. R: With patterns.");
                                flag=1;
                                }
                              break;
    case PANEL_EVENT_DRAG_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess(
                       "Display the semilog curve. L: Alone. R: With patterns.");
                                flag=1;
                                }
                              break;
    case LOC_STILL: if (!flag) {
                      panel_begin_preview(item,event);
                      expl_mess(
                      "Display the semilog curve. L: Alone. R: With patterns.");
                      flag=1;
                      }
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
                               panel_cancel_preview(item,event);
                               expl_mess("");
                               flag=0;
                               }
                             break;
    case LOC_WINEXIT: if (flag) {
                        panel_cancel_preview(item,event);
                        expl_mess("");
                        flag=0;
                        }
                      break;
  }
}

/*************************************************************************
/* events for loglog button                                              *
/*************************************************************************
extern void loglog_button_proc(item,event)
Panel_item item;
```

```
Event *event;
{ static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                     panel_accept_preview(item,event);
                     panel_begin_preview(item,event);
                     }
                  break;
    case MS_RIGHT: if (event_is_down(event)) {
                     panel_accept_preview(item,event);
                     panel_begin_preview(item,event);
                     }
                  break;
    case PANEL_EVENT_MOVE_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess(
                     "Display the loglog curve. L: Alone. R: With patterns.");
                                flag=1;
                                }
                              break;
    case PANEL_EVENT_DRAG_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess(
                     "Display the loglog curve. L: Alone. R: With patterns.");
                                flag=1;
                                }
                              break;
    case LOC_STILL: if (!flag) {
                      panel_begin_preview(item,event);
                      expl_mess(
                      "Display the loglog curve. L: Alone. R: With patterns.");
                      flag=1;
                      }
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
                               panel_cancel_preview(item,event);
                               expl_mess("");
                               flag=0;
                               }
                             break;
    case LOC_WINEXIT: if (flag) {
                        panel_cancel_preview(item,event);
                        expl_mess("");
                        flag=0;
                        }
                      break;
  }
}

/*************************************************************************
/* events for derivative button                                          *
/*************************************************************************
extern void derivative_button_proc(item,event)
Panel_item item;
```

```
Event *event;
{ static int flag;

  switch (event_id(event)) {
    case MS_LEFT: if (event_is_up(event)) {
                     panel_accept_preview(item,event);
                     panel_begin_preview(item,event);
                     }
                  break;
    case MS_RIGHT: if (event_is_down(event)) {
                     panel_accept_preview(item,event);
                     panel_begin_preview(item,event);
                     }
                  break;
    case PANEL_EVENT_MOVE_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess(
                    "Display the derivative curve. L: Alone. R: With patterns.")
                                flag=1;
                                }
                              break;
    case PANEL_EVENT_DRAG_IN: if (!flag) {
                                panel_begin_preview(item,event);
                                expl_mess(
                    "Display the derivative curve. L: Alone. R: With patterns.")
                                flag=1;
                                }
                              break;
    case LOC_STILL: if (!flag) {
                      panel_begin_preview(item,event);
                      expl_mess(
                    "Display the derivative curve. L: Alone. R: With patterns.")
                      flag=1;
                      }
                    break;
    case PANEL_EVENT_CANCEL: if (flag) {
                               panel_cancel_preview(item,event);
                               expl_mess("");
                               flag=0;
                               }
                             break;
    case LOC_WINEXIT: if (flag) {
                        panel_cancel_preview(item,event);
                        expl_mess("");
                        flag=0;
                        }
                      break;
  }
}
```

```
/*****************************************************************************
/* "win_init.c" : initializations for windows.                               *
/*****************************************************************************
#define WIN_MAIN

#include "win.h"
#include "env.h"
#include <stdio.h>
#include <string.h>


/*****************************************************************************
/* references                                                                *
/*****************************************************************************


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void init_windows();
  static int read_wells_list();
  static void create_images();
  static void create_messages();
  static void open_fonts();


/*****************************************************************************
/* initializations for windows                                               *
/*****************************************************************************
extern void init_windows()
{ nbr_wells=read_wells_list(WELLS_LIST);

  create_images();

  create_messages();

  open_fonts();
}


/*****************************************************************************
/* read the list of wells                                                    *
/*****************************************************************************
static int read_wells_list(name)
char *name;
{ int i;
  FILE *in_file;

  if (!(in_file = fopen(name,"r"))) return 0;

  for (i=0;!feof(in_file);i++)
    fscanf(in_file,"%s*\n",w_name[i]);
  fclose(in_file);
  return i-1;
}
```

```
/*****************************************************************************
/* create the images for panels                                              *
/*****************************************************************************
static void create_images()
{ int i,j,width,length;
  struct pixrect *background;
  Frame frame;
  Panel panel;

  frame = window_create(NULL,FRAME,FRAME_NO_CONFIRM,TRUE,0);
  panel = window_create(frame,PANEL,0);

  opened_square = mem_create(DIM_SQ,DIM_SQ,1);
  pr_vector(opened_square,0,0,DIM_SQ-1,0,PIX_SRC,1);
  pr_vector(opened_square,0,0,0,DIM_SQ-1,PIX_SRC,1);
  pr_vector(opened_square,0,DIM_SQ-1,DIM_SQ-1,DIM_SQ-1,PIX_SRC,1);
  pr_vector(opened_square,DIM_SQ-1,0,DIM_SQ-1,DIM_SQ-1,PIX_SRC,1);

  closed_square = mem_create(DIM_SQ,DIM_SQ,1);
  pr_rop(closed_square,0,0,DIM_SQ,DIM_SQ,PIX_NOT(PIX_SRC),opened_square,0,0);

  for (i=0;i<nbr_wells;i++)
    image[i][0] = panel_button_image(panel,w_name[i],MAX_CHAR_NAME,0);

  width = image[0][0]->pr_size.x;
  length = image[0][0]->pr_size.y;

  background = mem_create(width,length,1);
  for (i=0;i<width;i++)
    for (j=0;j<length;j++)
      pr_put(background,i,j,(i+j)%2);

  for (i=0;i<nbr_wells;i++) {
    image[i][1] = mem_create(width,length,1);
    pr_rop(image[i][1],0,0,width,length,PIX_SRC,background,0,0);
    pr_rop(image[i][1],0,0,width,length,PIX_SRC|PIX_DST,image[i][0],0,0);
    }
  window_destroy(frame);
}


/*****************************************************************************
/* create the messages                                                       *
/*****************************************************************************
static void create_messages()
{ int i,j;

  for (i=0;i<nbr_wells;i++) {
    strcpy(mess[i][0],"Start analysis for well ");
    strcpy(mess[i][1],"Show window for well ");
    strcpy(mess[i][2],"Close window for well ");

    for (j=0;j<3;j++) {
      strcat(mess[i][j],w_name[i]);
      strcat(mess[i][j],".");
```

```
      }
  }
}


/*****************************************************************************
/* open the fonts                                                            *
/*****************************************************************************
static void open_fonts()
{ scale_font=pf_open("/usr/lib/fonts/fixedwidthfonts/screen.r.7");
}
```

```
/*****************************************************************************
/* "win_main.c" : call the different procedures used for windows             *
/*****************************************************************************
#include <stdio.h>

/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern void init_windows();
extern void create_base_frame();
extern void create_expl_bar();
extern void create_well_frames();
extern void create_closer();
extern void create_confirmer();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void create_windows();


/*****************************************************************************
/* create windows                                                            *
/*****************************************************************************
extern void create_windows()
{
  init_windows();
  create_base_frame();
  create_expl_bar();
  create_well_frames();
  create_closer();
  create_confirmer();
}
```

```
/*****************************************************************************
/* "win_util.c" : utilities for windows.                                     *
/*****************************************************************************
#include "win.h"


/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern void call_closer();

extern void opened_button_proc();
extern void closed_button_proc();

extern void draw_initial_curve();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void bind_window_with_well();
  static int find_next_frame();
  static void set_opened_square();
    static void close_well_from_square();

extern void free_window();
  static void set_closed_square();
    static void open_well_from_square();

extern void show_window();

extern  void destroy_all_windows();


/*****************************************************************************
/* link one well to one window                                               *
/*****************************************************************************
extern void bind_window_with_well(n)
int n;
{ int i;

  while (!(i=next_frame_free()))
    call_closer();

  w_window[n]=i;
  window_set(w_frame[i],FRAME_LABEL,w_name[n],
                        WIN_CLIENT_DATA, n,
                        0);
  set_opened_square(n);
  draw_initial_curve(n);
}


/*****************************************************************************
/* find the next free frame                                                  *
/*****************************************************************************
```

```
static int next_frame_free()
{ int i;

  for (i=1;i<MAX_NBR_WINDOWS;i++)
    if (!bound_frame[i]) {
      bound_frame[i]=1;
      return i;
      }

  return 0;
}


/*************************************************************************
/* set the square to "opened"                                            *
/*************************************************************************
static void set_opened_square(n)
int n;
{

  panel_set(square[n],PANEL_LABEL_IMAGE,opened_square,
                      PANEL_EVENT_PROC, opened_button_proc,
                      PANEL_NOTIFY_PROC,close_well_from_square,
                      0);
}


/*************************************************************************
/* close a well from the corresponding square item                       *
/*************************************************************************
static void close_well_from_square(item,event)
Panel_item item;
Event *event;
{ int w_number=(int) panel_get(item,PANEL_CLIENT_DATA);

  free_window(w_window[w_number]);
}


/*************************************************************************
/* free one window                                                       *
/*************************************************************************
extern void free_window(f_number)
int f_number;
{ int w_number=(int) window_get(w_frame[f_number],WIN_CLIENT_DATA);

  bound_frame[f_number]=0;
  w_window[w_number]=0;
  set_closed_square(w_number);
  window_set(w_frame[f_number],WIN_SHOW,FALSE,0);
}


/*************************************************************************
/* set the square to "closed"                                            *
/*************************************************************************
```

```
static void set_closed_square(n)
int n;
{

  panel_set(square[n],PANEL_LABEL_IMAGE,closed_square,
                      PANEL_EVENT_PROC, closed_button_proc,
                      PANEL_NOTIFY_PROC,open_well_from_square,
                      0);
}


/*************************************************************************
/* open a well from the corresponding square item                        *
/*************************************************************************
static void open_well_from_square(item,event)
Panel_item item;
Event *event;
{ int w_number=(int) panel_get(item,PANEL_CLIENT_DATA);

  bind_window_with_well(w_number);
  show_window(w_number);
}


/*************************************************************************
/* show the window for the well n                                        *
/*************************************************************************
extern void show_window(n)
int n;
{ window_set(w_frame[(w_window[n])],WIN_SHOW,TRUE,0);
}


/*************************************************************************
/* destroy all the windows before quitting                               *
/*************************************************************************
extern void destroy_all_windows()
{

  window_set(closer,FRAME_NO_CONFIRM,TRUE,0);
  window_set(confirmer,FRAME_NO_CONFIRM,TRUE,0);
  window_set(base_frame,FRAME_NO_CONFIRM,TRUE,0);

  window_destroy(closer);
  window_destroy(confirmer);
  window_destroy(base_frame);
}
```

```
/*****************************************************************************
/* "win_wells.c" : definition of the wells windows. Date : 05/26/88          *
/*****************************************************************************
#define GPH_MAIN

#include "win.h"
#include "gph.h"


/*****************************************************************************
/* references                                                                *
/*****************************************************************************
extern void free_window();

extern void draw_initial_curve();
extern void draw_filtered_curve();
extern void draw_semilog_curve();
extern void draw_loglog_curve();
extern void draw_derivative_curve();
extern void draw_semilog_patterns();
extern void draw_loglog_patterns();
extern void draw_derivative_patterns();

extern void initial_button_proc();
extern void filtered_button_proc();
extern void semilog_button_proc();
extern void loglog_button_proc();
extern void derivative_button_proc();


/*****************************************************************************
/* contents                                                                  *
/*****************************************************************************
extern void create_well_frames();

  static Menu new_sub_menu();
    static void close_subframe_from_menu(); /* procedure for the Notifier */

  static void create_curves_panel();
    static void display_initial_curve();    /* procedure for the Notifier */
    static void display_filtered_curve();   /* procedure for the Notifier */
    static void display_semilog_curve();    /* procedure for the Notifier */
    static void display_loglog_curve();     /* procedure for the Notifier */
    static void display_derivative_curve(); /* procedure for the Notifier */

  static void create_canvas();


/*****************************************************************************
/* create the windows for the wells                                          *
/*****************************************************************************
extern void create_well_frames()
{ int i;
  static int x_pos[] = {0,330,480,180,630};
  static int y_pos[] = {0,270,240,300,210};
```

```
  for (i=1;i<MAX_NBR_WINDOWS;i++) {
    w_frame[i] = window_create(base_frame,FRAME,
                                    FRAME_SHOW_LABEL, TRUE,
                                    FRAME_NO_CONFIRM, TRUE,
                                    WIN_X, x_pos[i],
                                    WIN_Y, y_pos[i],
                                    WIN_HEIGHT, VERTICAL_SIZE+100,
                                    WIN_WIDTH, HORIZONTAL_SIZE,
                                    0);

    window_set(w_frame[i],WIN_MENU,new_sub_menu(i),0);

    create_curves_panel(i);

    create_canvas(i);

    window_fit_height(w_frame[i]);
    }
}


/************************************************************************
/* set the menu for the subframe                                        *
/************************************************************************
static Menu new_sub_menu(f_number)
int f_number;
{ Menu menu;

  menu = (Menu) window_get(w_frame[f_number],WIN_MENU);

  menu_set(menu, MENU_REMOVE, 1,
                 MENU_REMOVE, 2,
                 MENU_INSERT, 0, menu_create_item(
                                    MENU_STRING, "Close",
                                    MENU_ACTION_PROC, close_subframe_from_menu,
                                    0),
                 MENU_CLIENT_DATA, f_number,
                 0);

  return(menu);
}


/************************************************************************
/* close a frame from menu                                              *
/************************************************************************
static void close_subframe_from_menu(m,mi)
Menu m;
Menu_item mi;
{ free_window((int) menu_get(m,MENU_CLIENT_DATA));
}


/************************************************************************
/* create the panel for displaying curves                               *
/************************************************************************
```

```
static void create_curves_panel(f_number)
int f_number;
{ Panel panel;

  panel = window_create(w_frame[f_number],PANEL,
                    WIN_CONSUME_PICK_EVENTS, WIN_NO_EVENTS,
                                             LOC_MOVE,
                                             LOC_DRAG,
                                             LOC_STILL,
                                             LOC_WINENTER,
                                             LOC_WINEXIT,
                                             LOC_RGNENTER,
                                             LOC_RGNEXIT,
                                             WIN_MOUSE_BUTTONS,
                                             0,
                    0);

  (void) panel_create_item(panel,PANEL_BUTTON,
                PANEL_LABEL_IMAGE, panel_button_image(panel,"Initial",0,0),
                PANEL_EVENT_PROC, initial_button_proc,
                PANEL_NOTIFY_PROC, display_initial_curve,
                0);
  (void) panel_create_item(panel,PANEL_BUTTON,
                PANEL_LABEL_IMAGE, panel_button_image(panel,"Filtered",0,0),
                PANEL_EVENT_PROC, filtered_button_proc,
                PANEL_NOTIFY_PROC, display_filtered_curve,
                0);
  (void) panel_create_item(panel,PANEL_BUTTON,
                PANEL_LABEL_IMAGE, panel_button_image(panel,"Semilog",0,0),
                PANEL_EVENT_PROC, semilog_button_proc,
                PANEL_NOTIFY_PROC, display_semilog_curve,
                0);
  (void) panel_create_item(panel,PANEL_BUTTON,
                PANEL_LABEL_IMAGE, panel_button_image(panel,"Loglog",0,0),
                PANEL_EVENT_PROC, loglog_button_proc,
                PANEL_NOTIFY_PROC, display_loglog_curve,
                0);
  (void) panel_create_item(panel,PANEL_BUTTON,
                PANEL_LABEL_IMAGE, panel_button_image(panel,"Derivative",0,0),
                PANEL_EVENT_PROC, derivative_button_proc,
                PANEL_NOTIFY_PROC, display_derivative_curve,
                0);

   window_fit_height(panel);
}


/*************************************************************************
/* display initial curve                                                 *
/*************************************************************************
static void display_initial_curve(item,event)
Panel_item item;
Event *event;
{ int client = (int) window_get((Frame) window_get(
                                  (Panel) panel_get(item,PANEL_PARENT_PANEL),
                                  WIN_OWNER),
```

```
                                    WIN_CLIENT_DATA);
  draw_initial_curve(client);
}


/*************************************************************************
/* display filtered curve                                                *
/*************************************************************************
static void display_filtered_curve(item,event)
Panel_item item;
Event *event;
{ int client = (int) window_get((Frame) window_get(
                                    (Panel) panel_get(item,PANEL_PARENT_PANEL),
                                    WIN_OWNER),
                                WIN_CLIENT_DATA);
  draw_filtered_curve(client);
}


/*************************************************************************
/* display semilog curve                                                 *
/*************************************************************************
static void display_semilog_curve(item,event)
Panel_item item;
Event *event;
{ int client = (int) window_get((Frame) window_get(
                                    (Panel) panel_get(item,PANEL_PARENT_PANEL),
                                    WIN_OWNER),
                                WIN_CLIENT_DATA);
  switch(event_id(event)) {
    case MS_LEFT: draw_semilog_curve(client);
                  break;
    case MS_RIGHT: draw_semilog_curve(client);
                   draw_semilog_patterns(client);
                   break;
  }
}


/*************************************************************************
/* display loglog curve                                                  *
/*************************************************************************
static void display_loglog_curve(item,event)
Panel_item item;
Event *event;
{ int client = (int) window_get((Frame) window_get(
                                    (Panel) panel_get(item,PANEL_PARENT_PANEL),
                                    WIN_OWNER),
                                WIN_CLIENT_DATA);
  switch(event_id(event)) {
    case MS_LEFT: draw_loglog_curve(client);
                  break;
    case MS_RIGHT: draw_loglog_curve(client);
                   draw_loglog_patterns(client);
                   break;
  }
```

```
}


/*****************************************************************************
/* display derivative curve                                                  *
/*****************************************************************************
static void display_derivative_curve(item,event)
Panel_item item;
Event *event;
{ int client = (int) window_get((Frame) window_get(
                                    (Panel) panel_get(item,PANEL_PARENT_PANEL),
                                    WIN_OWNER),
                                WIN_CLIENT_DATA);
  switch(event_id(event)) {
    case MS_LEFT: draw_derivative_curve(client);
                  break;
    case MS_RIGHT: draw_derivative_curve(client);
                   draw_derivative_patterns(client);
                   break;
  }
}


/*****************************************************************************
/* create the graphic window                                                 *
/*****************************************************************************
static void create_canvas(f_number)
int f_number;
{ Canvas canvas;

  canvas = window_create(w_frame[f_number],CANVAS,
                         WIN_HEIGHT, VERTICAL_SIZE,
                         0);
  pw[f_number] = canvas_pixwin(canvas);
}
```

```
;;; -*- mode: ART; Package: ART-USER; Base:10. -*-
;;; file: well/input.art

;;; This file contains the initial facts and schemata.

;;; The program will use a one level viewpoint structure. This level,
;;; called "hypothetical" will contain all hypothesis regarding the
;;; nature of the medium.

(def-viewpoint-levels hypothetical)

;;; Initial facts and relations.

(defrelation start-up (?init)
        "flag for opening window for user interface")
(defrelation abs-d-error (?error)
        "absolute imprecision allowed on the derivatives")
(defrelation significant-length (?sig-length)
        "length in fraction of a log cycle")
(defrelation open-stream (?stream)
        "stream between ART and C")
(defrelation read-data (?data)
        "raw data from stream")

(deffacts initial
        (start-up yes)
        )

(deffacts parameters "used for curve analysis"
        (abs-d-error 0.15)
        (significant-length 0.25)
        )

(defschema derivative
        (instance-of relation)
        (inverse derivative-of))

(defschema semilog
        (instance-of relation)
        (inverse semilog-of))

(defschema loglog
        (instance-of relation)
        (inverse loglog-of))

(defschema model
        (instance-of relation)
        (inverse model-of))

(defschema straight-line
        (instance-of slot)
        (slot-how-many multiple-values))

(defschema hump
        (instance-of slot)
        (slot-how-many multiple-values))
```

```
(defschema well
        (derivative)
        (semilog)
        (loglog)
        (model)
        (last-time)
        (number)
        )

(defschema well-semilog
        (semilog-of)
        (straight-line)
        )

(defschema well-loglog
        (loglog-of)
        (log-derivative)
        (straight-line)
        )

(defschema well-derivative
        (derivative-of)
        (time)
        (p-d-derivative)
        (max-p-derivative)
        (local-max-time)
        (straight-line)
        (hump)
        )

(defschema well-model
        (model-of)
        (reservoir)
        (reservoir-exp ())
        (early)
        (early-exp ())
        (intermediate)
        (late)
        (wellbore-storage)
        (wellbore-exp ())
        (fractured)
        (fractured-exp ())
        (boundary)
        (boundary-exp ())
        )

(defrule start-C-program
  ?x <- (start-up yes)
=>
        (retract ?x)
        (setq c-stream #L(run-unix-program "~antoine/Wes/Exec/c_wes"
                                                :input :stream
                                                :output :stream
                                                :wait nil))
        (assert (open-stream =c-stream)))
```

```
(defrule read-stream
        (declare (salience -1000))
  ?x <- (open-stream ?stream)
=>
        (retract ?x)
        (if (listen ?stream) then
          (assert (read-data =(seq*$ (read ?stream)))))
        (assert (open-stream ?stream)))


(defrule open-well
  ?x <- (read-data (1 ?well-number ?well))
=>
        (retract ?x)
        (assert (schema ?well
                        (instance-of well)
                        (semilog =(concat ?well 'semilog))
                        (loglog =(concat ?well 'loglog))
                        (derivative =(concat ?well 'derivative))
                        (model =(concat ?well 'model))
                        (number ?well-number))))

(defrule initial-data
  ?x <- (read-data (2 ?w-n $?data))
        (number ?well ?w-n)
        (loglog ?well ?well-l)
        (derivative ?well ?well-d)
=>
        (retract ?x)
        (assert (schema ?well
                        (last-time =(nth$ ?data 1))))
        (assert (schema ?well-l
                        (log-derivative =(nth$ ?data 2))))
        (assert (schema ?well-d
                        (time =(nth$ ?data 3))
                        (p-d-derivative =(nth$ ?data 4))
                        (max-p-derivative =(nth$ ?data 5)))))

(defrule semilog-straight-lines
  ?x <- (read-data (3 ?w-n ?n $?data))
        (number ?well ?w-n)
        (semilog ?well ?well-s)
=>
        (retract ?x)
        (for i from 1 to ?n do
           (assert (schema ?well-s
                           (straight-line =(nth$ ?data i))))))

(defrule loglog-straight-lines
  ?x <- (read-data (4 ?w-n ?n $?data))
        (number ?well ?w-n)
        (loglog ?well ?well-l)
=>
        (retract ?x)
        (for i from 1 to ?n do
```

```
            (assert (schema ?well-1
                            (straight-line =(nth$ ?data i))))))

(defrule derivative-straight-lines
  ?x <- (read-data (5 ?w-n ?n $?data))
        (number ?well ?w-n)
        (derivative ?well ?well-d)
=>
        (retract ?x)
        (for i from 1 to ?n do
           (assert (schema ?well-d
                           (straight-line =(nth$ ?data i))))))

(defrule humps
  ?x <- (read-data (6 ?w-n ?n $?data))
        (number ?well ?w-n)
        (derivative ?well ?well-d)
=>
        (retract ?x)
        (for i from 1 to ?n do
           (assert (schema ?well-d
                           (hump =(nth$ ?data i))))))

(defrule local-max-time
  ?x <- (read-data (7 ?w-n ?l-m-time))
        (number ?well ?w-n)
        (derivative ?well ?well-d)
=>
        (retract ?x)
        (assert (schema ?well-d
                        (local-max-time ?l-m-time))))

(defrule quit
  ?x <- (read-data (8))
  ?y <- (open-stream ?stream)
=>
        (retract ?x ?y)
        (close ?stream))
```

```
;;; -*- Mode: ART; Package: ART-USER; Base:10. -*-
;;; file: well/model.art

;;; Rules used to extract a model by generating hypotheses and checking
;;; those hypotheses on other plots (semilog or loglog).


;;; Locates the end of early-time period by a global or local maximum

(defrule early-data-global-max
        (max-p-derivative ?well-d (?tm ?max-p-d))
        (test (< ?tm 1000))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (early ?well-m ?tm)))
        )

(defrule early-data-local-max
        (max-p-derivative ?well-d (?tm&:(?tm >= 1000) ?))
        (local-max-time ?well-d ($? ?l-m-t $?))
        (not (hump ?well-d (hill ? ? (?th&:(?l-m-t = ?th) ?))))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (early ?well-m ?l-m-t)))
        )

;;; If there is a hill hump, whose maximum is different from the
;;; max-p-derivative and located before it, then, this maximum
;;; corresponds to the end of the early time period

(defrule early-data-hump
        (max-p-derivative ?well-d (?tm ?))
        (hump ?well-d (hill ? ? (?th&:(?th < ?tm) ?)))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (early ?well-m ?th)))
        )

;;; If there is a negative slope at the beginning, it means that there is
;;; no early data, and that the intermediate-time period starts there.

(defrule no-early-data
        (p-d-derivative ?well-d (?fpdd&:(?fpdd < 0) $?))
        (time ?well-d (?first-t $?))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (early ?well-m =?first-t)))
        )

;;; If there is a slope 1 at the beginning of the derivative curve,
;;; then there is wellbore storage.
```

```
(defrule wellbore-storage-on-derivative
        (p-d-derivative ?well-d (?p-d1 ?p-d2 $?))
        (abs-d-error ?error)
        (test (< (abs (1- (/ (+ ?p-d1 ?p-d2) 2))) ?error))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (wellbore-storage ?well-m yes)))
        )

;;; If the first two slopes at the beginning of the loglog curve are close
;;; to 1, then there is wellbore storage.

(defrule wellbore-storage-on-loglog
        (loglog-of ?well-l ?well)
        (log-derivative ?well-l (?sl1 ?sl2 $?))
        (model ?well ?well-m)
        (abs-d-error ?error)
        (test (< (abs (1- (/ (+ ?sl1 ?sl2) 2))) ?error))
=>
        (hypothesize (assert (wellbore-storage ?well-m yes)))
        )


;;; If there is a slope 1/2 at the beginning, then: fractured

(defrule fractured-on-derivative
        (p-d-derivative ?well-d (?p-d1 ?p-d2 $?))
        (abs-d-error ?error)
        (test (< (abs (- (/ (+ ?p-d1 ?p-d2) 2) 0.5)) ?error))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (fractured ?well-m yes)))
        )

;;; A 1/2 slope on loglog means fractured system

(defrule fractured-on-loglog
        (loglog-of ?well-l ?well)
        (log-derivative ?well-l (?sl1 ?sl2 $?))
        (model ?well ?well-m)
        (abs-d-error ?error)
        (test (< (abs (- (/ (+ ?sl1 ?sl2) 2) 0.5)) ?error))
=>
        (hypothesize (assert (fractured ?well-m yes)))
        )

;;; Boundary conditions: If the slope at the end is negative, hypothesize
;;; there is a pressure maintenance boundary

(defrule pressure-maintenance-boundary
        (p-d-derivative ?well-d ($? ?pdd1 ?pdd2))
        (test (and (< ?pdd1 0)
                   (< ?pdd2 ?pdd1)))
        (derivative-of ?well-d ?well)
```

```
        (model ?well ?well-m)
=>
        (hypothesize (assert (boundary ?well-m pressure-maintenance)))
        )

;;; If the slope at the end is positive, hypothesize there is a closed
;;; system (no-flow boundary)

(defrule no-flow-boundary
        (p-d-derivative ?well-d ($? ?pdd1 ?pdd2))
        (test (and (> ?pdd1 0)
                   (> ?pdd2 ?pdd1)))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (boundary ?well-m no-flow)))
        )

;;; If, after early time, there is a doubling of the slope of the semilog
;;; curve, it means that there is a no-flow boundary

(defrule no-flow-on-semilog
        (early ?well-m ?e-time)
        (model-of ?well-m ?well)
        (semilog ?well ?well-s)
        (abs-d-error ?error)
        (straight-line ?well-s ((?it1&:(?it1 > ?e-time) ?) (?ft1 ?) ?sl1))
        (straight-line ?well-s ((?it2 ?) ? ?sl2))
        (test (and (> ?it2 ?ft1)
                   (< (abs (- ?sl2 (* 2 ?sl1))) ?error)))
=>
        (hypothesize (assert (boundary ?well-m no-flow)))
        )

;;; If there is a horizontal straight line at the end there it is an
;;; infinite system

(defrule infinite-system
        (derivative-of ?well-d ?well)
        (last-time ?well ?last-t)
        (abs-d-error ?error)
        (significant-length ?sig-l)
        (straight-line ?well-d (? (?lt ?) ?slope))
        (test (and (< (abs ?slope) ?error)
                   (< (- (log ?last-t 10) (log ?lt 10)) ?sig-l)))
        (model ?well ?well-m)
=>
        (hypothesize (assert (boundary ?well-m infinite)))
        )

;;; If there is a valley hump, followed by either a hill hump or a
;;; horizontal straight-line then hypothesize double porosity reservoir

(defrule double-porosity
        (abs-d-error ?error)
        (hump ?well-d (valley (?it1 ?) ? ?))
```

```
        (or (hump ?well-d (hill (?it2&:(?it2 > ?it1) ?) ? ?))
            (straight-line ?well-d ((?it3&:(?it3 > ?it1) ?)
                                    ?
                                    ?slope&:((abs ?slope) < ?error))))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
=>
        (hypothesize (assert (reservoir ?well-m double-porosity)))
        )

;;; If there is a valley hump close to the end of the data set, then
;;; hypothezise that it is a homogeneous reservoir with a no flow boundary.

(defrule homogeneous-and-closed-system
        (significant-length ?sig-l)
        (derivative-of ?well-d ?well)
        (last-time ?well ?last-t)
        (hump ?well-d (valley ? (?lt ?) ?))
        (test (< (- (log ?last-t 10) (log ?lt 10)) ?sig-l))
        (model ?well ?well-m)
=>
        (hypothesize (assert (reservoir ?well-m homogeneous)))
        (hypothesize (assert (boundary ?well-m no-flow)))
        )

;;; If there is a hill hump followed by a straight line until the end
;;; of the data set, then hypothesize that it is a homogeneous reservoir
;;; with an infinite boundary

(defrule homogeneous-and-infinite-system
        (significant-length ?sig-l)
        (derivative-of ?well-d ?well)
        (last-time ?well ?last-t)
        (abs-d-error ?error)
        (hump ?well-d (hill ? (?hump-lt ?) ?))
        (straight-line ?well-d ((?line-it ?) (?line-lt ?) ?slope))
        (test (and (< ?hump-lt ?line-it)
                   (< (- (log ?line-it 10) (log ?hump-lt 10)) 0.25)
                   (< (abs ?slope) ?error)
                   (< (- (log ?last-t 10) (log ?line-lt 10)) ?sig-l)))
        (model ?well ?well-m)
=>
        (hypothesize (assert (reservoir ?well-m homogeneous)))
        (hypothesize (assert (boundary ?well-m infinite)))
        )

;;; If there is a hill hump and no valley hump, then hypothesize
;;; homogeneous

(defrule homogeneous
        (hump ?well-d (hill ? ? (?th ?)))
        (derivative-of ?well-d ?well)
        (model ?well ?well-m)
        (early ?well-m ?th)
        (not (hump ?well-d (valley ? ? ?)))
=>
```

```
        (hypothesize (assert (reservoir ?well-m homogeneous)))
        )

;;; These rules merge viewpoints. They aggregate all possibilities for every
;;; possible combination of facts in the well-model schemata. It also prints
;;; all possible models for each well

(defrule find-model-with-wellbore-storage
        (early ?well-m ?)
        (wellbore-storage ?well-m yes)
        (reservoir ?well-m ?type)
        (boundary ?well-m ?bound)
        (model-of ?well-m ?well)
=>
        (fresh-line)
        (write-string "A possible model for well ")
        (write-string (string ?well))
        (write-string " is a ")
        (write-string (string ?type))
        (write-string " reservoir with wellbore storage and a ")
        (write-string (string ?bound))
        (write-string   " boundary.")
        )

(defrule find-fractured-model
        (early ?well-m ?)
        (fractured ?well-m yes)
        (reservoir ?well-m ?type)
        (boundary ?well-m ?bound)
        (model-of ?well-m ?well)
=>
        (fresh-line)
        (write-string "A possible model for well ")
        (write-string (string ?well))
        (write-string " is a ")
        (write-string (string ?type))
        (write-string " reservoir with a fractured well and a ")
        (write-string (string ?bound))
        (write-string   " boundary.")
        )

(defrule find-model
        (early ?well-m ?)
        (reservoir ?well-m ?type)
        (boundary ?well-m ?bound)
        (not (or (wellbore-storage ?well-m yes)
                 (fractured ?well-m yes)))
        (model-of ?well-m ?well)
=>
        (fresh-line)
        (write-string "A possible model for well ")
        (write-string (string ?well))
        (write-string " is a ")
        (write-string (string ?type))
        (write-string " reservoir with a ")
        (write-string (string ?bound))
```

```
        (write-string   " boundary.")
        )
```

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
1 CYCLOTRON ROAD
BERKELEY, CALIFORNIA 94720