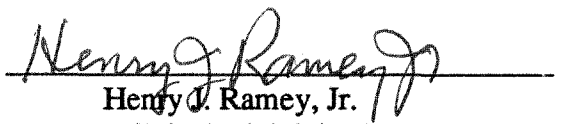
MICROCOMPUTER SIMULATION OF THE TRANSIENT FLOW OF REAL GAS THROUGH POROUS MEDIA

A THESIS
SUBMITTED TO THE DEPARTMENT OF PETROLEUM ENGINEERING
AND COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
ENGINEER

By
Francois Joseph Groff
June 1992

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as partial fulfillment for the degree of Engineer.



Henry J. Ramey, Jr.
(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as partial fulfillment for the degree of Engineer.

Roland N. Horne

Approved for the University Committee on Graduate Studies:

Dedicated to

My mother Alice
My father Robert
My friends Genevieve, Norman and Adegboyega.

In some way, you all made this work possible.

Acknowledgments

I would like to express my gratitude to Dr. Henry Ramey, Jr. for his suggestions and guidance during this study. His profound kindness has made this work possible. I also thank Dr. Roland Horne for serving on the committee for this thesis. I finally extend my gratitude to Jean Cook from the Stanford Geothermal Program, and to the Department of Petroleum Engineering at Stanford for their support and encouragement.

This work was supported by the Stanford Geothermal Program (SGP) under Department of Energy contract No. DE - F607 - 90ID12934 and by the Schlumberger Corporation through the Schlumberger Fellowship for Earth Sciences at Stanford. This support is gratefully acknowledged.

Abstract

An interactive program that lets users study the flow of real gas through porous media under a graphic environment has been presented. The program treats high velocity flow accurately so that it is not mistaken for formation damage. A rigorous treatment of wellbore storage is also a major departure from traditional well test models. A mathematical model written by Fligelman in 1981 is the foundation of this project. The program presented here, with its graphic capabilities places results at the user's fingertips. This program can be used in research activities to perform sensitivity analysis. It can also be used as a working tool for well test design or well test interpretation. Finally, it can be used as an educational tool for all the tasks just mentioned.

Table of Contents

List of Tables

page

List of Figures

page

1. Introduction

This section presents the purpose of this study and previous studies. The initial objective of early studies was to investigate pressure-dependent wellbore storage for high drawdown gas wells. The project described in this report is primarily concerned with improving gas well test analysis by providing an interactive graphic-driven software usable by all engineers. The main goals are to investigate the effects of wellbore storage, skin, and high-velocity flow on real gas transient pressure response during drawdown or injection tests. Drawdown tests are performed on gas wells to determine the flow capacity, kh, of the formation, the condition of the wellbore and high-velocity flow parameters.

Eilerts (1964) and Eilerts et al. (1965) solved real gas flow for both linear and radial reservoirs with inclusion of real gas properties and turbulence. Smith (1961) observed that the skin effect for gas wells often appeared to depend upon flowrate. Tek et al. (1962) and Swift and Kiel (1962) presented a fundamental basis for this observation: high-velocity flow could cause a pseudo-skin effect in the damaged region near a gas well. The language used when describing the mechanism that consumes energy at more than a linear rate with velocity is not consistent in the literature. The term used in flow equations (generally known as β) also has been given various names. Ramey (1965) and Agarwal et al. (1970) defined high-velocity flow as non-Darcy flow which can be treated as a flow-rate dependent skin effect. Firoozabadi and Katz (1979) argued that the term "non-Darcy flow" was not specifically descriptive. The same was said for defining Darcy flow for low velocities and an inertia regime for higher velocities, since the inertia effects are always present. Firoozabadi and Katz proposed to adopt the term "high-velocity flow" to describe the condition where neglecting $\beta\rho v^2$ calculates less pressure drop than would occur, and "velocity coefficient" to describe β instead of "turbulent factor" or "inertial coefficient". Fligelman et al. (1989) adopted the term "non-Darcy flow effect" defined as a pseudo-skin Dq_{sc} that is not simply an additional term as are the other skin factor components. Although high-velocity flow was studied in this paper, we will refer to this effect as "turbulence" for convenience.

Wellbore storage, skin effects and high-velocity flow were considered in a numerical solution for transient gas well test analysis for one-dimensional flow by Fligelman (1981). This solution is the starting point for this project.

The effect of high-velocity flow near a wellbore can be detrimental to the producing capacity of a gas well. The calculation of formation conductivity for cases of high flow rates using conventional semilog graphing yields values which are lower than the true formation conductivity. Turbulence effects, if not properly identified, can be mistaken for formation damage and can lead to serious errors in the interpretation.

Using a constant wellbore storage model to analyze well test data exhibiting changing storage may lead to large errors in the estimation of the skin damage. In the case of a drawdown with storage, when plotted on a log-log graph of the real gas pseudo-pressure, versus time, a shifting of the conventional graph from one dimensionless wellbore storage constant to a greater value might occur. As these cases would involve a high pressure drop of several thousands psi, it was believed that changing wellbore storage was responsible for this behavior. This led to Fligelman's (1981) mathematical model wherein wellbore storage is treated rigorously. In a similar model developed by Wattenbarger (1967), wellbore storage was not treated accurately.

A modern, interactive tool for investigating these important effects was needed. The objective of this project was to provide a numerical simulation for the isothermal and horizontal flow of real gas from a well through a homogeneous and isotropic formation on a personal computer, under a graphic environment. It is assumed that the gas flow in the formation is properly represented by a Forchheimer (1901) type equation. The wellbore boundary condition is a specified surface flow rate. The exterior boundary condition is either constant pressure, no-flow across the boundary, or infinite acting.

A major concern was to provide others involved in research activities and students with a software package as easy to use as possible, in order to produce future results, and/or design well tests. The software can also be used in future studies of other systems, as in geothermal engineering, for instance. Specific goals for the program were to allow a user to compare synthetic well tests with each other, compare field data and synthetic data, create libraries of type curves from synthetic well tests, obtain graphic representation of synthetic data as soon as the simulation is over, and finally, have easy access to past data.

1.1 Previous Work

Van Everdingen and Hurst (1949) published analytical solutions to the diffusivity equation for radial systems. Many solutions were available as Laplace transform inversion integrals and could not be expressed in term of elementary functions. The first ideal gas flow solutions were published by Aronofsky and Jenkins (1952) and Jenkins and Aronofsky (1953). They introduced computer techniques to solve non-linear gas flow equations, followed by Bruce et al. (1953). Cornell and Katz (1953) graphically integrated the differential equation for ideal gases and included the effects of turbulence. Van Everdingen (1953) and Hurst (1953) introduced the skin effect, later generalized by Wattenbarger (1967) by including a damaged annular region adjacent to the well. Smith (1961) studied the effect of flow rate on drawdown testing and Swift and Kiel (1962) and Tek et al. (1962) explained Smith's results as due to transient turbulent flow. Ramey (1965) considered the combined effects of turbulence, wellbore storage and skin damage.

Matthews (1961) presented an early comprehensive paper on pressure buildup analysis, and later Matthews and Russell (1967) published an SPE monograph on well testing, concentrating mainly on liquid flow. Until 1960, most gas flow studies concerned ideal gas. Carter et al. (1962, 1963a, 1963b) and later Russell et al. (1966) solved real gas flow problems and presented approximate methods for correcting ideal gas solutions to include real gas properties.

An important step was taken by Al-Hussainy et al. (1966a, 1966b) who introduced the real gas pseudo-pressure concept. Following Wattenbarger (1967), Fligelman (1981) used this concept to investigate short term drawdown tests where the exterior boundary does not influence the flow behavior at the well. Fligelman's program solves a non-linear partial differential equation that resulted from the consideration of various parameters, simultaneously or not. This work considered a constant rate inner boundary for the infinite reservoir case. Couri (1987) used Fligelman's approach to verify classical correlations in the gas engineering literature that did not consider the effects of wellbore storage, skin, and turbulent flow simultaneously. Couri also considered only drawdown cases.

1.2 Turbulent Flow

Turbulent flow causes an extra pressure drop at the well which can be mistaken for well damage. Forchheimer (1901) added an additional term to Darcy's law to include the increased pressure drop. According to Hubbert (1956) and Houpert (1959), what is referred to as a "non-Darcy" component does not correspond to the classical ideas of turbulent flow, but is caused by convective accelerations of fluid particles in passing through the pore space. Most experiments indicate that true turbulence occurs at Reynolds number values at least one order of magnitude higher than the Reynolds number values at which deviation from Darcy's law for laminar flow is observed. All experiments have confirmed the general applicability of the Forchheimer type of flow equation. Turbulence factors for the Forchheimer expression were determined by Cornell and Katz (1953). Katz and Coats (1968) presented a correlation of the turbulence coefficient as a function of permeability for the consolidated sandstones, dolomites and limestones. Additional experiments by Firoozabadi and Katz (1979) showed differences in slopes between correlations for consolidated media and unconsolidated media.

1.3 Wellbore Storage

The effect of wellbore storage was considered by Van Everdingen and Hurst (1949) and termed the "annulus unloading" effect. It was further investigated by Ramey (1965, 1970). Agarwal et al. (1970) presented a storage and skin type curve. Others presented several other type curves thereafter. See Earlougher and Kersch (1974). Ramey and Agarwal (1972) presented an analytical solution for a step change in wellbore storage. Hegeman et al. (1991) presented an analytical model in Laplace space representing increasing or decreasing wellbore storage during well testing for a variety of well / reservoir models. They investigated exponential and error function time dependence to represent changing wellbore storage, and considered buildup tests to demonstrate utility of their results.

We consider details of Fligelman's (1981) model in the next section.

2. Mathematical Model

A schematic of a radial flow model is shown in Fig. 2.1:

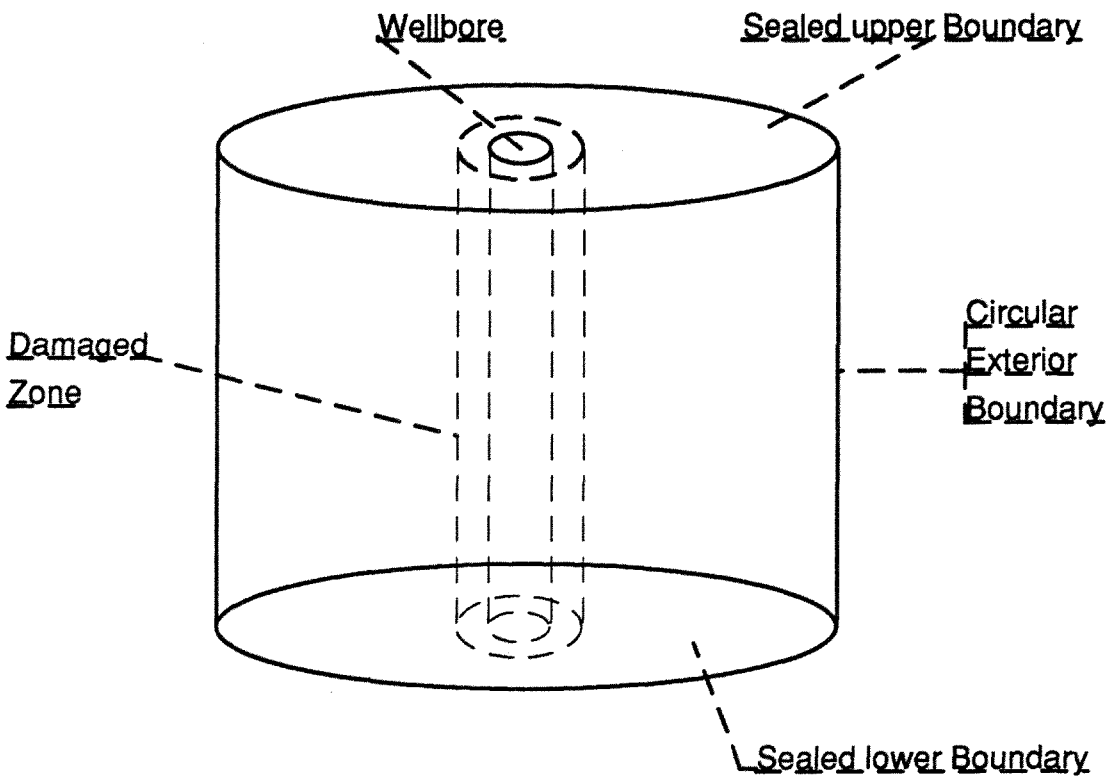


Figure 2.1. Radial Flow Model

The assumptions made by Fligelman (1981) were:

1. Horizontal radial flow, no gravity effects.
2. Isotropic rock properties.
3. Gas saturation at 100% of pore volume.
4. Isothermal flow.
5. Constant porosity value.
6. Newtonian fluid behavior.
7. No Klinkenberg effect.

2.1 Turbulence Effect

High-velocity, non-Darcy effects were represented by the Forchheimer equation transformed into a modified Darcy's law form by Swift and Kiel (1962).

$$u_r = -\delta_r \frac{k}{\mu(p)} \frac{\partial p}{\partial r} \tag{2.1}$$

δ_r is a radial Darcy's law correction factor that is equal to unity for laminar flow, and less than unity for turbulent flow.

$$\delta_r = \frac{1}{1 + \frac{\beta \rho(p) u_r k}{\mu(p)}} \tag{2.2}$$

β is the Katz, et al., (1959, 1968) turbulence parameter:

$$\beta = \frac{e^{23.83}}{k^{1.201}} \tag{2.3}$$

In the computer program, the velocity u_r for the Forchheimer equation is calculated through an iterative method:

$$u_r = \frac{-\frac{\mu(p)}{k} + \left[\left[\frac{\mu(p)}{k}\right]^2 - 4\beta \frac{M}{RT} \frac{\mu(p)}{k} \frac{\partial m(p)}{\partial r}\right]^{\frac{1}{2}}}{2\beta\rho(p)} \tag{2.4}$$

2.2 Formation Damage

We assume an annular damaged region around the well with an altered permeability k_1 and radius r_1:

$$s = \left[\frac{k}{k_1} - 1\right] \ln \frac{r_1}{r_w} \tag{2.5}$$

of course many combinations of r_1 and k_1 can provide the same skin effect, s.

2.3 Initial Condition

The initial condition is that the pressure throughout the circular system at time zero is equal to a constant:

$$p = p_i \; ; \; t=0 \; , \; r_w \leq r \leq r_e \tag{2.6}$$

Or, in term of pseudopressure:

$$m(p) = m(p_i) \; ; \; t = 0 \; , \; r_w \leq r \leq r_e \tag{2.7}$$

2.4 Inner Boundary Condition

We assume a constant surface production rate. In this case, a material balance on the wellbore leads to:

$$q_{sc} = q_{wb} + q_{sf} \tag{2.8}$$

in which:

$$q_{wb} = -\frac{T_{sc}}{p_{sc}T} V_{wb} \left[C_{g(p)} \frac{T}{\bar{T}} \frac{\partial\left[\bar{p}/\bar{z}(p)\right]}{\partial\left[p/z(p)\right]} \frac{1}{k} \frac{\mu(p)}{2} \frac{\partial m(p)}{\partial t} \right] \tag{2.9}$$

and:

$$q_{sf} = \frac{T_{sc}}{p_{sc}T} \pi h \left[\delta_1 r \frac{\partial m(p)}{\partial r} \right]_{r=r_w} \tag{2.10}$$

2.5 Outer Boundary Condition

The outer boundary is circular.
The condition for the infinite outer boundary is:

$$\lim_{r\to\infty} m(p) = m(p_i) \;\; ; \;\; t > 0 \tag{2.11}$$

If a no-flow condition is imposed at the outer boundary:

$$\left.\frac{\partial m(p)}{\partial r}\right|_{r=r_e} = 0 \;\; ; \;\; t > 0 \tag{2.12}$$

2.6 Dimensionless Quantities

The partial differential equation describing flow was transformed into a dimensionless form using:

Dimensionless time:

$$t_D = \frac{k}{\phi\mu(p_i)c_g(p_i)r_w^2}t \tag{2.13}$$

Dimensionless pseudo-pressure for constant mass rate:

$$m_D = \frac{\pi h T_{sc}}{q_{sc}p_{sc}T}\left[m(p_i) - m(p)\right] \tag{2.14}$$

Dimensionless rate:

$$q_D = \frac{q_{sc}p_{sc}T}{\pi h T_{sc} m(p_i)} \tag{2.15}$$

Dimensionless wellbore storage:

$$C_D = \frac{V_{wb}\left[\frac{T}{\overline{T}}\frac{\partial\left[\overline{p}/z(p)\right]}{\partial\left[p/z(p)\right]}\right]_i}{2\pi r_w^2 h\phi} \tag{2.16}$$

Dimensionless distance:

$$r_D = \frac{r}{r_w} \tag{2.17}$$

Dimensionless diffusivity:

$$\alpha_D = \frac{\mu(p)c_g(p)}{\mu(p_i)c_g(p_i)} \tag{2.18}$$

2.7 Resulting Equations

After substitution and rearrangement, the various flow equations and conditions are:

In the undamaged zone:

$$\frac{1}{r_D}\frac{\partial}{\partial r_D}\left[\delta\, r_D \frac{\partial m_D(r_D,t_D)}{\partial r_D}\right] = \frac{k}{k_1}\alpha_D[m_D(r_D,t_D)]\frac{\partial m_D(r_D,t_D)}{\partial t_D} \tag{2.19}$$

In the damaged zone:

$$\frac{1}{r_D}\frac{\partial}{\partial r_D}\left[\delta_1 r_D \frac{\partial m_{1D}(r_D,t_D)}{\partial r_D}\right] = \frac{k}{k_1}\alpha_D[m_{1D}(r_D,t_D)]\frac{\partial m_{1D}(r_D,t_D)}{\partial t_D} \tag{2.20}$$

Initial condition:

$$m_D(r_D,0) = 0 \tag{2.21}$$

At the interface between damaged and undamaged zones:

$$\delta_1 k_1 \frac{\partial m_{1D}(r_{1D},t_D)}{\partial r_D} = \delta k \frac{\partial m_D(r_{1D},t_D)}{\partial r_D} \tag{2.22}$$

The inner boundary condition is:

$$\alpha_D[m_D(1,t_D)]\frac{\dfrac{\partial[\bar{p}/\bar{z}(p)]}{\partial[p/z(p)]}}{\left[\dfrac{\partial[\bar{p}/\bar{z}(p)]}{\partial[p/z(p)]}\right]_i}C_D\frac{d[m_D(1,t_D)]}{dt_D}-\frac{k_1}{k}\left[r_D\delta_1\frac{\partial m_D(r_D,t_D)}{\partial r_D}\right]_{r_D=1}=1 \qquad (2.23)$$

The outer boundary conditions are:

For the infinite acting case:

$$\lim_{r_D\to\infty} m_D(r_D,t_D)=m_D(r_D,0) \qquad (2.24)$$

For the no-flow condition:

$$\frac{\partial m_D(r_{eD},t_D)}{\partial r_D}=0 \qquad (2.25)$$

2.8 Comments on the Finite-Difference Simulation

The equations described were transformed into a finite difference form by Fligelman (1981). Although a time-weighting factor can be used with simulation, all runs are made with a fully-implicit scheme.

The simulation runs with a dynamic time step. In order to study early time flow accurately without wasting computing time after the transient period, the incremental time step is always taken as 10% of the lowest time value inside a log cycle.

Another important point is the selection of the radial space coordinates. In order to cover the entire reservoir while focusing on the near-well effects, a logarithmic scale was used with the following transformation:

$$v = \frac{\ln r_D}{\ln r_{eD}} \tag{2.26}$$

At early time, the outer reservoir radius is very small. When the pseudopressure at the outer limit is affected by production, v is doubled, which means that the external radius is squared. This provides a large number of points near the wellbore where pressure gradients are large, and fewer mesh points further from the well. We consider programming in the next section.

3. Programming Language and Development Features

The code was developed on a personal computer. The numerical model written in FORTRAN reads the input parameters, computes the gas properties, solves the gas flow equations and computes the results in a "pixel" format ready to be graphed. The interface program that manages the screen and the windows with the use of the mouse and graphic tools such as push-on buttons and pull-down menus is written in C language and compiled with the Windows software development kit. This program performs the acquisition of the main input parameters, calls the FORTRAN simulator through a "shell" command, opens and reads ASCII files containing the results of the simulation, plots graphs on the screen and sends plots to the printer.

The choice of FORTRAN was straightforward because Fligelman's (1981) program used by Couri (1987) was written in FORTRAN for a mainframe computer. The C programming language has had a large impact on the computing world in the last seven years. It allows programmers to write code for both very simple tasks and very sophisticated applications. It allows modular program design, with an extensive set of instructions and capabilities. Finally, many of the programming features of Windows were designed with the C language in mind.

The choice of Windows was also straightforward since this program has become a standard desktop tool for most corporations. Windows is a graphic-based, multitasking windowing environment that runs under MS-DOS. Applications written for Windows have a consistent appearance and command structure that makes new Windows programs easy to use. In order to write programs for Windows, the software development kit provides an abundance of built-in subroutines that allow easy implementation of pop-up menus, scroll bars, dialog boxes, icons, and many other features of a user-friendly graphical interface. The next section presents a user manual for the program.

4. User Manual for REAL GAS FLOW

This user manual is arranged in the order of the operations a user is most likely to perform. Section 4.1 describes how to initiate the program. Section 4.2 presents an overview of REAL GAS FLOW. Section 4.3 describes how to arrange the input data. Section 4.4 explains how to run the simulation. Section 4.5 describes how to plot the results. Section 4.6 presents special features available with the program. Finally, Section 4.7 presents two alternatives for printing, Section 4.8 pinpoints a few details relevant to leaving the application, and Section 4.9 introduces two computing aids with Windows.

4.1 Installation

The hardware configuration is presented and followed by a guide to set up and start the program.

4.1.1 Configuration

REAL GAS FLOW was designed to run on any IBM PC or compatible with a processor 286 or higher (386/486), 16 Mhz or more. A math coprocessor (8087 or higher, 287/387) is not required, but recommended because of the matrix manipulation during the simulation. 2 MB of RAM is enough to use the application, even though Windows is commonly used with 4 MB.

Table 4.1 shows simulation time for a drawdown test run on different configurations of computers. Run-time is sensitive to both processor type and frequency. [The well was flowed for 1000 hours or 40 days. All computers were equipped with math coprocessors.]

	286-6 Mhz	386-16Mhz	486-25Mhz
Darcy Flow	8mins	4mins	2mins
NonDarcy Flow	12mins	7mins	3mins

Table 4.1. Simulation Time on Various Computers

REAL GAS FLOW is a "Window application", and Windows 3.0 or later is required, and so is a mouse. All actions during the program execution are triggered by using the left button on the mouse. Finally, a graphics adapter/monitor either color or monochrome is required.

There is no specific memory requirements for REAL GAS FLOW as long as Windows is installed. REAL GAS FLOW can also be run from a floppy disk drive, but this practice is not recommended because of disk-access time.

4.1.2 Setting Up

REAL GAS FLOW is provided on a disk with a group of example files. The executable files are RGF.EXE and FRANC.EXE. All other files on the disk are not required to run the program, but are provided to give the user a better understanding of the program by looking at old synthetic gas well tests.

RGF.EXE and FRANC.EXE must be copied into the working directory and remain there at all times. To avoid building huge directories, it is recommended that sub-directories be created with old tests. Old synthetic data can be retrieved and processed by REAL GAS FLOW independently of their location on the hard disk or any floppy disk. To install the program and the examples in a directory called RGF:

1. Insert the RGF floppy disk in drive A
2. Go into the disk or directory where RGF is to be created
3. Type: mkdir RGF
4. Type: cd RGF
5: Type: copy A:*.*

Files for REAL GAS FLOW have been named in a way that makes it easy for a user to organize the working directory. When we design a new well test, a name for the test must be defined, for example: EXAMPLE. All data files related to the test EXAMPLE will differ only by their three character extensions : EXAMPLE.FLR is the flow rate history file; EXAMPLE.ACT is the pressure history file at the active well, and so on. More on the different files used by REAL GAS FLOW is presented in Section 4.2.3.

After having made a backup copy of the disk provided and copied all files and especially RGF.EXE and FRANC.EXE in the working directory , REAL GAS FLOW must be set up into "program manager ". Using the mouse, click on WINDOWS SET UP inside the MAIN folder of program manager, then click on OPTIONS and select SET UP APPLICATIONS. Select the hard disk on which the working directory is located and then run the search by clicking OK. This can take a few minutes depending on the number of applications on the disk. After the search, select RGF.EXE on the left window in the dialog box, click on ADD and then OK. RGF will be added, with its recognizable icon as a window application. At this point of course, it can be moved into any group.

4.1.3 Starting Up

REAL GAS FLOW can be started from the program manager window by double-clicking on its icon. At this point, the main menu will fill the entire screen. REAL GAS FLOW can also be started under the DOS environment by going into the working directory and typing the command: WIN RGF.

4.1.4 A Quick Guide to REAL GAS FLOW

Before learning more about the program, users will familiarize themselves with REAL GAS FLOW (RGF) by using a first example. The following example files have been provided on the disk: GUIDE.MAI and GUIDE.FLO. The following instructions should be followed:

Start the program.
Click on INPUT in the main menu bar.
Click on OPEN inside of the new window.
Select GUIDE.MAI in the working directory and press OK.
The main parameter input file is now displayed.
Click on OK at the bottom of the window.
Click on RUN in the main menu bar.
The simulator is now running in a DOS window for a few minutes.
Click on OUTPUT in the main menu bar.
Select GUIDE.ACT in the working directory.
Click on OK in both messages on the screen.
Click on PLOT MENU in the main menu bar.
Select Log-Log plot.
Click on GRAPH in the main menu bar.
The results of the simulation are now displayed.
Click on ERASE in the main menu bar.

The user should now have a basic understanding of the software capabilities.

4.2 Overview

This section provides an overview of REAL GAS FLOW. The methodology followed by the program is presented, the main menu bar is introduced, and a description of the files used during a simulation is provided.

4.2.1 Methodology

After starting RGF, the main menu bar appears on the screen. The process is self explanatory. Figure 4.1 shows the arrangement of the major components of REAL GAS FLOW.

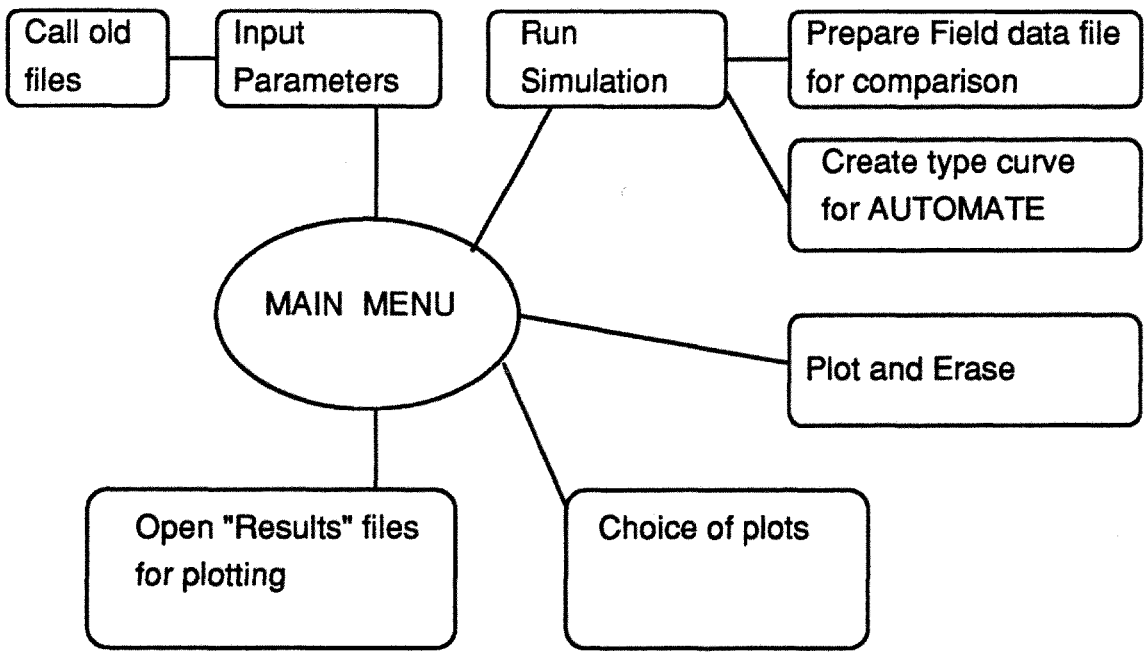


Figure 4.1. Main Structure of REAL GAS FLOW

The first task is to input all the parameters which are relevant to the test we intend designing. Clicking on the INPUT button provides a window (or dialog box) where the

characteristics of the well and the reservoir, the gas properties and the flow considered must be defined. At this point, we can decide if we wish to compare field data with the computed data produced by the program, and also whether computed results should be stored as a future custom type curve for the AUTOMATE well test interpretation software.

Instead of inputting a new set of parameters from scratch, an old test can be called and opened. This can be done to verify parameters from old tests, or to modify a parameter from a previously-run test design in order to perform a sensitivity analysis. Once the list of input parameters is filled, it can be automatically saved by simply clicking on an OK button. Furthermore, a control device reminds one that we cannot move on and save the input file if a critical parameter has been accidentally omitted.

The second task is to run the simulator by clicking on the RUN button in the main menu bar. Windows 3.0 will open a DOS window, and perform the calculations. Run-time will vary from computer to computer, depending on the processor. During computation, a message is displayed in the DOS window giving information on the status of the simulation. The simulator also prepares field data files (if any) so they can be displayed on the screen and compared with computed data. Finally, the simulator will save the results in a custom type curve for AUTOMATE (if required), and additional messages will appear on the screen for this purpose.

The next logical step is to view the result of the simulation. Clicking OUTPUT in the main menu Bar gives access through a dialog box to the content of any directory on the hard disk or any floppy disk. An output file from the last simulation, or any old simulation can be loaded by selecting the desired file in the appropriate directory and clicking OPEN in the dialog box. A set of messages informs us that the computer is loading the file and when this task is completed. By clicking PLOT MENU on the main menu bar, we have a choice of different plots ranging from a simple rate history plot to straight line analysis or log-log plots. Graphs can be plotted on the screen, superposed, or erased by using the two remaining buttons in the main menu bar.

4.2.2 Main Menu Bar

The use of the six main buttons on top of the screen when the program is started was explained briefly in the previous chapter. A quick summary follows:

INPUT: Calls a dialog box that allows one to design a new well test or call an old one.

RUN: Run the FORTRAN simulator in a DOS window.

OUTPUT: Allows user to select output files in any direction in order to create graphs on the screen.

PLOT MENU: Present a list of nine different graphs.

GRAPH: Execute the plotting of data on the screen.

ERASE: Clears the screen, leaving only the main menu Bar.
The ERASE button can be used at any moment if the user wishes to clear the screen.

The top title inside the screen heading reads REAL GAS FLOW when the program is first started. This heading will change throughout the use of the program to include the title of a graph when a plot is made, and the name of the output file that is being graphed.

4.2.3 File Types

All files used by REAL GAS FLOW are ASCII files. All the files related to the same simulation will have the same name and will differ only by the three-digit extension. Some files are created by the program while others must be created by a user.

4.2.3.1 Files Created by a User

Let's suppose we are working on a well test design named TEST1. We must create a file TEST1.FLO containing the rate history using an editor. The details for all input files will be found in Section 4.3: Setting Up the Data. If we wish to compare the computed data from the simulation with field data, we must provide the program with a pressure history file named TEST1.PRS.

4.2.3.2 Files Created by REAL GAS FLOW

The first file that RGF will create is the main parameter input file called by clicking the INPUT button in the main menu bar. This parameter file is saved by RGF under TEST1.MAI. The most important file containing graphic information from the simulation data produced by the program is TEST1.ACT for all data related to the active well. This file is the largest one used by the program, and will require on the average 10 to 20 Kbytes of memory, depending on the duration of the test.

Finally, if a file TEST1.PRS was provided by a user in order to compare computed and field data, a new file TEST1.FLD will be created by RGF containing graphic information related to the field data. Also, if RGF is asked to generate a custom type curve to be used with AUTOMATE, a file TEST1.MSG will be created containing the type curve, as well as other relevant information.

At all times when RGF is used, an internal file INPUT.FIL requiring 10 bytes of memory will appear on the working directory. This file is very small, and can be ignored or deleted by a user because it will be recreated as needed by the program.

For each test design there will be between four and eight files on the working directory depending on the different options selected.

4.3 Setting Up the Data

In order to create the rate history file, a user will require an editor under DOS that saves data in ascii format. The same goes for the field pressure history file if needed. The main data file (TEST1.MAI) presented in detail in the next chapter does not require the use of an editor since it is created by RGF.

4.3.1 Main Data File

After starting REAL GAS FLOW, a user should press the INPUT button in the main menu bar. This action produces a dialog box on the screen with the heading: "Parameter Input" as in Fig. 4.2.

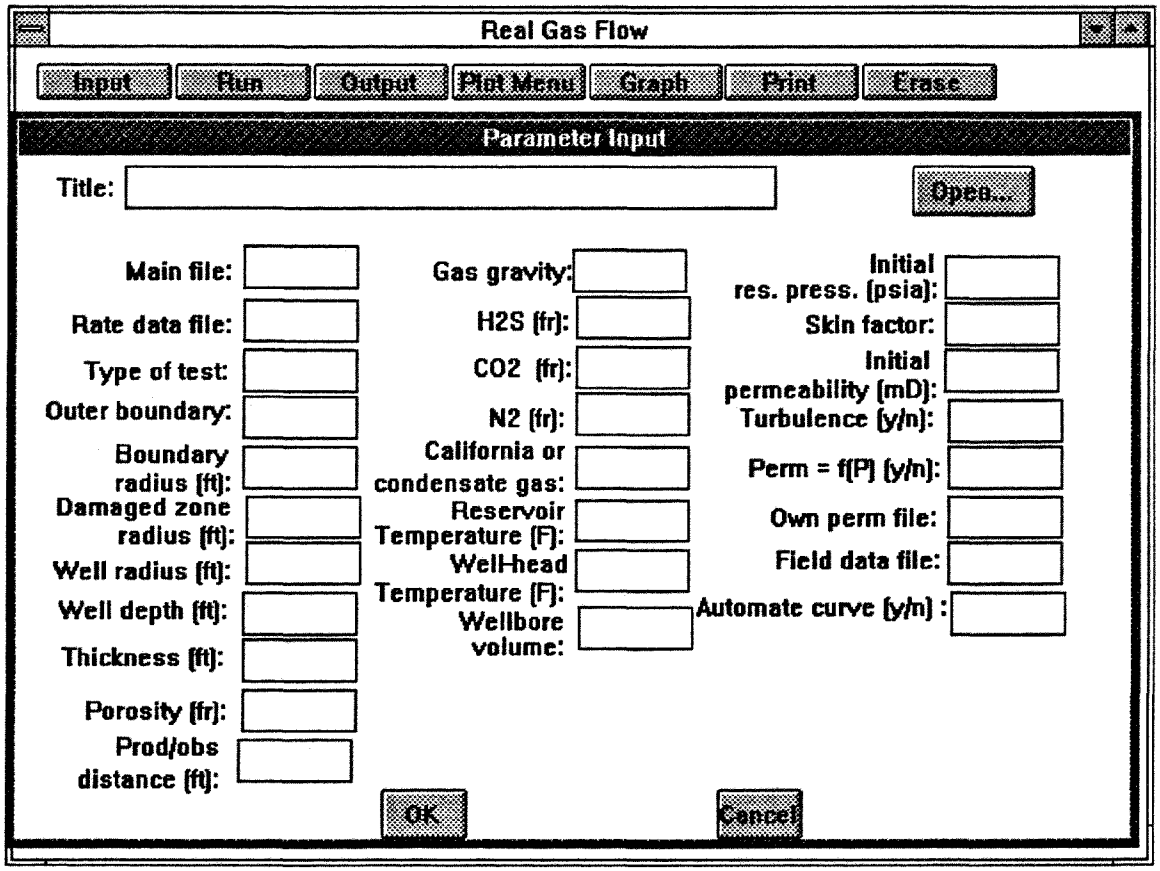


Figure 4.2. Input Parameter Window to Create and Display Main Input Files

There are 24 parameters that should be considered, plus a title that is optional. There are three action buttons in the dialog box: OK, CANCEL, and OPEN.

OPEN: Opens a second dialog box displaying the content of a selected directory. This directory will be refered to as the active directory. Selecting a main file (with the extension .MAI) in this second window will display it in the parameter input window.

OK: Accepts the parameter presently displayed in the dialog box as input data for the next run of the simulator. These parameters could be freshly entered or could come from an old file with the use of the OPEN button. OK will validate the display and make the dialog box disappear. The parameter file will be saved automatically in memory inside the active directory. OK will save all files in the active directory. Therefore the active directory and the working directory should be the same in order to run a simulation. If some parameters have been omitted intentionally or accidentally, pressing OK will produce an error message.

CANCEL: Closes the parameter input window without saving any file, and goes back to the main menu bar.

4.3.1.1 Opening an Old Main File

A main file can be opened in order to be displayed for changes or to run a simulation. A user can display any main file on the template on the screen by clicking OPEN in the parameter input window. The new OPEN window will display all files in the active directory. To limit the list to the main files only, the user can type *.mai in the text box instead of *.* and then click on OPEN or by hitting the ENTER or RETURN key on the keyboard. Only *.mai files will be displayed in the list box. If there are more choices than can fit the list box, scroll bars are provided so that the mouse can be used to move up and down the list.

Changing the active directory will show a weakness of the Windows environment: the user must possess a slight knowledge of the DOS environment.

One alternative to Change Directory is to type the entire path for the new directory inside the text box. For example, if the present active directory is, as in Fig.4.3 D:\we\francois\newtest*.mai a user can type in the text box: D:\we\john\doe*.mai to have access to this set of files.

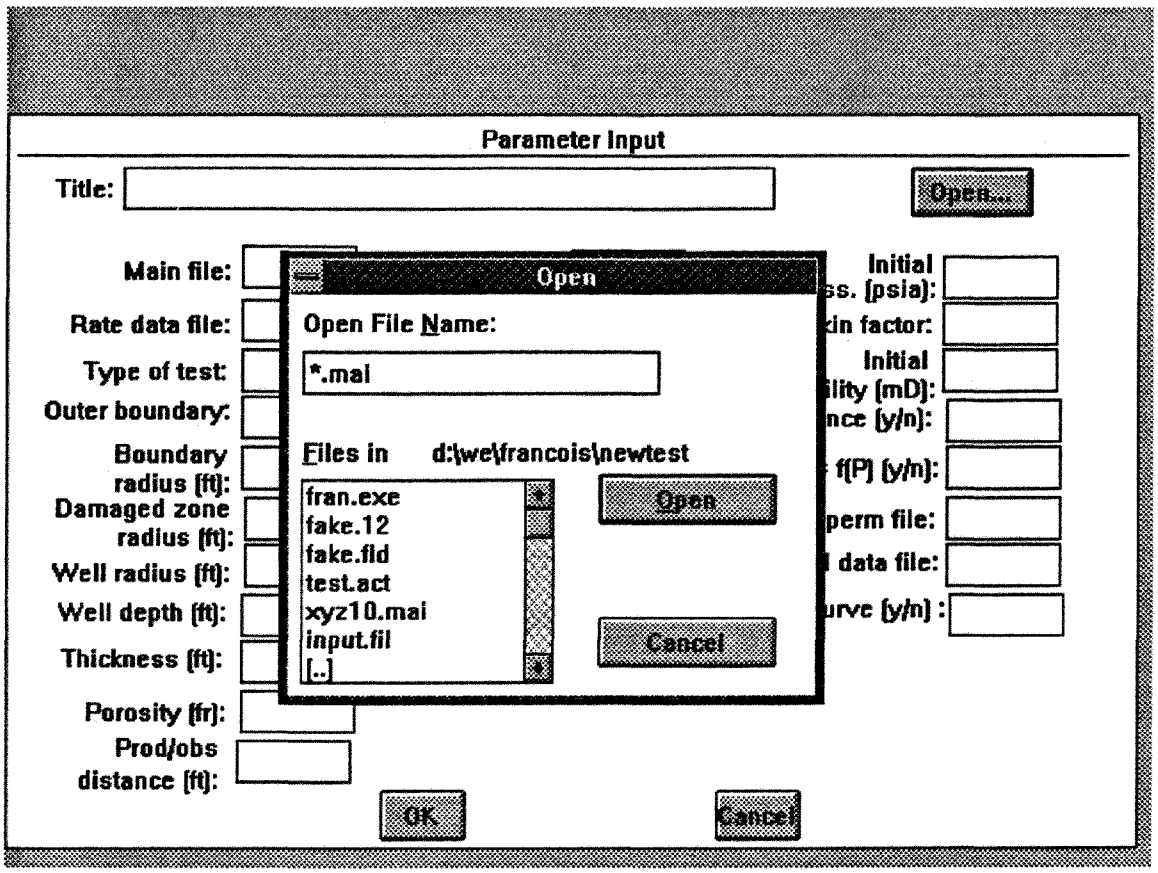


Figure 4.3. OPEN Window to Access Files on Disks

An alternative is to type in the text box: D:*.mai and use the mouse to select the successive subdirectories by double-clicking on them. Once a user is in the desired directory, the quickest way to select a file is to double-click on its name.

Old main files can be used to run the simulator if old result files have been lost or erased. However the most useful aspect of displaying old main files is to be able to perform sensitivity analysis. This can be done by displaying an old main file, changing one input parameter (permeability for instance) and saving the new input file under a name that is slightly different from the previous one. Figures 4.4 and 4.5 show two main files using the same rate data file. The two files differ only by the value of permeability (10 and 100) and by their name.

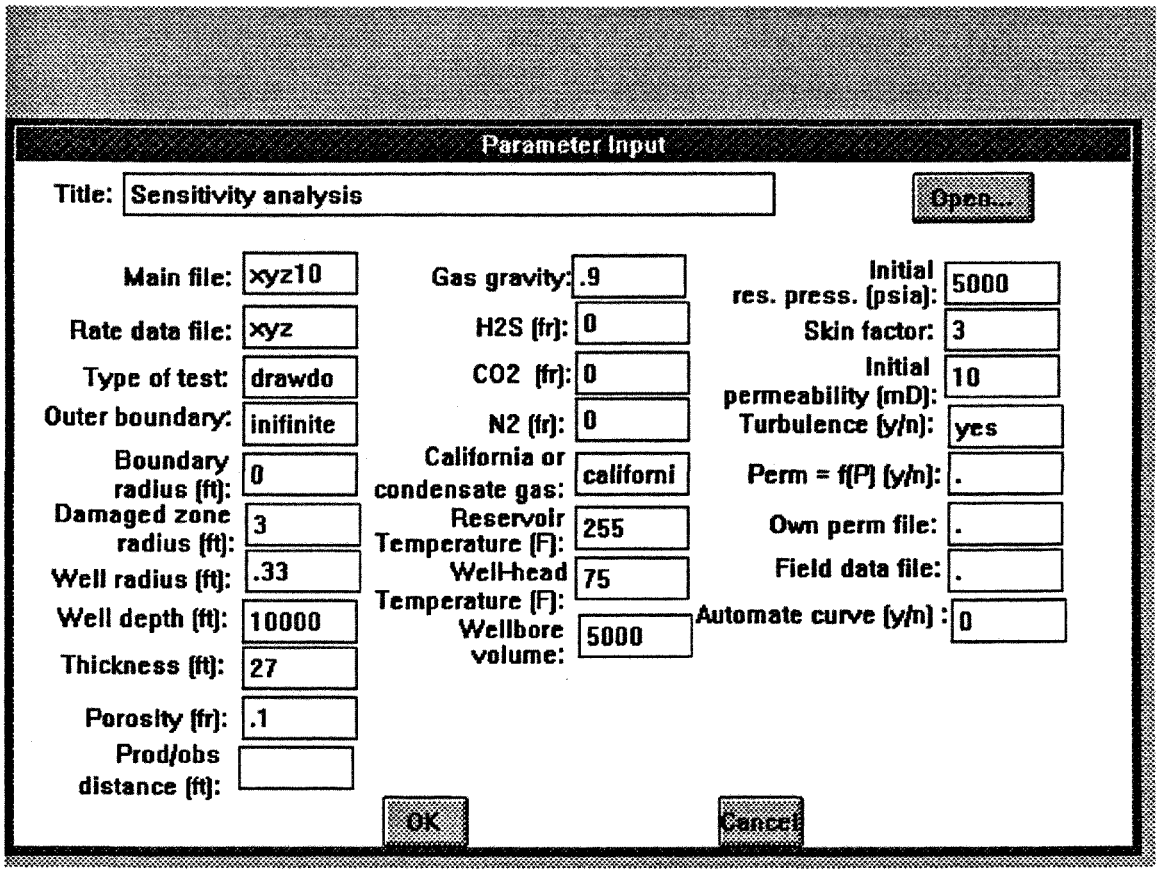


Figure 4.4. Example of File for Sensitivity Analysis, with k=10mD

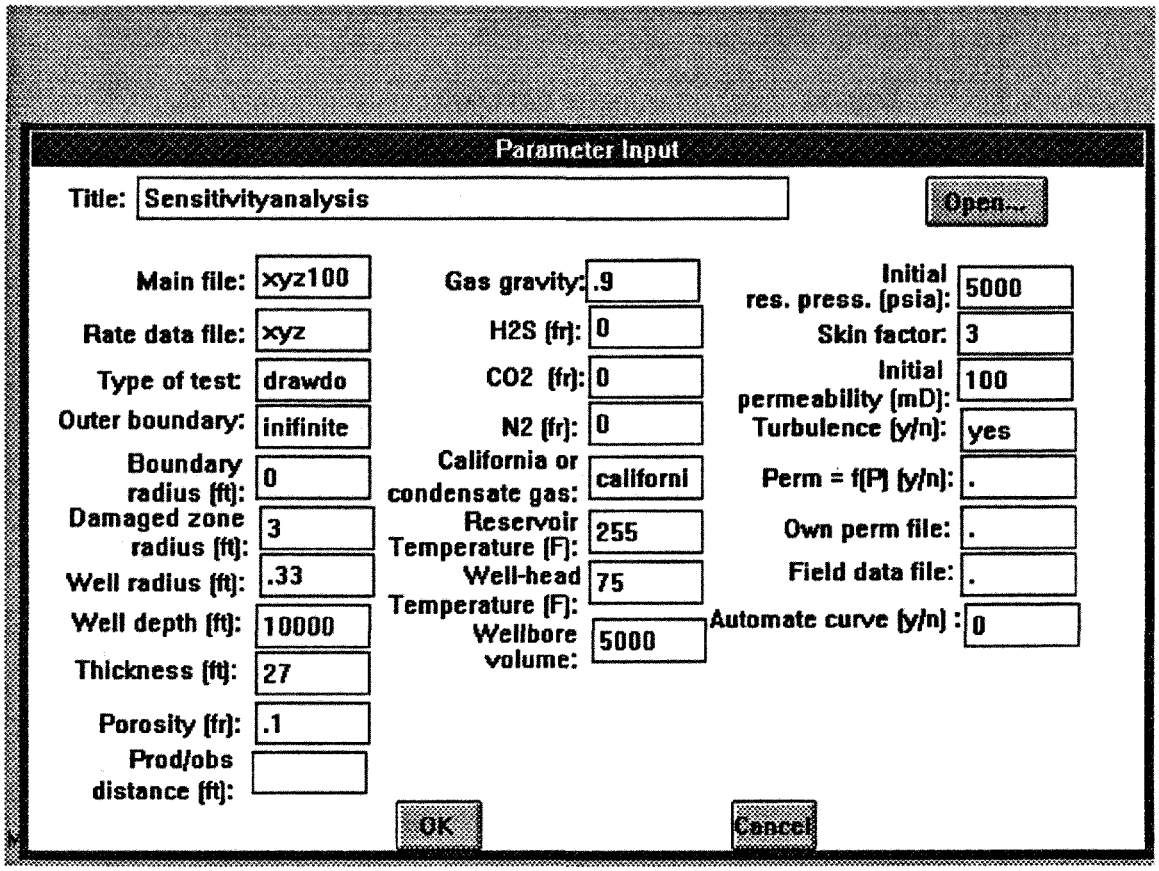


Figure 4.5. Example of File for Sensitivity Analysis, with k=100mD

Once an old file is displayed inside the parameter input window, clicking on OK will save the displayed file in the active directory and, if and only if, the active directory is also the working directory (that contains RGF) will the simulator be able to run. This is an area where caution is required because distraction can lead to running the simulator with an unwanted input file.

4.3.1.2 Creating a New Main File

In order to run a new simulation, a main file must be created. The well test can be designed by trial and error, and RGF permits one to perform sensitivity analysis in an easy fashion.

Table 4.2 is a practical summary of the 24 fields required to complete the file.

Name of Field	Meaning	Unit expected
title	title of test to run	optional
main file	main input file name	1 to 8 letters
rate data file	rate data file name	1 to 8 letters
type of test	drawdown or injection	letters
outer boundary	defines external boundary	letters
boundary radius	external radius	feet
damaged zone radius	-	feet
well radius	-	feet
well depth	-	feet
thickness	pay zone thickness	feet
porosity	-	fraction
gas gravity	relative gravity to air	fraction
CO2, H2S	gas impurities	fractions
type of gas	-	letters
reservoir temperature	-	Fahrenheit
well head temperature	-	Fahrenheit
wellbore volume	-	cubic feet
initial reservoir pressure	-	psia
skin factor	skin damage	dimensionless
permeability	average in reservoir	mD
turbulence	consider or not	yes / no
field data file	name of file if applicable	1 to 8 letters
Automate type curve	create or not	code: 0, 1 or 2,

Table 4.2. Summary of Main Input Data File

To fill up a file, the mouse must be used to move from one field to another, clicking into the box of the field to enter next. The order in which the fields are entered is irrelevant. Only the title is optional. All other fields are required. Both lower case and upper case can be used when creating a file, as long as the user is consistent! For instance, both "drawdown" and "DRAWDOWN" would be accepted as type of test, but "Drawdown" would be rejected. The program uses only oil field units.

To save a main file that has just been created, the user should press the OK button. The file will be saved in the active directory (again, this is the directory shown when clicking on OPEN in the parameter input window or on OUTPUT in the main menu bar.)

We cannot overemphasize the point that in order to run the simulation, the main file must be in the working directory. The working directory is the directory that contains RGF.EXE.

Saving a main file will add the extension .MAI to whatever was entered in the main file field. If the user wishes to call a test TEST123, RGF will automatically save the main file as TEST123.MAI. If a file TEST123.MAI already exists in the directory, RGF will write over it!

If the user presses CANCEL instead of OK, the parameter input window will disappear, and all parameters currently entered will be erased from the screen without being saved. Also RGF cannot be minimized (upper right arrow on window applications) while the parameter input window is on the screen.

Finally, we should be aware that under the Windows configuration, all characters do not require the same space on the screen. For instance an "m" will require more than three times the space of an "i". On the average, eight characters should fit in each field box, but a box that can accept fourteen "i" characters will accept only four "m" or seven "a" characters. When choosing names for the different data files, remember that DOS requires that spaces or periods are excluded as characters.

4.3.1.3 Detailed Description of Each Field

(1) TITLE--This field is optional, and can contain any type of character. It is useful to a user, in order to remember specific runs, dates, or well locations.

(2) MAIN FILE--This is the filename under which the input parameters will be saved, with the addition of the extension .MAI. Any character can be used except periods or spaces, with a maximum of eight characters.

(3) RATE DATA FILE--Requirements are the same as for the main file name. The user should give the name of the file they previously created under a DOS editor, and saved with an extension .FLO. Example: NAME.FLO. The rate data file must be saved or copied in the working directory as well as all other relevant input files in order for the simulator to run.

(4) TYPE OF TEST--In the future, Buildup test or multirate test could be added to REAL GAS FLOW. In the present version, only drawdown and injection tests can be run. The user should therefore write INJECTION or DRAWDOWN getting as many letters in the box as possible. A drawdown is defined as a flowing period where at time zero the pressure is constant and identical at all points throughout the reservoir.

(5) OUTER BOUNDARY--Three options exist. The program checks only the first two letters of the input, but as many letters can be used as will fit in the box.

Infinite Acting:	The program checks on:	IN.
No Flow:	The program checks on:	NO.

with:

The condition for the infinite outer boundary is:

$$\lim_{r\to\infty} m(p) = m(p_i) \;\; ; \;\; t > 0 \tag{4.1}$$

If a no-flow condition is imposed at the outer boundary:

$$\left.\frac{\partial m(p)}{\partial r}\right|_{r=r_e} = 0 \;\; ; \;\; t > 0 \tag{4.2}$$

(6) BOUNDARY RADIUS--If the outer boundary condition is no flow or constant pressure, a radius must be provided in feet. In the case of infinite acting, an entry such as zero ("0") should be made.

(7) DAMAGED ZONE RADIUS--This field concerns a cylindrical zone around the wellbore altered by the drilling process which has a permeability different from the reservoir. A damaged zone radius in feet is required. If there is no damage, a radius of zero ("0") is required.

(8) WELL RADIUS--A value in feet is required.

(9) WELL DEPTH--This value is required to compute the gas properties table in order to compute wellbore storage. A value in feet is required.

(10) THICKNESS--The reservoir is modeled as a constant thickness circular layer. A value in feet is required.

(11) POROSITY--Enter fractional porosity between 0.01 and 0.8.

(12) GAS GRAVITY--The specific gravity of the gas compared to air at standard conditions is required.

(13-14) The concentration of the gas impurities N2, CO2, and H2S should be entered as fractions. Values should therefore be between zero and 1.0.

(15) TYPE OF GAS--This information is required in order to select the proper Standing (1977) correlation. RGF performs a test on the first letters of the information provided. The user is encouraged to write as many letters as possible in the field box.

California:	The program checks on:	CAL
Condensate:	The program checks on:	CON

(16) RESERVOIR TEMPERATURE--A temperature value in degrees Fahrenheit is required. The simulated flow by RGF is isothermal. The temperature value is used both in the flow equations and to generate the gas properties table.

(17) WELLHEAD TEMPERATURE--As for the reservoir temperature, a degree Fahrenheit value is required. This parameter is used to compute the temperature gradient in the well. The temperature is required to compute wellbore storage.

(18) WELLBORE VOLUME--The wellbore volume between the pay zone and the surface is required in cubic feet. This value is used to compute wellbore storage, and therefore the inner boundary condition. As presented in Eq. 2.16, the dimensionless wellbore storage is:

$$C_D = \frac{V_{wb}\left[\frac{T}{\bar{T}}\frac{\partial[\bar{p}/\bar{z}(p)]}{\partial[p/z(p)]}\right]_i}{2\pi r_w^2 h\phi} \tag{4.3}$$

(19) INITIAL RESERVOIR PRESSURE --This is the pressure in psia at any point in the reservoir before the flowing period.

(20) SKIN FACTOR--This value is dimensionless and reflects the formation damage around the well. If k is the permeability in the reservoir and k_1 the permeability in the damaged region, the skin factor s is defined in Eq. 2.5:

$$s = \left[\frac{k}{k_1} - 1\right] \ln \frac{r_1}{r_w} \tag{4.4}$$

(21) PERMEABILITY--This is the constant permeability in mD of the homogeneous reservoir.

(22) TURBULENCE--A yes or no answer is required here. RGF checks only the first letter and looks for a y or an n in either upper or lower case. YES means that turbulence effects will be included by RGF in the Forchheimer equation. NO means that the flow will follow Darcy's law.

(23) FIELD DATA FILE--This field asks for the name of a time/pressure file allowing the user to compare field test data with the computed results from a simulation. More is presented on this feature in Section 4.4.1. If field data is to be used, the data file must be created by the user with an editor under DOS before starting the program. As with other data files, the field data file should be saved in the working directory with the extension .PRS. Example: NAME.PRS. In the parameter input window, the file name should be provided without the extension. If field data will not be used, a space (blank) should be entered in the field box in order to save and leave the input parameter screen.

(24) AUTOMATE TYPE CURVE--Creates user-made type curves to be used with the AUTOMATE-II Computer-Aided Well Test Analysis package. It is possible to build a library of type curves in order to match field data, or to run a non-linear regression. The requirements for this field are:
Enter "0" if the user doesn't want a type curve to be saved from the simulated results.
Enter "1" for a standard p_{pD} versus t_D type curve, where the estimated parameters in AUTOMATE will be k, the permeability, and ϕ the porosity.
Enter "2" for a storage type curve p_{pD} versus t_D/C_D where the estimated parameter in AUTOMATE will be k, the permeability, and C, the storage factor.

More on this utility will be found in Section 4.6: Special Features.

4.3.2 Flow Rate File

Some conventions must be adopted.

(1) Time zero is the time at which the well starts flowing.

(2) All time data are in hours.

The flow rates will be in Mcf/D. It is mandatory to have a flow rate data point at time zero since we need to specify at what time the well started to flow. A typical flow rate data file for a constant rate drawdown test is:

```
0.0     5000.
500.    end
```

This means that the well flows at 5000 Mcf/D from time zero for 500 hours. If the user is familiar with AUTOMATE, the same drawdown on AUTOMATE would have a rate data file like:

```
0.0     5000.
```

RGF requires one more line in order to tell the simulator when to stop. This means that any rate data file already created for AUTOMATE can be used by adding a second line containing a time and "end" or "stop" instead of a rate. Notice the free format for the data file. The data can appear in any column in any format, provided that there are only two numbers (a time and a flow rate) on each line. It is important to provide RGF with a final line that specifies when to stop computing. To find an easier way to modify data files under Windows without having to go back to the DOS environment, please refer to Section 4.9: Computing Aides.

4.4 Running the Simulation

Clicking on RUN in the main menu bar will run the simulator. The main issue is: what case is being run? A second point concerns the various messages on the screen during the simulation.

4.4.1 Running the Right Case

There should be no uncertainty as long as the user gives special attention to the next few steps:

(1) Make sure that the main data file is created and saved in the working directory. This is always the case if the active directory and the working directory are identical.

(2) Be aware that the main file that will feed the simulator is the last one that was OK'ed before pressing RUN.

(3) Make certain that all files required for a specific simulation to run are available in the working directory.

(4) Double check values and units in the main data file before saving it by pressing OK.

4.4.2 During the Simulation

MS Windows 3.0 runs the simulator in a DOS window. The window will display periodically what fraction of the computation has been completed. A simulation ends by informing the user that "Everything went fine" and that "Data are now being stored for graphics". If all data files have been set accordingly, the only error message that might appear on the screen is to inform the user that with the parameters provided, convergence toward a solution is not possible. Mistakes made while editing the input data files can end the simulation prematurely. In this case the final message will not appear on the screen. This is a sign for the user to double check input files, beginning with their location in the proper directory. Once a simulation is completed, the screen returns to the main menu which appears when RGF is first started.

4.5 Plotting Results

Nine different graphs can be obtained from the results by the push of a button. To get a list of the graphs, press on PLOT MENU to get the screen shown in Fig. 4.6. The Horner plot and the line source solution are not accessible but have been placed in the menu for future development.

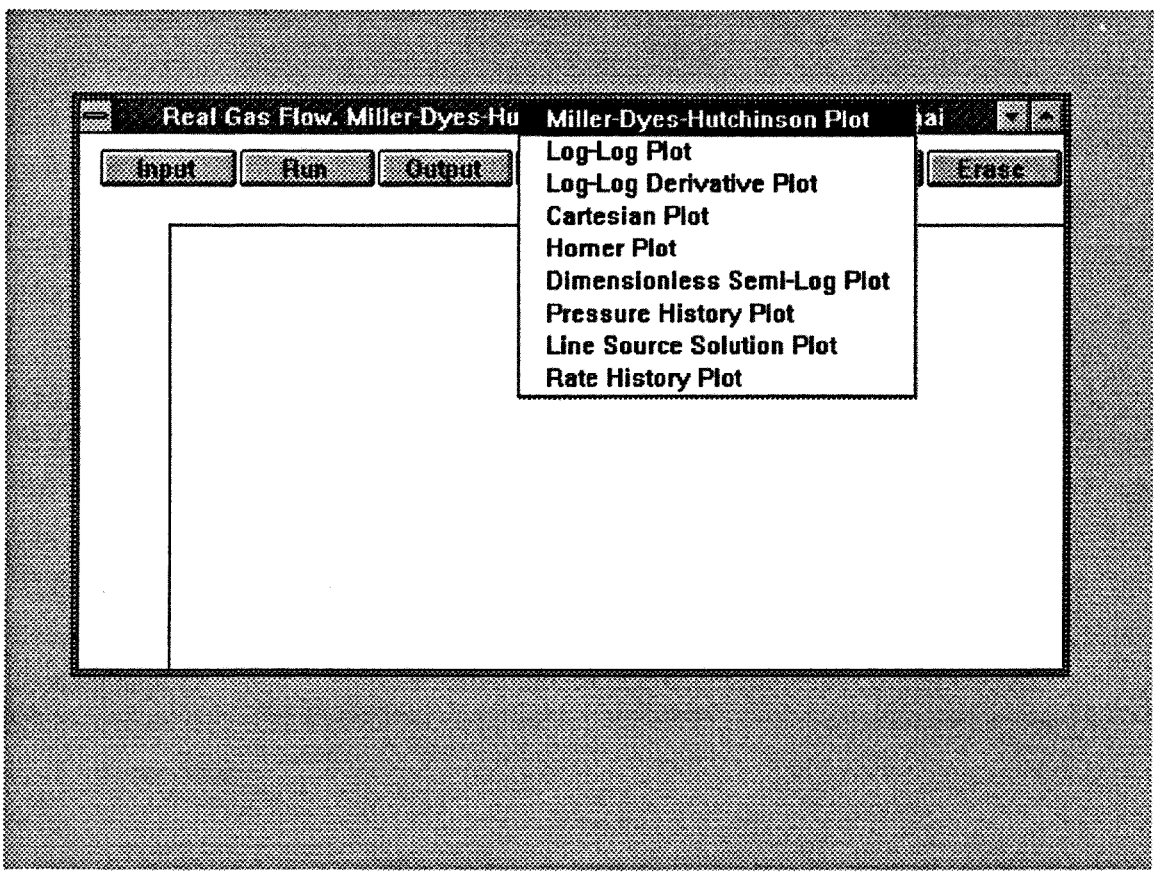


Figure 4.6. Choice of Available Plots for Viewing Results

4.5.1 Plot Menu

(1) Miller Dyes Hutchinson: Semi-log plot of pseudopressure in psia versus time in hours

(2) Log-log : Log-log plot of log(Δmp) in psia versus log(Δt) in hours

(3) Log-log derivative: Log-log plot of $\Delta t \log(\partial mp/\partial \Delta t)$ in psia versus log(Δt) in hours

For the computed data from the simulator, the derivative is a simple forward first degree finite-difference calculation with no attention to noise reduction.

(4) Cartesian: Cartesian graph of normalized pseudopressure in psia versus time in hours
The normalized pseudopressure for this plot is:

$$P_{Pn} = P_i + \frac{\mu_i z_i}{P_i} \int_{P_i}^{P} \frac{PdP}{\mu z} \tag{4.5}$$

This is an excellent way to express the pseudopressure because a unit of pressure is obtained. This expression was introduced by Meunier et al. (1987) and is also presented in Horne (1990).

(5) Horner: Not accessible
(6) Dimensionless: Semi-log plot of dimensionless pseudopressure versus dimensionless time $t_D/10^6$. The dimensionless variables are described by Eqs. 2.13 and 2.14.
(7) Pressure history: Cartesian plot of pressure in psia versus time in hours
(8) Line source solution: Not accessible
(9) Rate history: Cartesian plot of rate in Mcf/Dversus time in hours

4.5.2 Overview

Any output file containing results from simulation can be opened to create graphs on the screen. The following two sections describe how to open an output file and plot the data on the screen. A third section introduces the use of a cursor to pick points on a graph.

4.5.2.1 Opening an Output File

RGF treats files from the current simulation and those run in the past in the same manner. To select a file for plotting, go to the main menu bar and press the OUTPUT button.

An OPEN window similar to that for input files appears on the screen as in Fig. 4.7. The pressure response data at the active well for a simulation where the main file was EXAMPLE.MAI will be found in the file EXAMPLE.ACT. RGF created this file in the working directory at the end of a simulation. Using the file manager utility of Windows 3.0, a user can move the output files (with an extension .ACT) to different directories or sub-directories for storage.

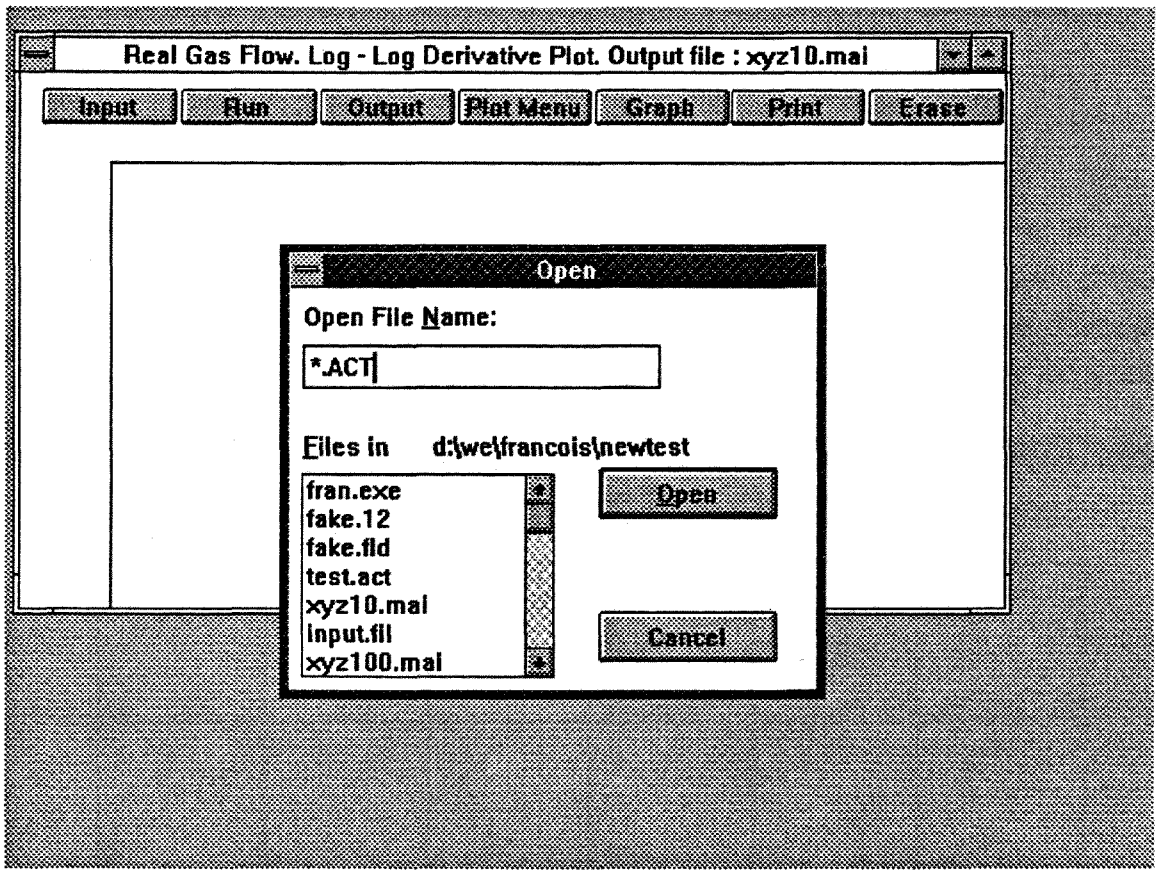


Figure 4.7. OPEN Window to Access Output Files Stored on Disks

To open a file, the procedure is the same as for opening a main file. It can be easier to view the available files by typing *.ACT in the text box instead of *.*. The user can move up or down in the tree of directories and change disk drive to search for old files, and can cancel the search by pressing CANCEL with the mouse.

Pressing OPEN after having selected an output file with the mouse will load the data into the graphics subroutine and will trigger two messages that require the approval of the user by clicking on OK with the mouse. The two messages are: "About to read & convert data points. This may require a few seconds to process." and "Completed reading of output file." The screen will then go back to the initial main menu bar. RGF is now ready to graph results. IMPORTANT: INPUT and OUTPUT show the same active directory. If output files are selected in a directory different from the working directory, the user should make certain that the next time a main file is created, it is saved in the working directory.

4.5.2.2 Plotting the Data

Once an output file has been selected, select a specific plot by clicking on the plot menu, and clicking on the GRAPH button as in Fig. 4.8. The ERASE key erases everything on the screen except the main menu bar. It can be used to view different graphs in sequence. It is also possible to superpose two or more graphs on the same screen.

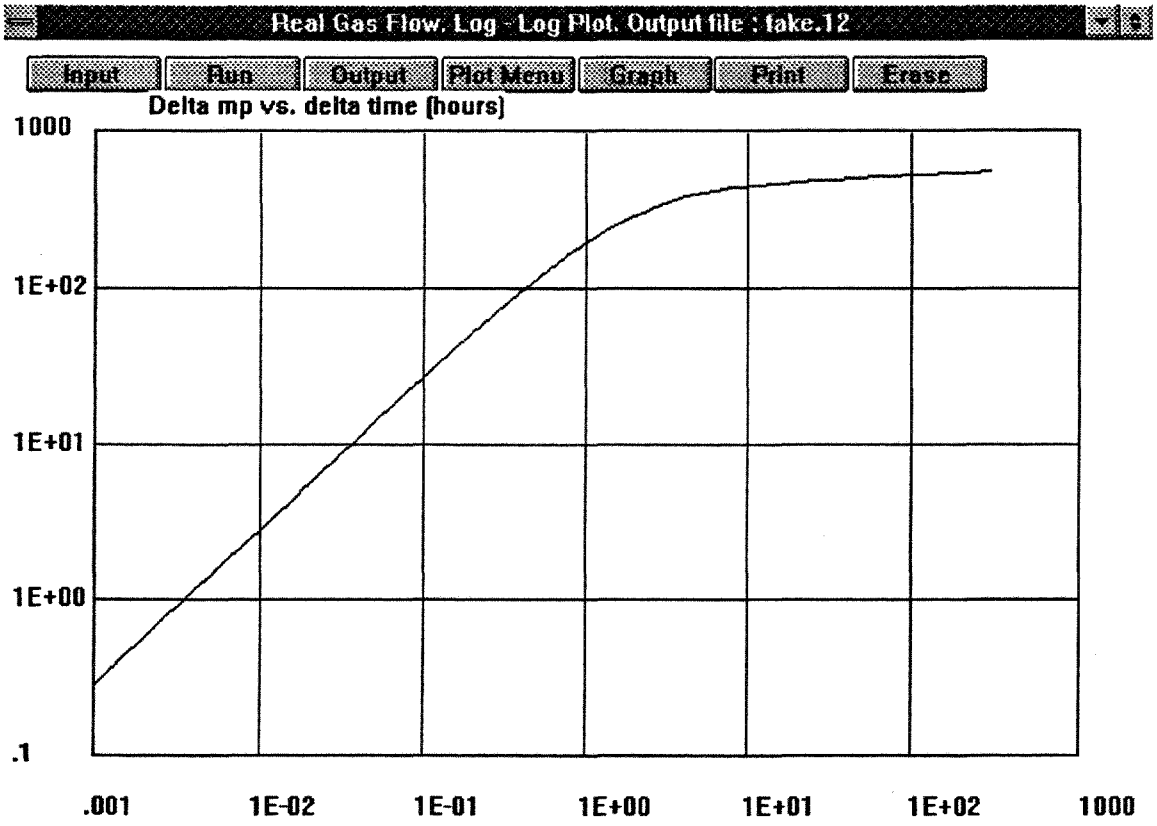


Figure 4.8. Typical Log-Log Plot with REAL GAS FLOW

Since different plots have different scales, only the log-log plot and the log-log derivative plot are meaningful to display together. To do this, select the output file, then select one plot in the PLOT MENU, press GRAPH, select the second plot in the plot menu and press GRAPH again. The screen should display a graph similar to Fig. 4.9.

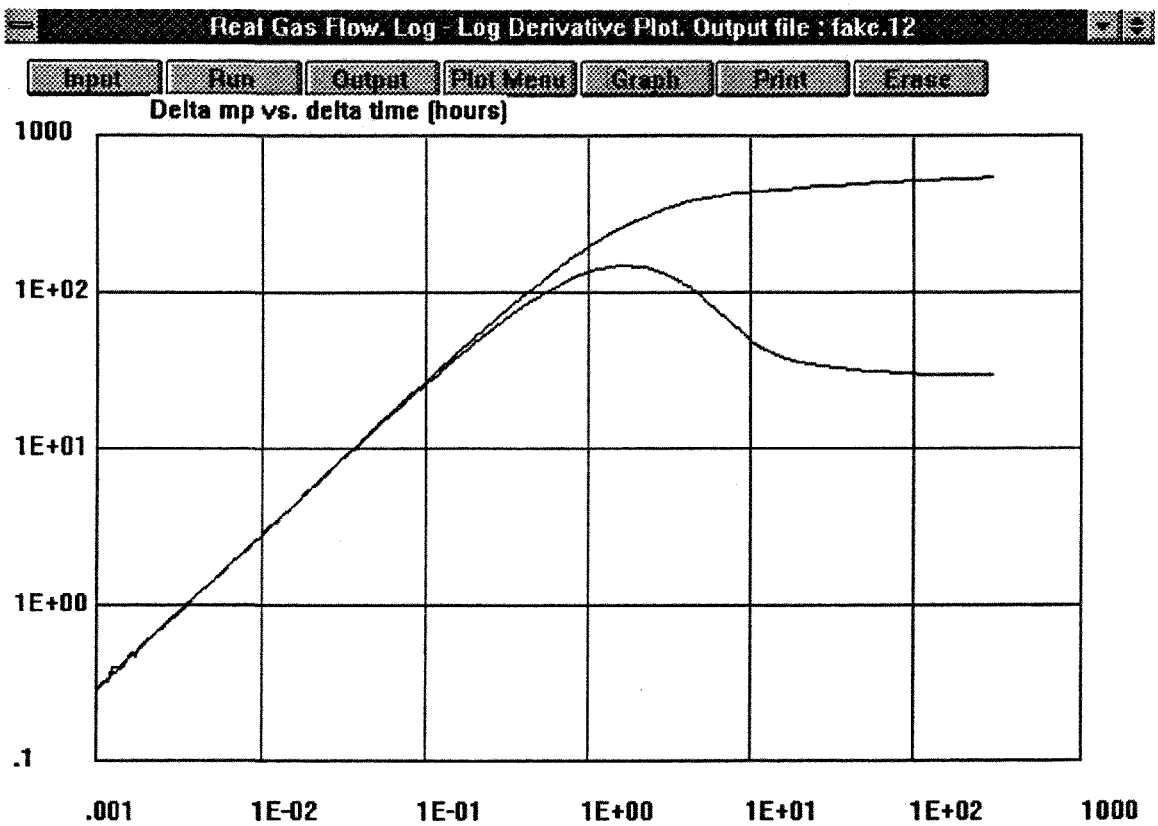


Figure 4.9. Typical Graph of Both Log-Log and Derivative Plots

RGF's ability to superpose plots from different output files is useful in order to compare results in the case of a sensitivity analysis. An unlimited number of plots can be superposed so as to see the influence of one parameter on well test results. If two runs were made with two different skin factors, s=+2 and s=+10, the output files would be SKIN2.ACT and SKIN10.ACT. Let's assume we wish to see a log-log plot for both files. First select skin2.act with the OUTPUT button, then select log-log plot in the

PLOT MENU and click on GRAPH. The first plot is now on the screen. Using the OUTPUT button again, select skin10.act, click on GRAPH again, and the second plot appears. See Fig. 4.10. There is no need to go to PLOT MENU since RGF kept in memory that the last selection was LOG-LOG. This can be done repeatedly if more than two plots are desired. Superposing plots for comparison is meaningful only if the scales are the same. This is almost always the case when a sensitivity analysis is performed. On the other hand, two very different well tests would not be comparable, because the plot scales could be different.

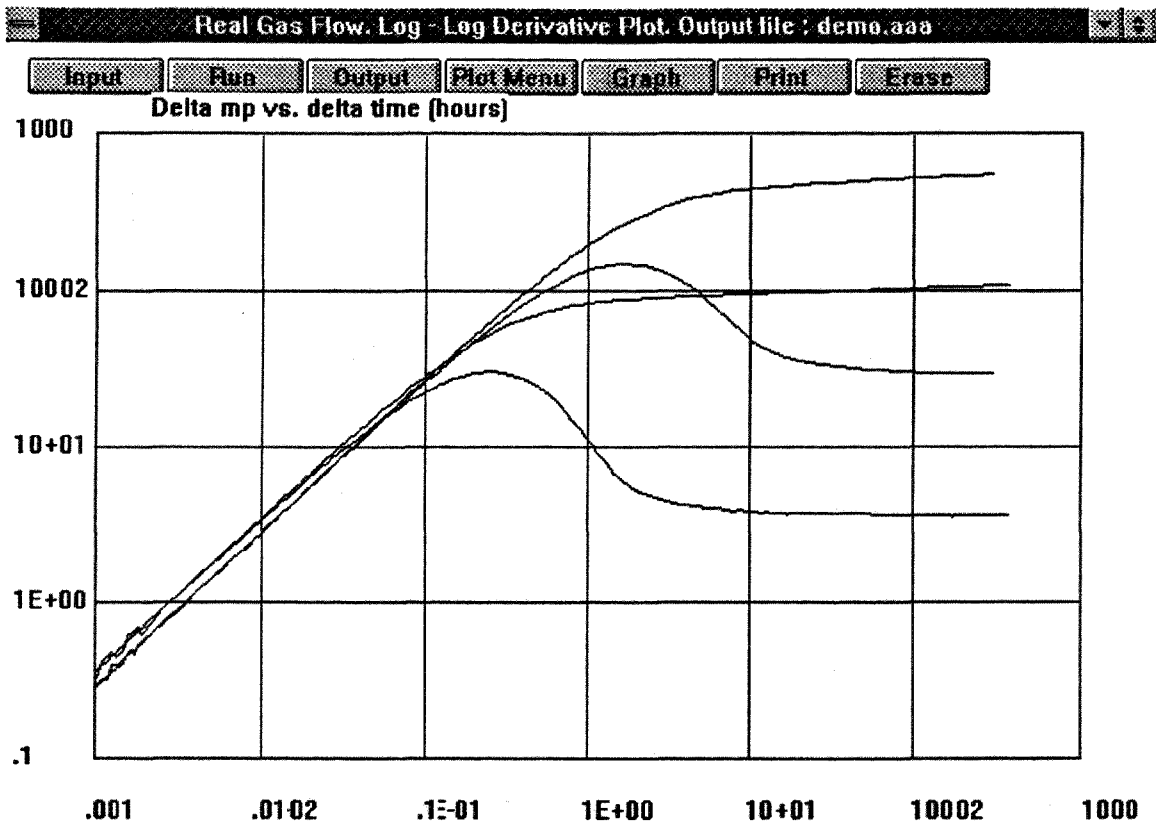


Figure 4.10. Superposition of Log-Log Plots From Two Different Output Files

4.5.2.3 Using the Cursor Utility

Once a plot is on the screen, clicking once with the mouse anywhere inside of the plot area will draw a cursor. Whenever the mouse is clicked, the cursor will reposition itself at the very tip of the arrow. Holding the left button of the mouse down, the cursor can be dragged on the screen.

As soon as the cursor appears on the screen, the x and y coordinates of the intersection between the cursor and the plotted curve are printed on the upper right side of the screen. For example, on Fig. 4.11, the cursor shows the maximum of the derivative curve at delta(mp)=30.03 psia^2/cp and time=0.23 hours.

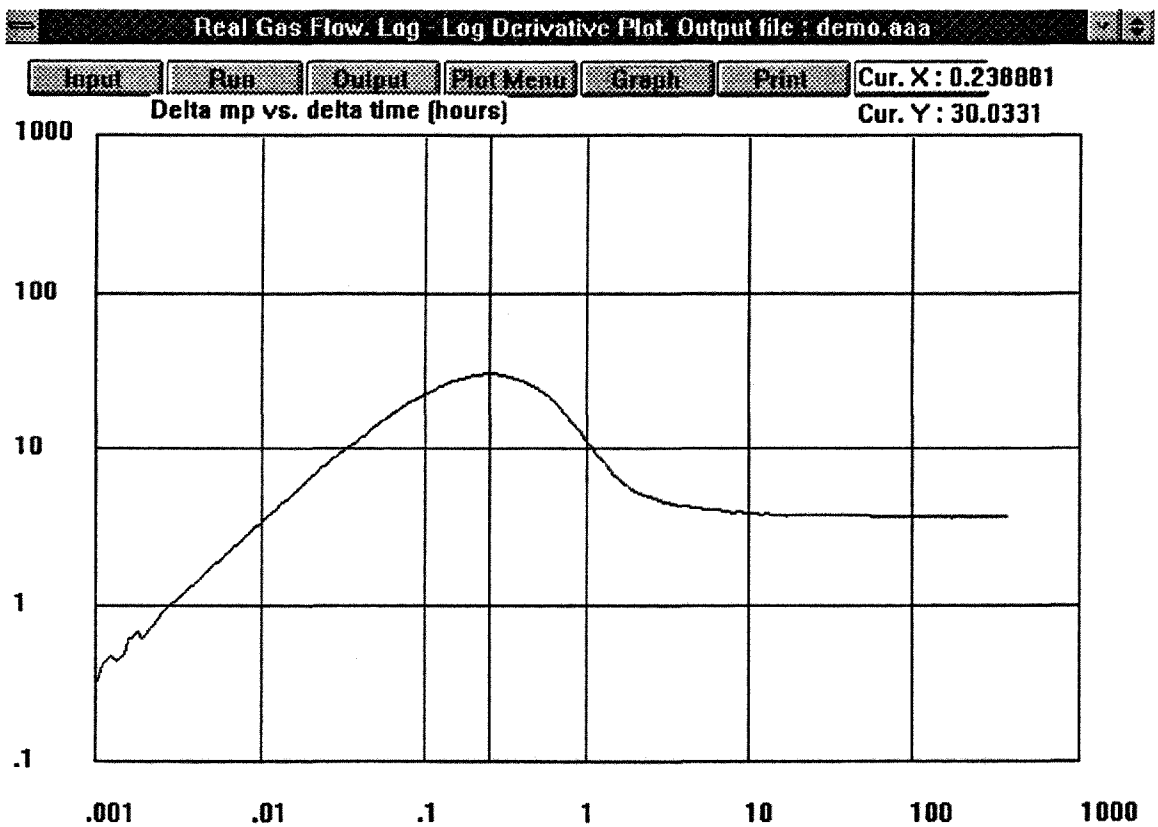


Figure 4.11. Using the Cursor Utility to Pick Points on a Graph

This cursor allows a user to compute the slopes of straight lines in a rapid fashion. When more than one curve is plotted at the same time, the cursor will show coordinates relevant to the last curve plotted on the screen. It is therefore important to remember which one that was!

4.6 Special Features

This section describes two features available with RGF. First, the procedure to compare field data and computed data is presented. The remaining sections introduce the generation of type curves for use with other well test analysis programs such as AUTOMATE.

4.6.1 Using Field Data

An overview of the process of comparing field data and computed data from a simulation follows. A user creates a data file of time and pressure data from field measurements in the working directory. This data file will have a name, and an extension .PRS. The name will be listed in the main file. RGF will read NAME.PRS and convert it into a graphic file: NAME.FLD ready to be graphed.("FLD" for "field"). It is important to declare the name of the field data file in the main file each time a run is made in order for RGF to check that all data will be plotted with the same scale.

4.6.1.1 Setting Up the Field Data

A pressure file must be arranged in pairs of numbers, one pair per line, with time in hours and a pressure in psia. Again, free format is used. We require that the file contains no more than 500 lines in order to limit computing and plotting time. A typical part of a file for a drawdown would look like:

```
0.01    4507
0.05    4505
0.1     4490
0.2     4485
0.8     4480
1.0     4473
10      4460
100     4450
....
```

For users familiar with AUTOMATE, the same pressure files can be used for both programs. However RGF does not require a pressure point at time zero. A user should be certain to input an accurate initial pressure value in the main file before the simulation. Writing the pressure file, it is very important not to leave blank lines at the end of the file as RGF will read them as pairs of zeros.

When a pressure file has been provided to compare field and computed data, a message appears in the DOS window at the end of the simulation telling the user that the field data is being converted in order to be plotted.

4.6.1.2 Plotting Field Data

All field data files to be plotted will have an extension .FLD. These files can be handled the same way as the output files from a simulation. To select a file, go to OUTPUT and make the selection in the working directory. Then use PLOT MENU, GRAPH, and ERASE, the same way as for all other files. For instance, field data and simulated data can be superposed as on Fig. 4.12.

Unlike simulated data that is graphed in blue as a continuous function, field data will be shown in green or red in large points as a discrete function.

A second difference with the way field data is handled by RGF lies in the computation of the derivative. As presented in Horne (1990), in order to reduce the noise in the shape of this derivative, only data points that are separated by at least 0.2 of a log cycle are used, rather than points that are immediately adjacent.

The derivative can be written as:

$$t\left[\frac{\partial P}{\partial t}\right]_i = \left[\frac{\partial P}{\partial \ln t}\right]_i$$

$$= \frac{\ln(t_i/t_{i-k})\Delta P_{i+j}}{\ln(t_{i+j}/t_i)\ln(t_{i+j}/t_{i-k})} + \frac{\ln(t_{i+j}t_{i-j}/t_i^2)\Delta P_i}{\ln(t_{i+j}/t_i)\ln(t_i/t_{i-k})} - \frac{\ln(t_{i+j}/t_i)\Delta P_{i-k}}{\ln(t_i/t_{i-k})\ln(t_{i+j}/t_{i-k})} \quad (4.6)$$

$$\ln t_{i+j} - \ln t_i \geq 0.2$$

$$\ln t_i - \ln t_{i-k} \geq 0.2$$

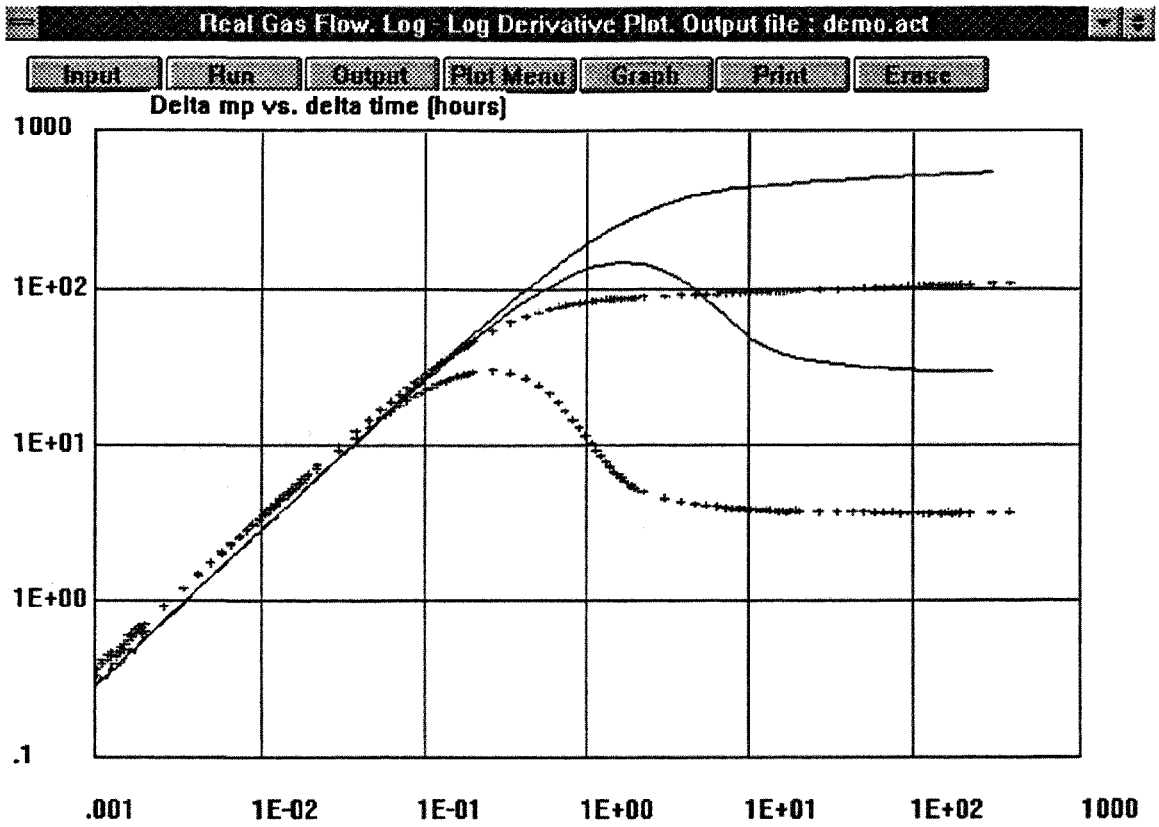


Figure 4.12. Superposition of Field and Synthetic Data

4.6.2 Creating Type Curves

The following procedure is based on the use of the AUTOMATE Well Test Analysis package, but similar software that uses user-installable type curves could have been used. A useful complement to this section is the AUTOMATE-II user manual (any version), and especially Appendix D: User installed type curves.

4.6.2.1 Background

AUTOMATE allows a user to enter their own type curves for plotting and/or matching. To do so, the type curve must be digitized in a standard format file. This is the task that Real Gas Flow will perform. The name of this file must be included in the message file: DIGITIZE.MSG that is inside the AUTOMATE working directory. Modifying DIGITIZE.MSG can be achieved under DOS with any editor.

The format for the type curve input data file must be rigorously observed. This requires no action by the user since RGF will write the entire file. The correct format is nevertheless presented in detail in the AUTOMATE manual, in case the user decides to make custom changes. In addition to the digitized type curve, AUTOMATE requires a minimum of 12 parameters related to the scales and legends of the plots.

To use the installed type curves, CUSTOM CURVES must be selected inside the AUTOMATE menu, followed by DIGITIZED TYPE CURVES. An automated match can also be performed by selecting DIGITIZED TYPE CURVES from the list of available models.

4.6.2.2 Creating One Type Curve

As presented in section 4.3.1.3, three entries related to a type curve can be made in the main file before a simulation is run:

"0" if no type curve is desired
"1" for a standard p_{pD} versus t_D plot
"2" for a storage p_{pD} versus t_D/C_D plot.

In all cases, both the pseudopressure and the derivative curves will be provided. If a simulation is run from TEST1.MAI, the type curve file will be: TEST1.MSG. Additionally, the curves will be marked by their respective value of CD.e^2s at their end, C_D being defined in Eq. 2.16.

Again, for one type curve only, a user does not need to know the content of the type curve file NAME.MSG. But plotting field data against a unique type curve is

definitely not enough to analyze a test fully. It nevertheless gives a user a good idea of how the field and the simulated data compare.

4.6.2.3 Building a Library of Type Curves

This task is more difficult since it requires a user to "paste" type curve files together. Therefore, a good knowledge and understanding of the content of these files is required. See the AUTOMATE manual. The physical task of putting files together is simple and not time consuming. It must be done in DOS. Figure 4.13 shows a simple library of five type curves. The type curve data file for this particular pair of curves is presented in Table 4.3 and should be studied.

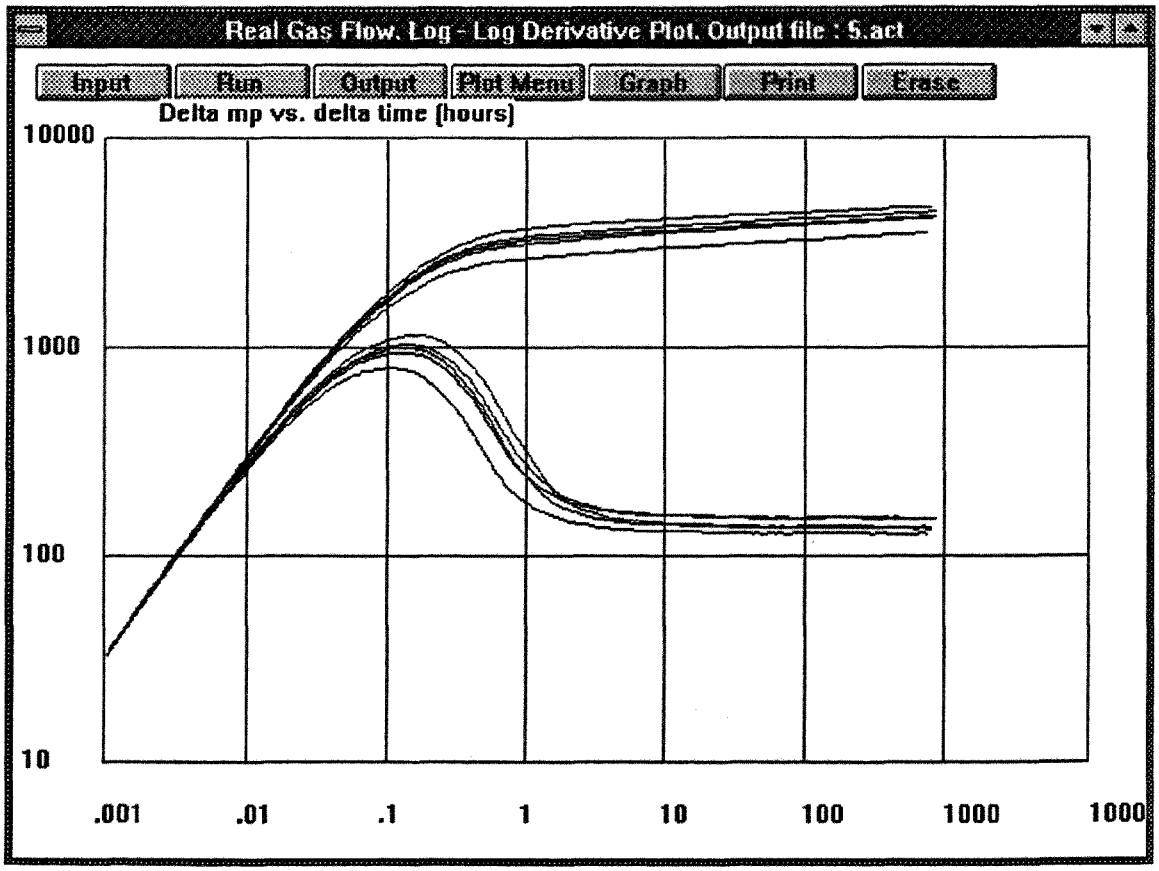


Figure 4.13. Example of Combination of Five Type Curves to Build a Library

A type curve file with derivative values has the following structure:

Type Curve From Real Gas Flow
tD/cD
mpD

2
0
6
0
4
0
0
5
55

0.1400	0.1180	0.1178	0.1375	0.1378	0.1372
0.3000	0.2708	0.2701	0.2904	0.2915	0.2889
0.4600	0.4194	0.4177	0.4394	0.4417	0.4362
0.6200	0.5640	0.5611	0.5846	0.5887	0.5793
0.7800	0.7049	0.7007	0.7265	0.7325	0.7186
0.9400	0.8424	0.8366	0.8652	0.8734	0.8543
2.0000	1.5410	1.5230	1.7090	1.7380	1.6710
3.6000	2.6270	2.5810	2.7990	2.8720	2.7070
5.2000	3.5360	3.4570	3.7230	3.8470	3.5680
6.8000	4.3120	4.1980	4.5200	4.6990	4.2990
8.4000	4.9840	4.8350	5.2150	5.4520	4.9260
10.0000	5.5720	5.3880	5.8290	6.1230	5.4710
26.0000	8.6460	8.2000	9.3620	10.1700	8.4310
34.0000	9.5540	9.0000	10.2700	11.2700	9.1330
42.0000	10.1800	9.5370	10.9100	12.0700	9.6040
50.0000	10.6200	9.9140	11.3700	12.6700	9.9330
58.0000	10.9400	10.1900	11.7100	13.1200	10.1700
66.0000	11.1900	10.3900	11.9600	13.4600	10.3500
74.0000	11.3800	10.5500	12.1600	13.7400	10.4900
82.0000	11.5200	10.6800	12.3200	13.9600	10.6000
90.0000	11.6500	10.7800	12.4500	14.1300	10.7000
98.0000	11.7500	10.8700	12.5500	14.2800	10.7700
160.0000	12.0800	11.1700	13.0000	14.8600	11.1300
240.0000	12.4000	11.4700	13.2800	15.1900	11.3900
320.0000	12.6000	11.6600	13.4600	15.3800	11.5600
400.0000	12.7400	11.8000	13.6000	15.5200	11.6900
480.0000	12.8500	11.9100	13.7000	15.6300	11.7900
560.0000	12.9400	12.0000	13.7900	15.7200	11.8700
640.0000	13.0200	12.0700	13.8700	15.8000	11.9500
720.0000	13.0900	12.1400	13.9300	15.8700	12.0100
800.0000	13.1500	12.2000	13.9900	15.9300	12.0700
880.0000	13.2000	12.2500	14.0400	15.9800	12.1200
960.0000	13.2500	12.3000	14.0900	16.0300	12.1700
1400.0000	13.3800	12.4300	14.2900	16.2400	12.3600
2200.0000	13.6600	12.7000	14.5300	16.4800	12.6000
3000.0000	13.8400	12.8800	14.7000	16.6500	12.7600
3800.0000	13.9700	13.0100	14.8300	16.7800	12.8900
4600.0000	14.0800	13.1200	14.9300	16.8800	12.9900
5400.0000	14.1700	13.2100	15.0100	16.9700	13.0700
6200.0000	14.2400	13.2800	15.0900	17.0400	13.1400
7000.0000	14.3100	13.3500	15.1500	17.1100	13.2100
7800.0000	14.3700	13.4100	15.2100	17.1600	13.2600
8600.0000	14.4200	13.4600	15.2600	17.2200	13.3200
9400.0000	14.4700	13.5100	15.3100	17.2600	13.3600
12000.0000	14.5100	13.5500	15.4300	17.3900	13.4900
20000.0000	14.8200	13.8500	15.7000	17.6600	13.7500
28000.0000	15.0100	14.0500	15.8800	17.8400	13.9200
36000.0000	15.1500	14.1900	16.0100	17.9800	14.0500
44000.0000	15.2700	14.3000	16.1100	18.0800	14.1600
52000.0000	15.3600	14.3900	16.2000	18.1700	14.2400
60000.0000	15.4400	14.4700	16.2800	18.2500	14.3200
68000.0000	15.5000	14.5400	16.3400	18.3100	14.3800
76000.0000	15.5600	14.6000	16.4000	18.3700	14.4400
84000.0000	15.6200	14.6500	16.4600	18.4300	14.4900
92000.0000	15.6700	14.7000	16.5000	18.4800	14.5400
100000.0000	15.7100	14.7400	16.5500	18.5200	14.5800

0 10 5
55

0.1400	0.1165	0.1162	0.1358	0.1362	0.1358
0.3000	0.2639	0.2626	0.2831	0.2850	0.2804
0.4600	0.4032	0.4004	0.4229	0.4273	0.4174
0.6200	0.5354	0.5308	0.5567	0.5636	0.5475
0.7800	0.6615	0.6542	0.6841	0.6944	0.6707
0.9400	0.7816	0.7720	0.8062	0.8204	0.7879
2.0000	1.3540	1.3260	1.4980	1.5450	1.4390
3.6000	2.1120	2.0460	2.2600	2.3690	2.1270
5.2000	2.6280	2.5220	2.7930	2.9670	2.5810
6.8000	2.9840	2.8380	3.1690	3.4080	2.8820
8.4000	3.2290	3.0470	3.4350	3.7360	3.0770
10.0000	3.3910	3.1770	3.6200	3.9770	3.1960
26.0000	3.4090	3.0410	3.6570	4.3670	2.8880
34.0000	3.0390	2.6520	3.2700	4.0450	2.4700
42.0000	2.6500	2.2720	2.8640	3.6560	2.0890
50.0000	2.2950	1.9430	2.4910	3.2650	1.7700
58.0000	1.9900	1.6720	2.1650	2.9010	1.5160
66.0000	1.7340	1.4540	1.8890	2.5730	1.3180
74.0000	1.5260	1.2820	1.6600	2.2850	1.1650
82.0000	1.3560	1.1470	1.4720	2.0360	1.0460
90.0000	1.2200	1.0410	1.3180	1.8220	0.9560
98.0000	1.1100	0.9583	1.1940	1.6400	0.8863
160.0000	0.8314	0.7595	0.8033	0.9874	0.6921
240.0000	0.6722	0.6457	0.6660	0.7332	0.6201
320.0000	0.6223	0.6071	0.6204	0.6561	0.5916
400.0000	0.5982	0.5870	0.5976	0.6231	0.5750
480.0000	0.5848	0.5743	0.5841	0.6045	0.5648
560.0000	0.5751	0.5663	0.5742	0.5922	0.5575
640.0000	0.5678	0.5598	0.5680	0.5831	0.5511
720.0000	0.5629	0.5549	0.5623	0.5766	0.5483
800.0000	0.5600	0.5540	0.5586	0.5749	0.5451
880.0000	0.5593	0.5506	0.5580	0.5688	0.5439
960.0000	0.5542	0.5481	0.5548	0.5663	0.5421
1400.0000	0.5523	0.5474	0.5522	0.5629	0.5412
2200.0000	0.5494	0.5452	0.5476	0.5569	0.5389
3000.0000	0.5442	0.5414	0.5422	0.5515	0.5339
3800.0000	0.5404	0.5365	0.5395	0.5475	0.5316
4600.0000	0.5392	0.5354	0.5386	0.5447	0.5290
5400.0000	0.5381	0.5328	0.5359	0.5439	0.5299
6200.0000	0.5363	0.5318	0.5348	0.5405	0.5273
7000.0000	0.5347	0.5319	0.5322	0.5406	0.5252
7800.0000	0.5337	0.5350	0.5361	0.5411	0.5282
8600.0000	0.5343	0.5350	0.5358	0.5414	0.5268
9400.0000	0.5318	0.5308	0.5308	0.5395	0.5238
12000.0000	0.5341	0.5270	0.5355	0.5427	0.5270
20000.0000	0.5393	0.5357	0.5383	0.5448	0.5302
28000.0000	0.5391	0.5357	0.5363	0.5438	0.5297
36000.0000	0.5361	0.5329	0.5343	0.5419	0.5269
44000.0000	0.5360	0.5315	0.5338	0.5413	0.5287
52000.0000	0.5355	0.5311	0.5346	0.5392	0.5233
60000.0000	0.5316	0.5275	0.5334	0.5406	0.5221
68000.0000	0.5319	0.5282	0.5308	0.5375	0.5214
76000.0000	0.5311	0.5272	0.5279	0.5405	0.5233
84000.0000	0.5301	0.5261	0.5334	0.5381	0.5221
92000.0000	0.5297	0.5284	0.5260	0.5382	0.5264
100000.0000	0.5318	0.5253	0.5300	0.5346	0.5237

0 10 5

Table 4.3. Data File for Library of Type Curves Generated by RGF

<line 1 to 12> :	12 parameters
<line 13 to n+13>:	Data points for the curves: t_D, $p_{pD}1$, $p_{pD}2$, etc.. All curves must have the same number of points and share the same time values.
<next lines> :	Parameters related to the curve labels.
<next n lines> :	Data point for the derivative curves. They must have the same number of points as the p_{pD} curves.
<last lines> :	Parameters related to derivative curve labels.

Under DOS, a short program is required to read the data points from each individual type curve file and to rewrite them as in the AUTOMATE manual.. All other parameters were edited manually when needed.

Going into AUTOMATE, field data can be plotted with the library of type curves by performing a manual match with the "type curve" command, after selecting "digitized type curves". A nonlinear regression can also be run with any library of type curves using the "automatch" function of AUTOMATE. The result can be displayed with the function "plot match".

4.7 Printing

There are two ways to print from RGF. It is possible to print a graph of the data using the PRINT button provided by RGF, or a hard copy of the entire screen can be made at any time using the CLIPBOARD utility of Windows.

4.7.1 Printing Directly

Printing directly from RGF to either a postscript file or to a printer is possible and easy. Clicking on the PRINT button in the main menu bar will print the content of the plotting box on the screen. PRINT doesn't send a hard copy of the screen to the printing device. Therefore, it does not print the input parameter window that displays main files for instance. PRINT sends a graph to the printer the same way the graph was sent to the screen. Therefore, only the last plotted curve will be printed. To print a superposition of curves (when comparing field and synthetic data for instance), or to print a hard copy of the screen, a Windows accessory called CLIPBOARD must be used.

4.7.2 Printing with CLIPBOARD

The CLIPBOARD utility is embedded into Windows. CLIPBOARD does not need to be opened or activated in order to be used, as long as it is present in the Windows directory. To produce a hard copy of any screen, press the PRINTSCREEN button on the keyboard (sometime PTSCR). The screen is stored into the CLIPBOARD. It can be pasted into any of the following programs: MS Word, Excel, or PowerPoint using the EDIT menu. At this point it can be sent to a printer.

4.8 Leaving REAL GAS FLOW

RGF can be exited at any time by double-clicking on the upper left square button. Two relevant issues when leaving the program are: Saving important files, and cleaning the working directory.

All files created by RGF are automatically saved in the working directory. After using RGF, it is recommended to save batches of test results under related names in specific directories or subdirectories. Remember that when reusung old tests, the main input files (as for the flow rate files and field data files) must be put back into the working directory.

Before leaving the working directory, one should remove unnecessary old input and output files. All files *.AAA and INPUT.FIL which are temporary files for RGF can be removed also.

4.9 Computing Aids

It is recommended to open a FILE MANAGER window in RGF's background in order to copy files between directories and sub-directories. This is an easy way to insure that all files needed are in the working directory before running a simulation.

NOTEPAD, one of the Windows accessories, can be used as an editor. All NOTEPAD files are also text files. The objective is to have easy access under Windows to files with the following extension:

*.FLO Flow rate files
*.PRS Pressure files for field data
*.MSG Type curve data files

in order to edit them.

To do so, a user must start the FILE MANAGER, go into the Windows directory and double-click on the file "win.ini". A file called NOTEPAD-WIN.INI will appear. In the [extensions] section, the following lines should be inserted:

flo=notepad.exe ^.flo
msg=notepad.exe ^.msg
prs=notepad.prs ^.prs

If win.ini is saved, and the computer rebooted, any files with the extensions specified will be accessible for editing as text files under FILE MANAGER. The next section shows an example of the use of RGF as a tool for well test analysis.

5. Using RGF as a Well Test Interpretation Tool

A major application of the software is to design well tests. Another application is to have an interactive tool with which sensitivity analysis can be performed. But RGF can also be used as a tool for interpreting field test data, especially when software based on traditional models shows poor confidence limits. For a high velocity and high pressure drop well test, changing wellbore storage, changing turbulence coefficient, or a combination, can cause a poor interpretation. Another cause of poor interpretation may result from a poor estimation of the initial pressure in the reservoir before the start of flow.

In this example, an associate created a set of synthetic field data with RGF, and submitted a pressure history file, a rate data file, and some additional data concerning the reservoir and the well completion to an associate (the analyst). This should simulate the analysis of a field test, where permeability and skin would be the unknowns.

The objective was to determine whether an analyst could find the permeability and skin by plot-fitting the synthetic field data and a succession of data simulated by trial and error with the help of RGF to find a satisfactory match.

REAL GAS FLOW offers a variety of graphs to choose from. Experience showed that a satisfactory way to perform the match, which was a time consuming task, was to follow these directions:

Plot fit with the log-log graph first.
Plot fit next with the combination log-log and derivative graph.
Check the coherence of the solution with a semi-log plot like MDH.

AUTOMATE (or a similar well test program) was helpful in finding estimated values of permeability and skin effect before starting with RGF.

The case studied was the following:

Flowrate of 10,000 Mcf/D for 1,000 hours or 42 days
Infinite acting case, no external boundary
Damaged region of three feet around the wellbore
Wellbore radius of 0.333 feet
Pay zone of 15 feet
Wellbore Depth of 20,000 feet. We considered a wellbore volume of 10,000 cubic feet.
The porosity was 15%, the gas gravity 0.9, and the reservoir temperature 375 °F.
The initial reservoir pressure before the start of the flow was 8,000 psia.

Part of the synthetic field data produced by RGF is presented in table 5.1.

AUTOMATE gives the following solution for this test:

Permeability k=12 mD +/- 3%.
Total skin = 4.4 +/- 5.5%.

A manual plot fit is a time consuming task, but experience will allow a user to anticipate how to change permeability and skin effect to affect the position of the plots on the screen of REAL GAS FLOW. The fastest way to perform a manual fit is to perform a sensitivity analysis with one parameter at a time; permeability first, then skin effect, then permeability again, and so on.

Time	Pressure
0	8000
0.001	7997
0.0027	7991
0.0044	7986
0.006	7981
0.0077	7976
0.012	7962
0.025	7923
0.04	7886
0.052	7851
0.066	7816
0.08	7783
0.1	7716
0.24	7460
0.37	7264
0.5	7110
0.64	6985
0.77	6881
1.91	6440
3.58	6243
5.24	6174
6.91	6139
8.24	6120
10.8	6095
24	6030
37.5	5996
50.8	5973
64	5956
77.5	5942
224	5865
458	5813
691	5784

Table 5.1. Pressure History for Synthetic Field Data

After a significant trial and error period, we were able to obtain the following match in Fig. 5.1. The data in discrete points is the field data. The simulated data, a continuous line, was obtained by entering into RGF's input parameter window all the information given before, in addition to an estimated permeability of 17 mD and a skin of factor of 3.

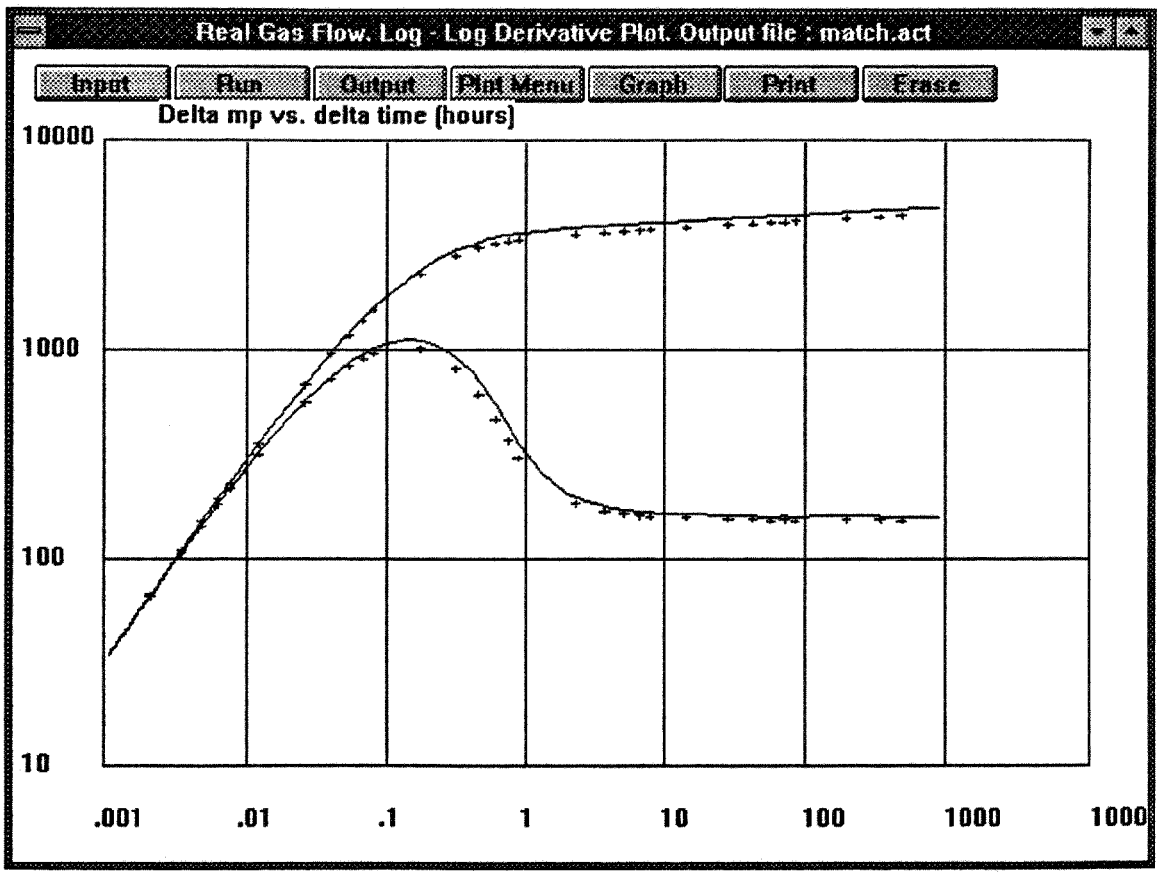


Figure 5.1. Plot-Fitting Field Data and Simulated Data by Trial and Error, Log-Log Plot

Even though the log-log graph shows the best match after a trial and error period of an hour and a half, the semilog plot shows some divergence at late times:

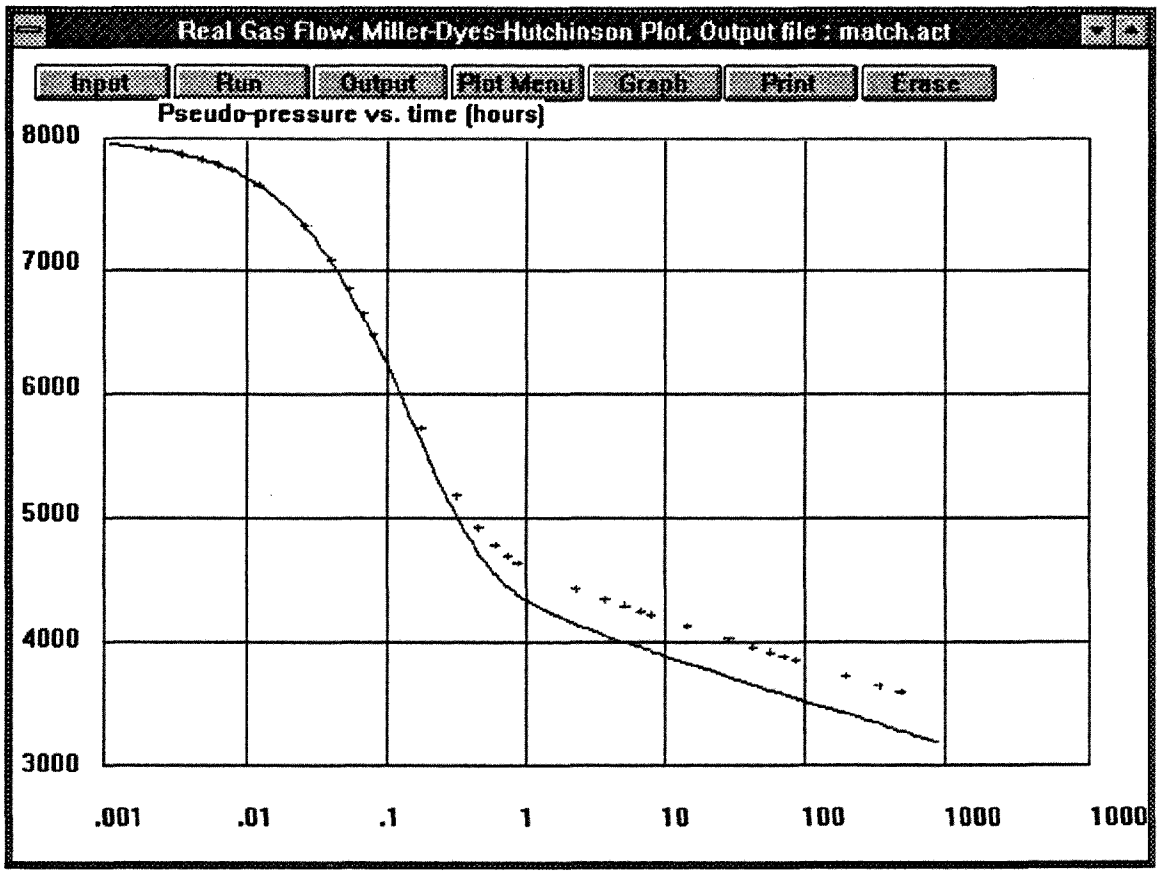


Figure 5.2. Plot-Fitting Field Data and Simulated Data by Trial and Error, Semi-Log Plot

The values that were used in order to create the set of data with RGF in the first place were:

Permeability = 17.5 mD.
Skin damage = 2.8

Our trial and error plot-fit was within 3% of the correct solution for the permeability and 7% for the skin. In this particular case, we were persistent enough to get very close to the solution by the method of trial and error. A process to accelerate the fit or confirm it is to bracket the values of s and k, build a library of type curves covering this range as presented in Section 4.6.2, and use AUTOMATE to perform a nonlinear regression of the field data onto the type curves.

For instance, for values of skin between 0 and 5 and values of permeability between 10 and 20, AUTOMATE will perform an automatic match as on Fig. 5.3. Keeping track of each type curve displayed and knowing its corresponding parameters, the best interpolation can be read directly on the screen.

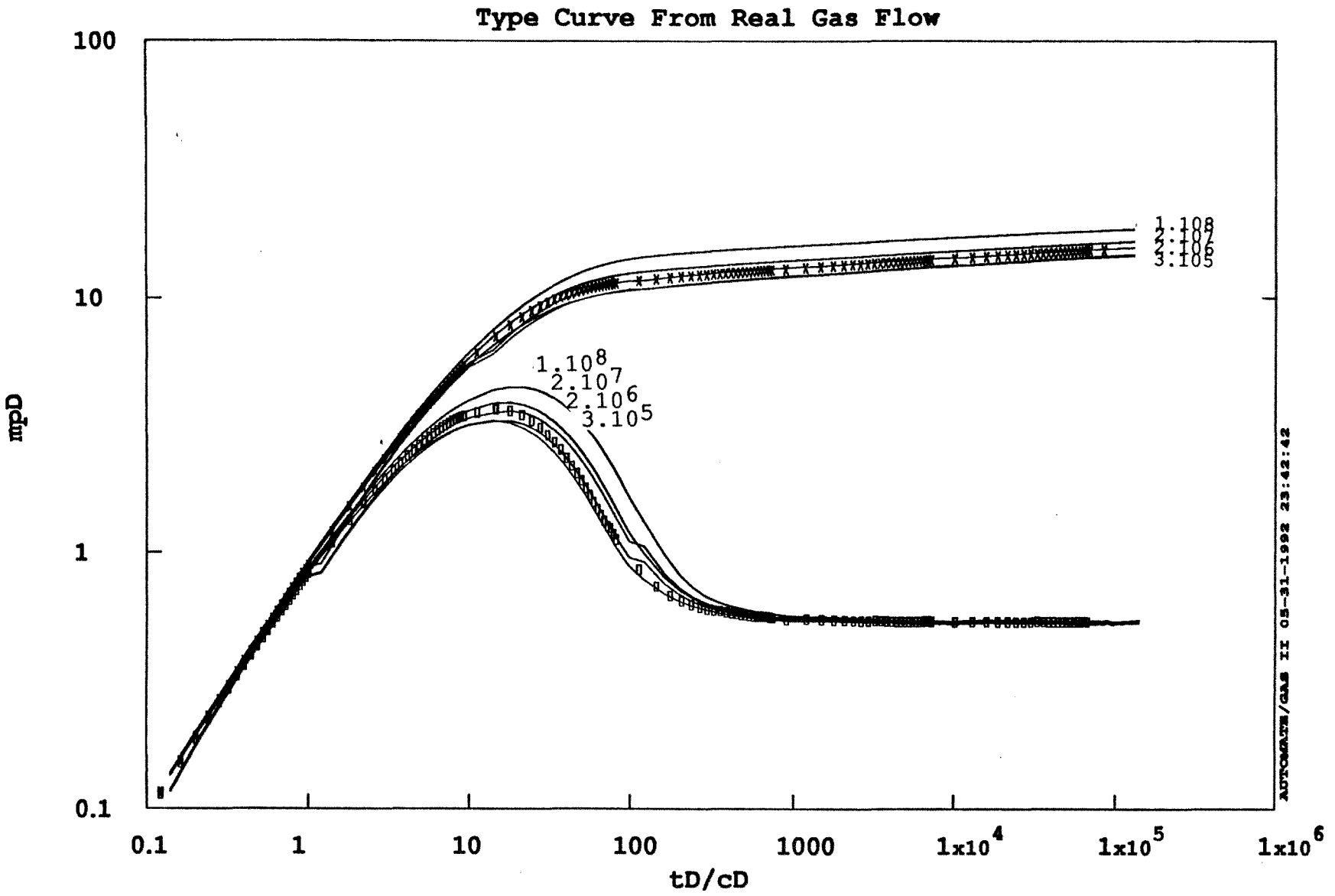


Fig. 5.3. Linear Regression Fit of Synthetic field Data on Generated Type Curves

6. Conclusions

An interactive software program that simulates the flow of real gas through porous media has been prepared. It runs under a graphic environment and allows a user to compare sets of data, perform sensitivity analysis, and display field data. The easy access to data and quick display of graphs on the screen should reduce time waste caused by having such a program on a mainframe computer with access via terminal. Both the installation and the use of the program are easy. A main menu bar controls the major actions. Creating input files or plotting data from old well tests is a matter of seconds.

This program will allow investigation of the effects of changing turbulence factor and changing wellbore storage on gas well tests. The program also can be used in order to design well tests. It has also been shown that this software can be an interpretation tool by plot fitting real data in a trial and error approach. The program will allow a user to work in combination with other software programs that perform non-linear regression on type curves.

Finally, this program can be tailored to work on other specific applications such as the problem of adsorption in geothermal engineering. Further improvements could also be provided to the software in order to consider a large variety of well tests such as buildup, multirate and interference tests.

Nomenclature

C_D	=	dimensionless wellbore storage
c_g	=	gas compressibility, psi^{-1}
h	=	formation thickness, ft
k	=	formation permeability, md
k_1	=	damaged annular region permeability, md
M	=	molecular weight, lb_m/mole
m(p)	=	real gas potential, psi^2/ cp
p	=	pressure, psia
$\bar{p}$	=	average wellbore flowing pressure, psia
p_i	=	initial formation pressure, psia
p_{sc}	=	standard pressure, psia
p_{pn}	=	normalized pseudopressure, psia
p_{pc}	=	pseudocritical pressure, psia
p^*_c	=	corrected critical pressure, psia
p_r	=	reduced pressure, dimensionless
q_{sc}	=	gas production rate, Mcf/D
q_{sf}	=	sandface gas flow rate, Mcf/D
q_{wb}	=	wellbore rate, Mcf/D
r	=	radial distance from wellbore center, ft
r_1	=	damaged annular region radius, ft
r_e	=	external radius, ft
r_w	=	wellbore radius, ft
R	=	universal gas constant
s	=	real skin effect

t	=	time, hrs
T	=	formation temperature, °R
$\overline{T}$	=	average wellbore flowing temperature, °R
T_{sc}	=	standard temperature, °R
Tpc	=	pseudocritical temperature, °R
Tr	=	reduced temperature, dimensionless
T^*_c	=	corrected critical temperature, °R
u	=	macroscopic gas velocity
V_{wb}	=	wellbore volume, cu ft
z	=	real gas law deviation factor
α	=	diffusivity parameter
β	=	turbulence parameter, md^{-1}
γ_g	=	specific gas gravity (to air)
δ_1	=	Darcy's law correction factor for damaged region
δ_r	=	radial Darcy's law correction factor
v	=	dimensionless logarithm ratio
$\mu(p)$	=	pressure dependent gas viscosity, cp
ϕ	=	porosity, fraction
$\rho(p)$	=	pressure dependent gas density, lb_m/cu ft

Subscripts:

1	=	damaged around the well
D	=	dimensionless
e	=	external

g	=	gas
i	=	initial
r	=	distance r from the wellbore center
sc	=	standard condition
sf	=	sandface
w, wb	=	wellbore

References

Agarwal, R.G., Al-Hussainy, R., and Ramey, H.J., Jr.: "An Investigation of Wellbore Storage and Skin Effect in Unsteady Liquid Flow: I. Analytical Treatment.", Soc. Pet. Eng. J. (Sept. 1970), 279.

Al-Hussainy, R., Ramey, H.J., Jr. and Crawford, P.B.: "The Flow of Real Gases Through Porous Media", J. Pet. Tech., (May 1966a), 624.

Al-Hussainy, R. and Ramey, H.J., Jr.: "Application of Real Gas Flow Theory to Well Testing and Deliverability Forecasting", J. Pet. Tech., (May 1966b), 637.

Aronofsky, J.S., and Jenkins, R.: "Unsteady Flow of Gas Through Porous Media-One Dimensional Case", Proceedings, 1st U.S. Nat. Cong. Appl. Mech. (1952), 763.

Aronofsky, J. S., Jenkins, R.: "A Simplified Analysis of Unsteady Real Gas Flow", Trans. AIME (1954) 201, 149.

Bruce, G.H., Peaceman, D.W., and Rachford, H.H., Jr.: "Calculations of Unsteady-State Gas Flow Through Porous Media", Trans., AIME (1953), 198, 79.

Carter, R.D.: "Solutions of Unsteady-State Radial Gas Flow", Trans., AIME (1962) 225, 549.

Carter, R.D.: Supplemental Appendix to "Determination of Stabilized Gas Well Performance from Short Flow Tests", ADI Doc. No. 7471, Library of Congress, Washington, D.C. (1963a).

Carter, R.D. Miller, S.C., and Riley, H.G.: "Determination of Stabilized Gas Well Performance from a Short Flow Test", Trans., AIME (1963b) 228, 651.

Cornell, D. and Katz, D.L.: "Flow of Gases Through Consolidated Porous Media", Ind. and Eng. Chem. (1953) 45, No. 2, 145.

Couri, F.R.: "Effect of Stress Sensitive Permeability, Skin and Wellbore Storage on High Velocity Flow in Gas Well Test", Master's Thesis, Stanford University, California, 1987.

Cullender, M.H. and Smith, R.V.:"Practical Solution of Gas Flow Equations for Wells and Pipelines with Large Temperature Gradient," Trans., AIME (1956), 207, 281.

Dranchuck, P.M., Purvis, R.A., and Robinson, D.B.: Computer Calculations of Natural Gas Compressibility Factors Using The Standing and Katz Correlation, Institute of Petroleum Technical Series, No. IP-74-008, (1974).

Earlougher, R.C., Jr., and Kersch, K.M.: "Analysis of Short-Time Transient Test Data by Type-Curve Matching," J. Pet. Tech. (1974), 793.

Eilerts, C.K.: "Integration of Partial Differential Equation for Transient Linear Flow of Gas-Condensate Fluids in Porous Structures", Trans., AIME (1964) 231, 291.

Eilerts, C.K., Sumner, E.F., and Potts, N.L.: "Integration of Partial Differential Equation for Transient Linear Flow of Gas-Condensate Fluids in Porous Structures", Trans., AIME (1965) 234, 141.

Firoozabadi, A., and Katz, D.L.: "An Analysis of High Velocity Gas Flow Through Porous Media," J. Pet. Tech. (Feb. 1979), 211.

Fligelman, H.: "Drawdown and Interference Test Analysis for Gas Wells with Wellbore Storage, Damage, and Non-Laminar Flow Effects", Ph.D. Dissertation, Stanford University, California, 1981.

Fligelman, H., Cinco-Ley, H., Ramey, H.J., Jr., Braester, C., and Couri, F.: "Pressure-Drawdown Test Analysis of a Gas Well - Application of New Correlations," SPE Formation Evaluation, (Sept. 1989), 406.

Forchheimer, P.H.: "Wasserbewegung durch Boden," z. Ver Deutsch. Ing. (1901), 45, 1781.

Hegeman, S., Hallford, D.L. and Joseph, J.A.: "Well Test Analysis with Changing Wellbore Storage", paper 21829, presented at the SPE Rocky Mountain Regional Meeting, Denver, CO, Apr. 15-17, 1991.

Horne, R.N.: Modern Well Test Analysis: A Computer-Aided Approach, Petroway Inc., Palo Alto, (1990).

Houpeurt, A.: "On the Flow of Gases in Porous Media," Revue de l'Institut Francais du Petrole, (1959), XIV, No. 11, 1468.

Hubbert, M.: "Darcy's Law and the Field Equations of Flow of Underground Fluids," Trans., AIME (1956), 207, 222.

Hurst, W.: "Establishment of the Skin Effects and its Impediment to Fluid Flow into a Wellbore," Pet. Eng., (Oct. 1953), 25 , B-6.

Jenkins, R. and Aronofsky, J.S.: "Unsteady Radial Flow of Gas Through Porous Media", J.Appl. Mech., ASME (June 1953), 20, No.2, 210.

Katz, D.L., and Coats, K.H.: Underground Storage of Fluids, Ann Harbor: Ulrich's Books, Inc. (1968).

Katz, D.L., Cornell, D., Kobayashi, R., Poettman, F.H., Vary, J.A., Elenbaas, J.R., and Weinaug, C.F.: Handbook of Natural Gas Engineering, New York: McGraw-Hill Book Co., Inc. (1959).

Lee, A.L., Gonzales, M.H., and Eakin, B.E.: "The Viscosity of Natural Gases", J. Pet.Tech. , (August 1966), 997.

Matthews, C.S.: "Analysis of Pressure Buildup and Flow Test Data," J. Pet. Tech. , (Sept. 1961), 862.

Matthews, C.S., and Russell, D.G.: "Pressure Buildup and Flow Tests in Wells," Soc. Pet. Eng. Monograph Series (1967), 1.

Meunier, D., Kabir, C.S. and Wittmann, M.J.: "Gas Well Test Analysis: Use of Normalized Pressure and Time Functions", SPE Formation Evaluation, (Dec. 1987), 629.

Ramey, H.J., Jr.: "Non-Darcy Flow and Wellbore Storage Effects in Pressure Buildup and Drawdown of Gas Wells", J. Pet. Tech., (Feb. 1965), 223.

Ramey, H.J., Jr.: "Short -time Well-Test Data Interpretation in Presence of Skin Effect and Wellbore Storage," J. Pet. Tech. , (Jan. 1970), 97.

Russell, D.C., Goodrich, J.H., Perry, G.E. and Bruskotter, J.F.: "Methods for Predicting Gas Well Performance", Trans., AIME (1966) 237, 99.

Smith, R.V.: "Unsteady-State Gas Flow into Gas Wells", Trans., AIME (1961), 222, 1151.

Standing, M.B.: Volumetric and Phase Behavior of Oil Field Hydrocarbon Systems , SPE reprint, 8th edition, (1977).

Swift, G.W., and Kiel, O.G.: "The Prediction of Performance Including the Effect of Non-Darcy Flow", Trans., AIME (1962) 225, 791.

Tek, M.R., Coats, K.H., and Katz, D.L.: "The Effect of Turbulence on the Flow of Natural Gases Through Porous Reservoirs", Trans., AIME (1962), 225, 799.

Van Everdingen, A.F.: "The Skin Effect and its Influence on the Productive Capacity of a Well.", Trans., AIME (1953), 198, 171.

Van Everdingen, A.F., and Hurst, W.: "The Application of the Laplace Transformation to Flow Problems in Reservoir Problems," Trans., AIME (1949), 186, 305.

Wattenbarger, R.A.: "Effects of Turbulence, Wellbore Damage, Wellbore Storage and Vertical Fractures on Gas Well Testing," Ph.D. Dissertation, Stanford University, Stanford, California, 1967.

Wichert, E., and Aziz, K.: "Calculating Z's for Sour Gases", Hydrocarbon Processing, (May 1972), 51.

Appendix A.

Computation of Gas Properties

The pseudocritical pressure p_{pc} in psia and temperature Tpc in degrees Rankine are determined from the gas gravity, using the Standing correlations.

For California gases:

$$p_{pc} = 677 + 15\gamma_g - 37.5\gamma_g^2 \tag{A.1}$$

$$T_{pc} = 168 + 325\gamma_g - 12.5\gamma_g^2 \tag{A.2}$$

For Condensate gases:

$$p_{pc} = 706 - 51.7\gamma_g - 11.1\gamma_g^2 \tag{A.3}$$

$$T_{pc} = 187 + 330\gamma_g - 71.5\gamma_g^2 \tag{A.4}$$

If the gas contains impurities, corrections are made using, Wichert and Aziz (1972):

$$e = 120(y_{CO_2} + y_{H_2S})^{0.9} - 120(y_{CO_2} + y_{H_2S})^{1.6} + 15(y_{H_2S}{}^{0.5} - y_{H_2S}{}^{4}) \tag{A.5}$$

$$p_c^* = \frac{p_c(T_c - e)}{T_c + y_{H_2S}(1 - y_{H_2S})e} \tag{A.6}$$

$$T_c^* = T_c - e \tag{A.7}$$

The reduced pressure and temperature are:

$$p_r = \frac{p}{p_{pc}} \tag{A.8}$$

$$T_r = \frac{T_R}{T_{Pc}} \tag{A.9}$$

The reduced density is computed iteratively using Newton's method. Then the z factor is estimated using the Dranchuck, et al. (1974) procedure:

$$\rho_r^{k+1} = \rho_r^k - \frac{f(\rho_r^k)}{f'(\rho_r^k)} \tag{A.10}$$

where:

$$f(\rho_r) = a\rho_r^6 + b\rho_r^3 + c\rho_r^2 + d\rho_r + e\rho_r^3(1+f\rho_r^2)\exp[-f\rho_r^2] - g \tag{A.11}$$

$$f'(\rho_r) = 6a\rho_r^5 + 3b\rho_r^2 + 2c\rho_r + d + e\rho_r^2(3+f\rho_r^2[3-2f\rho_r^2])\exp[-f\rho_r^2] \tag{A.12}$$

where:

$$a = 0.06423$$
$$b = 0.5353T_r - 0.6123$$
$$c = 0.3151T_r - 1.0467 - 0.5783 / T_r^2$$
$$d = Tr$$
$$e = 0.6816 / T_r^2$$
$$f = 0.6845$$
$$g = 0.27\rho_r$$
$$\rho_r^0 = 0.27\rho_r / T_r$$

and:

$$z = \frac{0.27\rho_r}{\rho_r T_r} \tag{A.13}$$

Gas compressibility is computed from:

$$c_g = \frac{1}{p_c p_r \left[1 + \frac{\rho_r}{z}\frac{\partial z}{\partial \rho_r}\right]} \tag{A.14}$$

where:

$$\frac{\partial z}{\partial \rho_r} = \frac{1}{\rho_r T_r}\left[5a\rho_r^5 + 2b\rho_r^2 + c\rho_r + 2e\rho_r^2(1 + f\rho_r^2 - f^2\rho_r^4)\exp[-f\rho_r^2]\right] \tag{A.15}$$

The viscosity is computed using the Lee et al. (1966) procedure which is approximate for sour gases since the density is corrected for impurities:

$$\mu_g = K.10^{-4}.\exp(X\rho^y) \tag{A.16}$$

where:

$$K = \frac{(9.4 + 0.02M)T^{1.5}}{209 + 19M + T}$$

$$X = 3.5 + \frac{986}{T} + 0.01M$$

$$y = 2.4 - 0.2X$$

The final table concerns the term $\frac{\partial[\bar{p}/\bar{z}(p)]}{\partial[p/z(p)]}$ used to calculate changing wellbore storage as defined in Eqs. 2.9 and 2.16. This task is performed by the subroutine WELLPRO based on the Cullender and Smith (1956) method to compute the static bottom hole pressure using:

$$0.01875\gamma_g H = \int_{p_t}^{p} \frac{Tz(p')}{p'}dp' \tag{A.17}$$

where H is the length of the section along which the integration is performed, and Pt and P are the wellhead and bottomhole pressure values, respectively. The total length of the well was divided into three equal sections. The right hand side of Eq. A.17 was integrated using a third-order numerical method. Using the top pressure of the lowest section as the bottom pressure of the next section, the procedure was repeated until the wellhead was reached. An average value of P/z was obtained by calculating P/z at various depths. A range of sandface pressure values at the well from 500 to 9000 psia was chosen, and the appropriate values of P/z were calculated. A spline function was used to cover the entire range of bottom hole pressures from 500 to 9000 psia. Figure A.1 shows the behavior of the term $\frac{\partial[\bar{p}/\bar{z}(p)]}{\partial[p/z(p)]}$ as a function of the bottom hole pressure for a wellhead temperature of 75^{o} F, a temperature gradient of 20^{o} F per 1000 Ft, a gas gravity of 0.9, and three different values of well depth: 3000, 9000 and 15000 Ft.

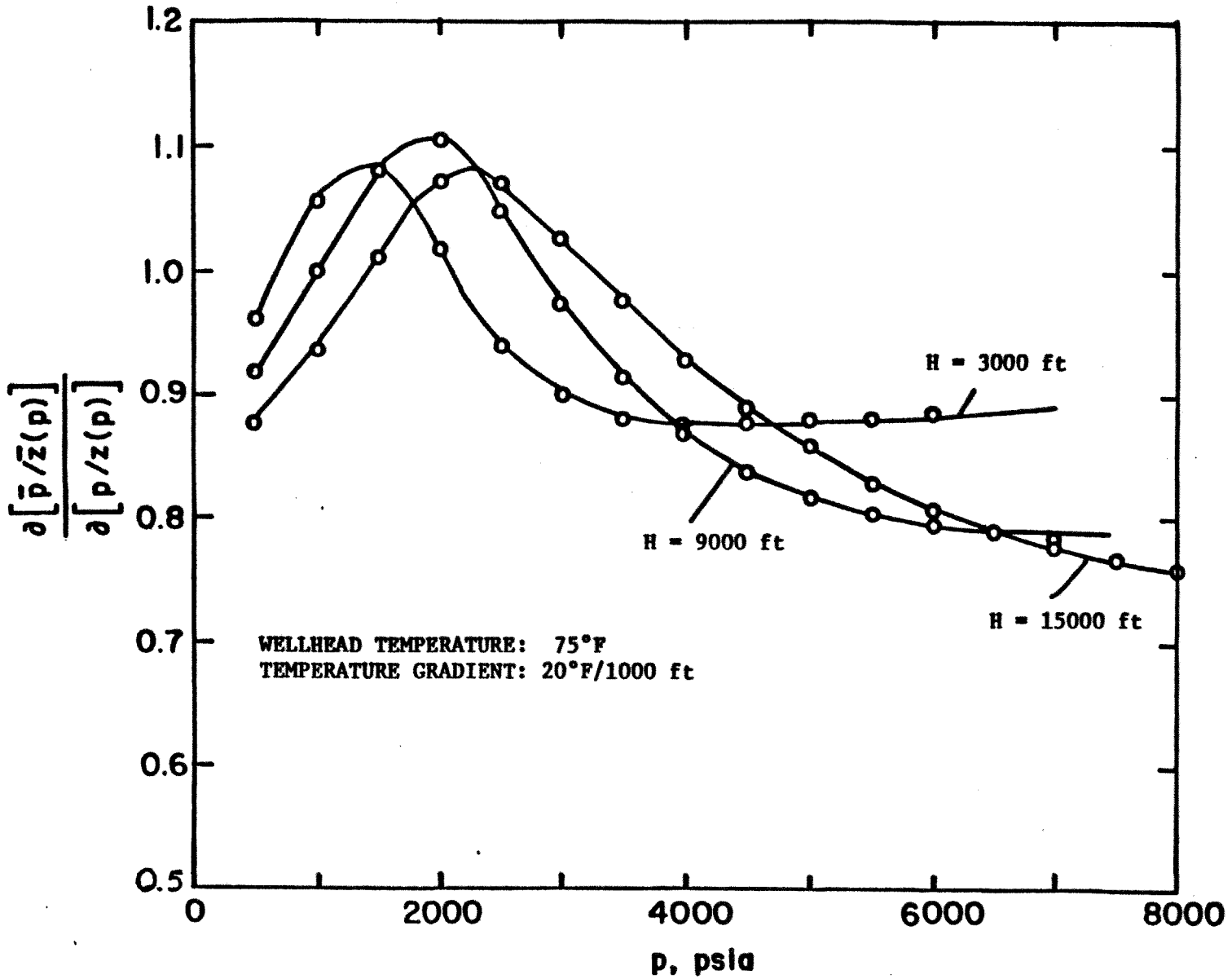
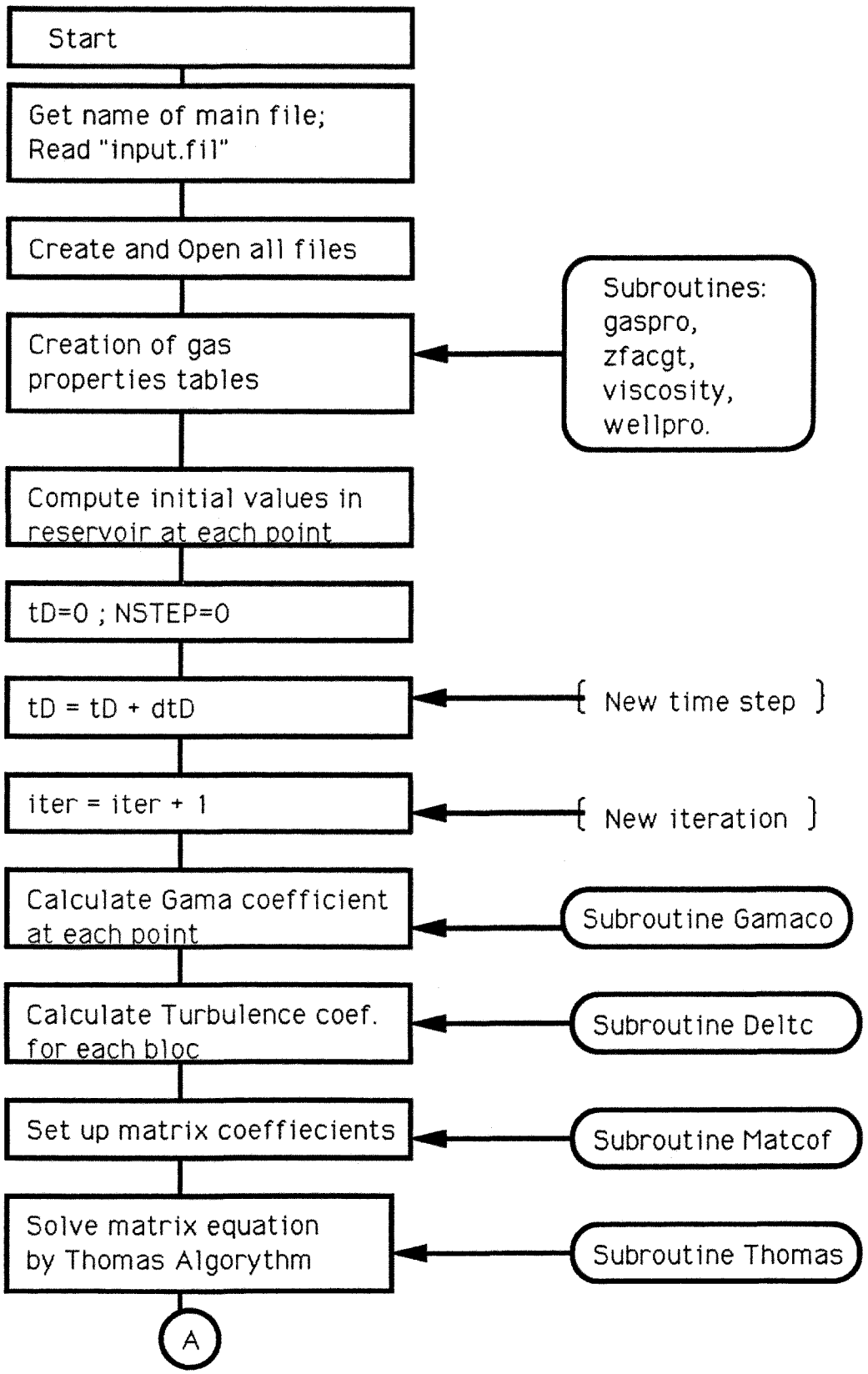
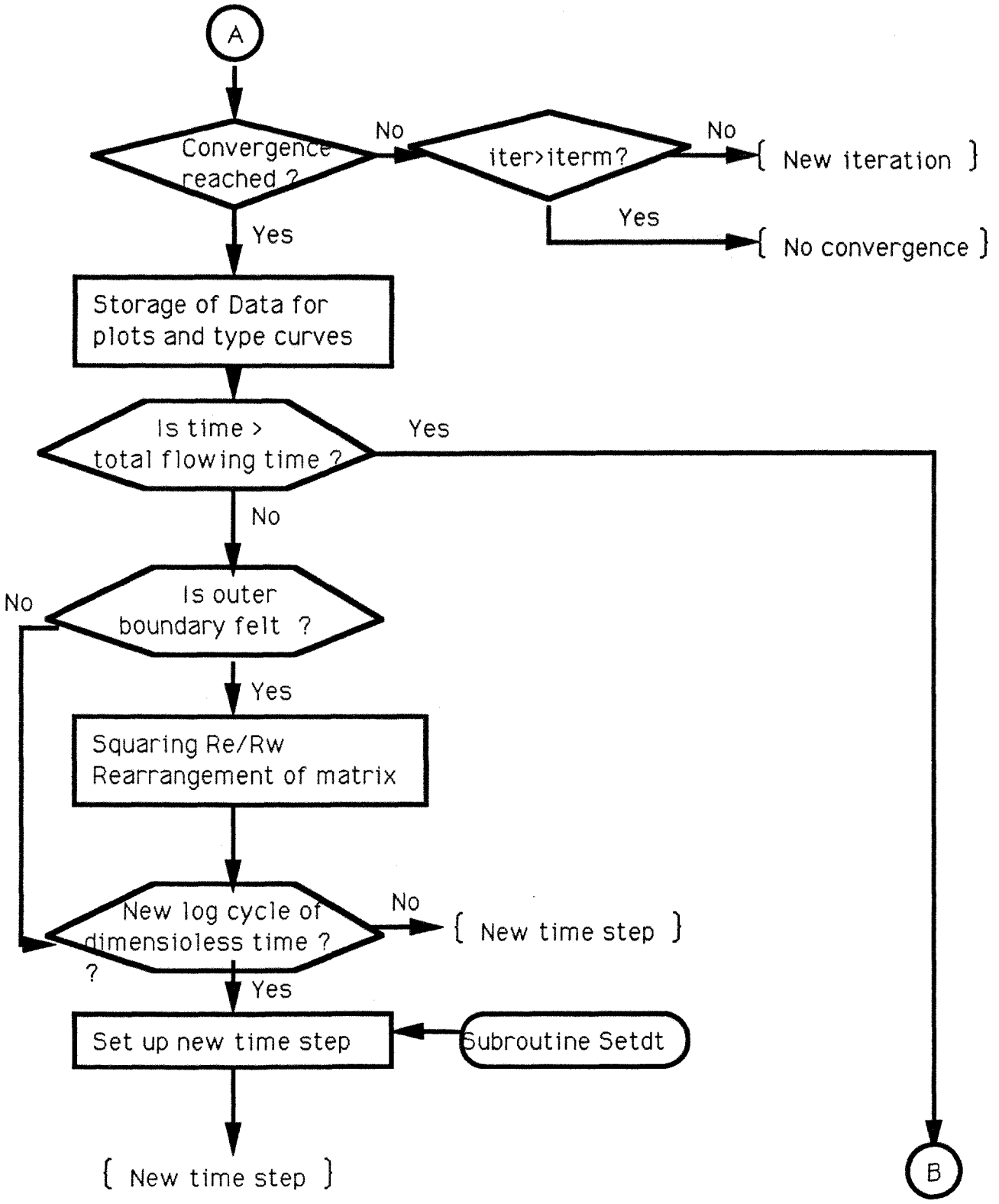
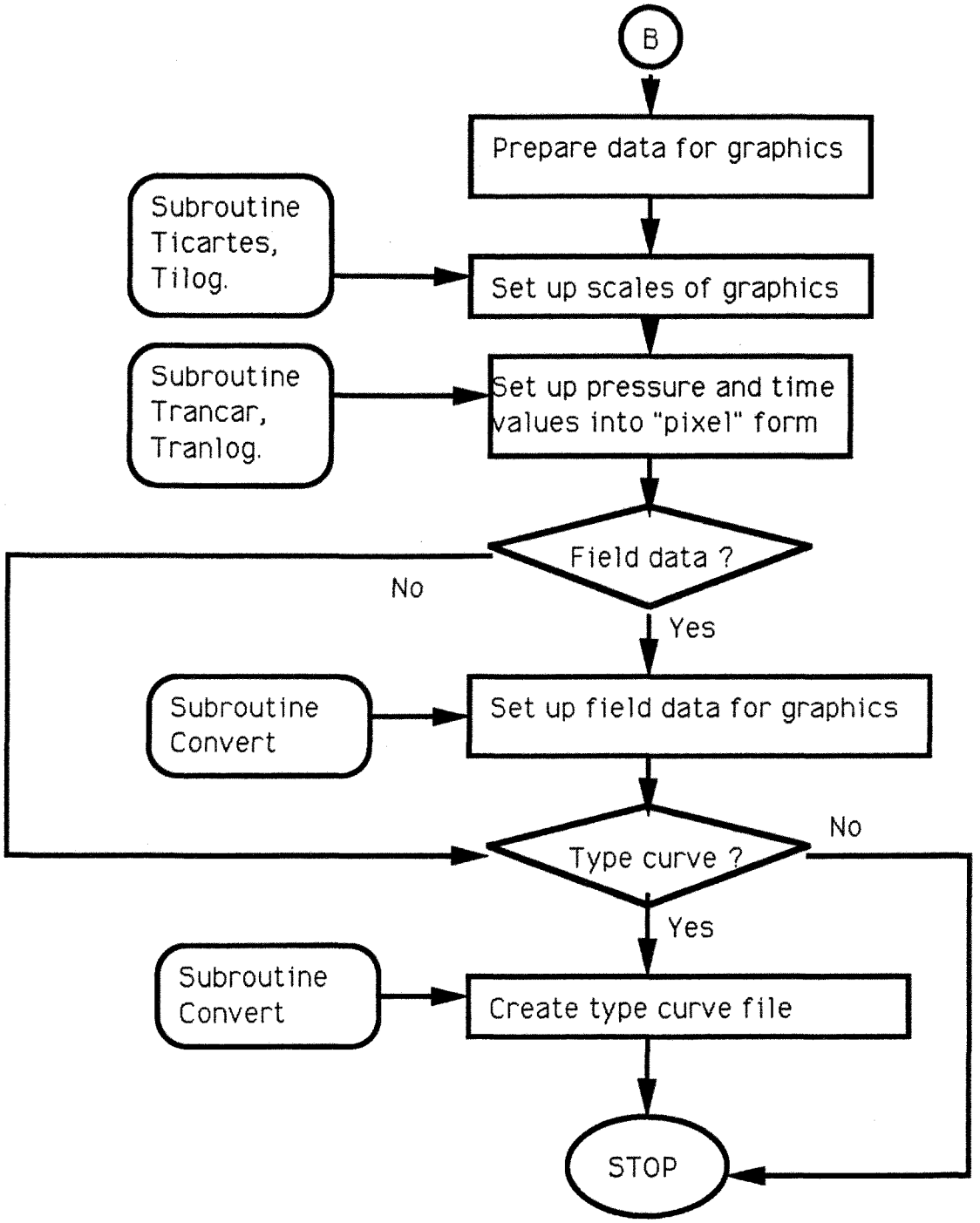


Fig. A.1. Wellbore Derivative $\frac{\partial[\overline{P}/\overline{z}(P)]}{\partial[P/z(P)]}$ Vs Bottom Hole Pressure for $\gamma_g = 0.9$

Appendix B. Flow Chart of Fortran Code




A
Convergence reached ?
No
iter>iterm?
No
{ New iteration }
Yes
Yes
{ No convergence }
Storage of Data for plots and type curves
Is time > total flowing time ?
Yes
No
No
Is outer boundary felt ?
Yes
Squaring Re/Rw
Rearrangement of matrix
New log cycle of dimensioless time ?
No
{ New time step }
?
Yes
Set up new time step
Subroutine Setdt
{ New time step }
B


B
Prepare data for graphics
Subroutine
Ticartes,
Tilog.
Set up scales of graphics
Subroutine
Trancar,
Tranlog.
Set up pressure and time
values into "pixel" form
Field data ?
No
Yes
Subroutine
Convert
Set up field data for graphics
Type curve ?
No
Yes
Subroutine
Convert
Create type curve file
STOP

Appendix C.

Source Code of Fortran Program for Flow Simulation

makefile


```
SOURCES = franc.f
OBJECTS = franc.o
program: $(OBJECTS)
	f77 -g $(OBJECTS) -o franc.x
.f.o:
	f77 -g -c $<
```

```
C     ************************************************************
C     COMPUTER MODEL OF RADIAL GAS FLOW
C     ************************************************************
C
      Implicit Real*4 (A-H,O-Z)
C
      Character*10 OUTERA, GASTYPE,type
      Character*3 CODBEA, gasty3
      Character*8 NAMEMA, NAMEFL, NAMEDT
      Character*12 MAINAME, FLONAME, ACTNAME
      Character*12 AAANAME, FIELDNAME, ACTFIELD, MSGNAME
C
      Dimension DEL(50), B(50), A(50,3), QSC(15),tidur(16)

      Real*4 MPTAB, MPI
C
      Common /table/ ptab(1000),mptab(1000),viscgt(1000)
      Common /table/ ztab(1000),vistab(1000),derivt(1000)
      Common /UR/ U(50) /G/ GAMA(50), GAMAP(50)
      Common /D/ DMP(50),DMITER(50),TERMJ(50)
      Common /D1/ DM(50) /DELTA/ DELT(50),DELTP(50)
      Common /DUR/ Dur(15)
      Common /typecu/ tcderiv(1000), tdtycu(1000), dmtycu(1000)
      Common /KAUTO/ Kauto(4)
C
      Open (Unit=7,File='input.fil')
      Rewind (Unit=7)
      Read (7,5800) MAINAME

      Open (Unit=8,File=MAINAME)
      Rewind (Unit=8)
C     ************************************************************
C     INPUT DATA:*****************************************************
C     ************************************************************
      Read (8,5500)
      Read (8,5900) NAMEMA
      Read (8,5900) NAMEFL
      Read (8,5000) TYPE
      Read (8,5000) OUTERA
      Read (8,5200) RE
      Read (8,5200) R1
      Read (8,5200) RW
      Read (8,5200) DEPTH
      Read (8,5200) THICK
      Read (8,5200) POROS
      Read (8,5200) SPG
      Read (8,5200) FRH2S
      Read (8,5200) FRCO2
      Read (8,5000) GASTYPE
      Read (8,5200) TEMWI
      Read (8,5200) TEMTI
      Read (8,5200) volume
      Read (8,5200) PRESI
      Read (8,5200) S
      Read (8,5200) PERMI
      Read (8,5100) CODBEA
      Read (8,5900) namedt
      Read (8,6000) Lcateg
C
      gasty3 = gastype(1:3)
C     ************************************************************
C     ********** open all data files ***************************
C     ************************************************************
      KINDEX = INDEX(NAMEFL,' ')
      KINDEX = KINDEX - 1
      FLONAME = NAMEFL(1:KINDEX) // '.FLO'

      KINDEX = INDEX(NAMEMA,' ')
      KINDEX = KINDEX - 1
      ACTNAME = NAMEMA(1:KINDEX) // '.ACT'
      AAANAME = NAMEMA(1:KINDEX) // '.AAA'
      IF((lcateg.eq.1).or.(lcateg.eq.2).or.(lcateg.eq.4)) then
        tycuck = 1.
        MSGNAME = NAMEMA(1:Kindex) // '.MSG'
        Open(unit=18,file=MSGNAME)
        Rewind(unit=18)
      else
        tycuck = 0.
      endif

      Open (Unit=10,File=FLONAME)
      Rewind (Unit=10)
      Open (Unit=11,File=ACTNAME)
      Rewind (Unit=11)
      Open (Unit=13,File=AAANAME)
      Rewind (Unit=13)
C
      fielcheck = 1.
      kindex = index(namedt,' ')
      if(kindex.EQ.1) then
         fielcheck = 0.
         goto 30
      endif
C
      kindex=kindex-1
      FIELDNAME=namedt(1:kindex) // '.PRS'
      ACTFIELD=namedt(1:kindex) // '.FLD'
      open(unit=16,file=FIELDNAME)
      rewind(unit=16)
      open(unit=17,file=ACTFIELD)
      rewind(unit=17)
C     ************************************************************
C     READ FLOW RATE FILE ********************************************
C     ************************************************************
   30 DURTOT = 0.
      nflo = 0
      Do 40 NDUR = 1, 15
        Read (10,*,err=42) TIDUR(NDUR), QSC(NDUR)
        nflo=nflo+1
   40 Continue
   42 do 45 ndur=1,nflo
        dur(ndur)=tidur(ndur+1) - tidur(ndur)
        durtot = durtot+dur(ndur)
   45 continue
C     ************************************************************
C     ******creation of gas-properties tables**********************
C     ************************************************************
      temw=temwi+460.
      temt=temti+460.
      call gaspro(gasty3,frco2,frh2s,temwi,spg,depth,temti)

      If ((OUTERA(1:2).EQ.'IN').OR.(OUTERA(1:2).EQ.'in')) Then
        REDI = 5.6234
        Go To 50
      End If

      RED = RE / RW
      REDI = RED ** 0.125
```

```
   50 R1D = R1 / RW

      If ((CODBEA(1:1).NE.'Y').AND.(CODBEA(1:1).NE.'y')) Then
        CODBET = 0.
        Go To 60
      End If

      CODBET = 1.
c     ****************************************************************
c     ********** initial values ***************************************
c     ****************************************************************

   60 NINTER = 36
      NDIM = 50
      ITERM = 50
c
      ERROR = 0.0001
      DTDI = 0.0001
      SIGMA = 1.
      TETA = 1.

      NPP = 0
      NQUATRO = 0
      DTD = DTDI
      RED = REDI
      M = nint(ninter*log(r1D)/log(reD) + 1)
c     ****************************************************************
c     EVALUATION OF BASIC PARAMETERS:*********************************
c     ****************************************************************
      MPI = sookmp(presi)
      VISCGI = sookcg(presi)
      vistii = sookvi(presi)
      zinit = sookzz(presi)
      derivi = sookde(presi)
      facini = (vistii*zinit)/presi

      DU = 1. / NINTER
      NMBR = NINTER + 1

      Do 80 I = 1, NMBR
        U(I) = DU * (I-1)
   80 Continue
c     ****************************************************************
c     ********** Compute dimensionless values ************************
c     ****************************************************************
      QD = 14.7*temw*qsc(1)/(thick*permi*mpi*520*1.987e-5)
      CD = derivi*volume*temw/((temw+temt)*3.1415*thick*poros*rw*rw)
      print *,QD
      TDMIN = DTD
      FS1 = 1.

      RAI = DTD / (DU**2)
      REDLG = LOG(RED)
      REDLG2 = REDLG ** 2
      TTOTD = (0.000264*PERMI) / (POROS*RW*RW*VISCGI)
      TDTOTA = TTOTD * DURTOT
c     ****************************************************************
c     ***** set initial values of min and max for graphs ************
c     ****************************************************************
      ZMITIME = 100.
      ZMITD = 1000000.
      ZMITPDT = 1.1
      ZMITRX = 100.
      ZMIXPRE = presi
      ZMIXMPW = 1.E+12
      ZMIDM = 100.
      ZMIDEMP = 1.E+12
      ZMIRATE = 1000000.
      zmilivee = 1000000.
c
      ZMATIME = 0.01
      ZMATD = 1.
      ZMATPDT = 9999.
      ZMATRX = 1000.
      ZMAXPRE = presi
      ZMAXMPW = 10.
      ZMADM = 0.
      ZMADEMP = 10.
      ZMARATE = 1.
      zmalivee = 1.
c     ****************************************************************
c     CALCULATION OF FS AND PERMS:************************************
c     ****************************************************************
      If (R1D.NE.1.) Then
        Go To 90
      End If

      FS = 1.
      Go To 100

   90 R1DLG = LOG(R1D)
      FS = 1. / (S/R1DLG+1.)
      PERMSI = PERMI * FS
c     ****************************************************************
c     PARAMETERS FOR CALCULATING THE TURBULENCE COEFFICIENT:**********
c     ****************************************************************
  100 If (CODBET.EQ.1.) Then
        Go To 110
      End If

      BETAI = 0.
      PERMRW = 0.
      Go To 120

  110 C1INV = 158.02119
      C2 = 2.89138E-14
      CRHO = 28.964 * SPG / (10.732E0*TEMW)
      CDERT = -CRHO * MPI * QD / (2.*RW*REDLG*DU)
      ARG = -1.201 * LOG(PERMI) + 23.83
      BETAI = EXP(ARG)
c     ****************************************************************
c     CALCULATION OF THE GROUP: K*K*BETAI/RW**************************
c     ****************************************************************
      PERMRW = PERMI * PERMI * BETAI / RW
c     ****************************************************************
c     ********* more inittial values *********************************
c     ****************************************************************

  120 NSTEP = 1
      NRED = 1
      LGSTNM = 0
      NMSTLG = 9
      LGCYNM = 1
      GPROD = 0.
      TD = DTD
c     ****************************************************************
c     STARTING VALUES OF DMP,DMITER,DELT,DELTP,GAMAP ***************
```

```
C     ****************************************************************
      Do 160 I = 1, NMBR
        DMP(I) = 0.
        DMITER(I) = 0.

        If (QD.GT.0.03) DMITER(I) = 0.35
        If (QD.GT.0.08) DMITER(I) = 0.65

        If (I.EQ.NMBR) Then
          Go To 130
        End If

        DELTP(I) = 1.
        DELT(I) = 1.

  130   If (I.GT.M) Then
          Go To 140
        End If

        Call GAMACO(NPTAB,I,QD,MPI,VISCGI,DMP(I),U(I),RED,REDLG2,FS,RAI,
     &       NCDDAT,DERIVI,PRESI,GAMAP(I))

        Go To 150

  140   Call GAMACO(NPTAB,I,QD,MPI,VISCGI,DMP(I),U(I),RED,REDLG2,FS1,
     &       RAI,NCDDAT,DERIVI,PRESI,GAMAP(I))

  150   GAMA(I) = GAMAP(I)

  160 Continue

      ALFAP1 = GAMAP(1)
      DMITP1 = DMITER(1)
C     ****************************************************************
C     EVALUATION OF PRESSURE VALUE IN AN ITERATIVE PROCEDURE
C     ****************************************************************
  170 ITER = 1

      If (NSTEP.EQ.1) Then
        Go To 190
      End If
C     ****************************************************************
C     DETERMINATION OF THE INITIAL GUESS VECTOR *********************
C     ****************************************************************
      Do 180 I = 1, NMBR
        DMITER(I) = DM(I) + SIGMA * (DM(I)-DMP(I))
        DMP(I) = DM(I)
        GAMAP(I) = GAMA(I)

        If (I.EQ.NMBR) Then
          Go To 180
        End If

        DELTP(I) = DELT(I)

  180 Continue

      DMITP1 = DMITER(1)
      ALFAP1 = GAMAP(1)
C     ****************************************************************
C     CALCULATION OF GAMA TERMS: ************************************
C     ****************************************************************
  190 Do 210 I = 1, NMBR
        If (I.GT.M) Then
          Go To 200
        End If

        Call GAMACO(NPTAB,I,QD,MPI,VISCGI,DMITER(I),U(I),RED,REDLG2,FS,
     &       RAI,NCDDAT,DERIVI,PRESI,GAMA(I))

        Go To 210

  200   Call GAMACO(NPTAB,I,QD,MPI,VISCGI,DMITER(I),U(I),RED,REDLG2,FS1,
     &       RAI,NCDDAT,DERIVI,PRESI,GAMA(I))

  210 Continue

      ALFA1 = GAMA(1)
C     ****************************************************************
C     CALCULATION OR THE TURBULENCE COEFFICIENT(DELT):**************
C     ****************************************************************
      If (BETAI.EQ.0.) Then
        Go To 220
      End If

      Call DELTC(NPTAB,NMBR,BETA,RED,QD,M,C2,C1INV,CRHO,CDERT,PERMI,
     &     PERMSI,PRESI,MPI)
C     ****************************************************************
C     REPRESENTATION OF THE MATRIX COEFFICIENTS:*********************
C     ****************************************************************
  220 Call MATCOF(NMBR,M,TETA,CD,REDLG,DU,DTD,FS,ALFAP1,ALFA1,NDIM,
     &     NSTEP,A,B,TD)
C     ****************************************************************
C     SOLVING THE EQUATION BY THOMAS ALGORITHM:**********************
C     ****************************************************************
      Call THOMAS(A,B,NINTER,NDIM)

      Do 230 I = 1, NINTER
        IP1 = I + 1
        DEL(IP1) = B(I)
  230 Continue

      DMITE2 = DMITER(2) + DEL(2)
      DMITE3 = DMITER(3) + DEL(3)

      A1 = 3. * TETA * DELT(1) / (2.*DU) + REDLG * ALFA1 * CD / (DTD*FS)

      DMITER(1) = ((3.*TETA*DELT(1)/(2.*DU)+TETA*DELT(2)/(2.*DU))*DMITE2
     &     -TETA*DELT(2)/(2.*DU)*DMITE3+TETA*REDLG/FS+ALFA1*TERMJ(1)) /
     &     A1

      DEL(1) = DMITER(1) - DMITP1
C     ****************************************************************
C     CHECKING FOR CONVERGENCE:**************************************
C     ****************************************************************
      ICHECK = 0
      Do 240 I = 1, NMBR
        ARG = DEL(I)

        If (ABS(ARG).GT.ERROR) Then
          ICHECK = 1
        End If

  240 Continue

      If (ICHECK.EQ.0) Then
        Go To 270
```

```
      End If

      ITER = ITER + 1
      if (ITER.LE.ITERM) Then
        Go To 250
      End If

      Print *, 'NO CONVERGENCE FOR QD=', QD, 'and Cd=', CD, 'at Td=',
     &    TD, 'and ITER=', ITER
      Go To 360
c     ****************************************************************
C     RE-CALCULATION OF DMITER ***************************************
c     ****************************************************************
  250 Do 260 I = 2, NMBR
        DMITER(I) = DMITER(I) + DEL(I)
  260 Continue

      DMITP1 = DMITER(1)

      Go To 190

  270 Do 280 I = 1, NMBR
        DM(I) = DMITER(I)
  280 Continue
c     ****************************************************************
c     *****Transformation to plotting values**************************
c     ****************************************************************
      if(nstep.LT.2) then
         xmprev=mpi
         tiprev=0.
      endif

      TIME = TD / TTOTD
      xmpw = mpi*(1.-QD*dm(1))
      xpre=dichot(xmpw)
      xnewmp = presi + (xmpw-mpi)*facini
      demp = facini*abs(mpi-xmpw)
      delivee = facini*time*abs(xmpw-xmprev)/abs(time-tiprev)
      xmprev=xmpw
      tiprev=time

      If (TIME.LE.1.E-03) Then
        Go To 290
      End If

      nquatro = nquatro + 1
      If (NQUATRO.EQ.5) Then
        NQUATRO = 1
      End If

      If (NQUATRO.NE.1) Then
        goto 290
      endif
c
        Write (13,5700) td, dm(1), delivee
        NPP = NPP + 1

        dimra2=10000.
        XPRE = dichot(xmpw)
        TPDT = (TPROD+TIME) / TIME
        TDRX = TD / DIMRA2
        RATE = QSC(1)
c     ****************************************************************
c     ********** Preparation of data for type curve ******************
c     ****************************************************************
      IF(tycuck.eq.1.) then
         IF(lcateg.eq.1) then
            tcderiv(npp)=delivee/(facini*mpi*abs(QD))
            tdtycu(npp)=td
            dmtycu(npp)=dm(1)
         Endif
c
         IF(lcateg.eq.2) then
            dmtycu(npp)=dm(1)
            tdtycu(npp)=tdrx
         Endif
c
         IF(lcateg.eq.4) then
            tcderiv(npp)=delivee/(facini*mpi*abs(QD))
            tdtycu(npp)=td/CD
            dmtycu(npp)=dm(1)
         Endif
      Endif
c     ****************************************************************
C     Computation of MIN and MAX values ******************************
c     ****************************************************************

        ZMITIME = AMIN1(ZMITIME,TIME)
        ZMATIME = AMAX1(ZMATIME,TIME)

        ZMITD = AMIN1(ZMITD,TD)
        ZMATD = AMAX1(ZMATD,TD)

        ZMIXPRE = AMIN1(ZMIXPRE,XPRE)
        ZMAXPRE = AMAX1(ZMAXPRE,XPRE)

        ZMIXMPW = AMIN1(ZMIXMPW,xnewmp)
        ZMAXMPW = AMAX1(ZMAXMPW,xnewmp)

        zmilivee = amin1(zmilivee,delivee,demp)
        zmalivee = amax1(zmalivee,delivee,demp)

        ZMIDM = AMIN1(ZMIDM,DM(1))
        ZMADM = AMAX1(ZMADM,DM(1))

        ZMIDEMP = AMIN1(ZMIDEMP,DEMP,delivee)
        ZMADEMP = AMAX1(ZMADEMP,DEMP,delivee)

        ZMIRATE = AMIN1(ZMIRATE,RATE)
        ZMARATE = AMAX1(ZMARATE,RATE)
c     ****************************************************************
C     PROCEDURE FOR THE NEXT TIME STEP*******************************
C     ****************************************************************
  290 NSTEP = NSTEP + 1
      If (TD.GT.TDTOTA) Then
        Go To 360
      End If
c     ****************************************************************
C     REARRANGEMENT OF 'DM' AND 'GAMA' TO FIT AN INCREASE IN RE/RW *
c     ****************************************************************
      If (DM(NMBR).LT.1.E-7) Then
        Go To 340
      End If

      NRED = NRED + 1

      If (NRED.GT.4) Then
        Go To 340
```

```
      End If

      RED = RED * RED
      REDLG = LOG(RED)
      REDLG2 = REDLG ** 2

      If (S.EQ.0..OR.R1D.EQ.1.) Then
        Go To 300
      End If
      M = nint(ninter*log(r1D)/log(reD) + 1)

  300 if(poxd.GE.1.) then
         mobs = nint(ninter*log(poxd)/log(reD) + 1)
         if(mobs.GT.(ninter+1)) then
            mobs = ninter + 1
         endif
      endif

      NN = (NMBR-1) / 2 - 1

      Do 310 I = 2, NN
        DM(I) = DM(2*I-1)
  310 Continue

      NN = NN + 1

      Do 320 I = NN, NMBR
        DM(I) = 0.
        GAMA(I) = 0.
  320 Continue

      If (CODBET.EQ.0.) Then
        Go To 340
      End If

      Do 330 I = 1, NMBR
        DMITER(I) = DM(I)
  330 Continue

      CDERT = -CRHO * MPI * QD / (2.*RW*REDLG*DU)
      Call DELTC(NPTAB,NMBR,BETA,RED,QD,M,C2,C1NV,CRHO,CDERT,PERMI,
     &     PERMSI,PRESI,MPI)
c     ****************************************************************
c     INCREASE IN TIME STEP:*****************************************
c     ****************************************************************
  340 LGSTNM = LGSTNM + 1

      If (LGSTNM.LE.NMSTLG) Then
        Go To 350
      End If

      LGCYNM = LGCYNM + 1
      TDMIN = 10. * TDMIN
      Call SETDT(LGCYNM,TDMIN,DU,DTD,LGSTNM,NMSTLG,RAI)

      Print *, 'Calculating'
  350 TD = TD + DTD
      Go To 170
c     ****************************************************************
c     ********** Preparation of Data for the Type Curve ***************
c     ****************************************************************
  360 If(tycuck.EQ.1.) then
         Kauto(1) = int(log10(zmadm)+1)
         Kauto(2) = int(log10(zmidm))
c
         IF(lcateg.eq.1) then
            Kauto(3) = int(log10(zmatd)+1)
            Kauto(4) = int(log10(zmitd))
         Endif
c
         IF(lcateg.eq.2) then
            Kauto(3) = int(log10(zmatrx)+1)
            Kauto(4) = int(log10(zmitrx))
         Endif
c
         IF(lcateg.eq.4) then
            Kauto(3) = int(log10(zmatd/CD)+1)
            Kauto(4) = int(log10(zmitd/CD))
         Endif
      Endif
c     ****************************************************************
c     ********** Storage of Graphic File for the Active Well ******
c     ****************************************************************
      Print *, 'Storing Data'
c
      mutime=100
      mupres=1000
      mudm=10
      murate=1000
      muxmpw=1000
c
      write(11,5800)'Active well'
      write(11,6000)npp
c
      zmitd = zmitd/1000000.
      zmatd = zmatd/1000000.
c
      if(fielcheck.eq.0.) then
         goto 362
      endif
      write(17,*)zmitime,zmatime
      write(17,*)zmitd,zmatd
      write(17,*)zmitrx,zmatrx
      write(17,*)zmixpre,zmaxpre
      write(17,*)zmixmpw,zmaxmpw
      write(17,*)zmilivee,zmalivee
      write(17,*)zmidm,zmadm
      write(17,*)zmidemp,zmademp
      write(17,*)zmirate,zmarate
      write(17,*)zmitpdt,zmatpdt
c
  362 call ticartes(zmitime,zmatime,mutime,kitimc,katimc,11)
      call tilog(zmitime,zmatime,kitiml,katiml,11)
      call tilog(zmitd,zmatd,kitdl,katdl,11)
      call tilog(zmitpdt,zmatpdt,kitpdt,katpdt,11)
      call tilog(zmitrx,zmatrx,kitrxl,katrxl,11)
      call ticartes(zmixpre,zmaxpre,mupres,kipres,kapres,11)
      call ticartes(zmixmpw,zmaxmpw,muxmpw,kixmpw,kaxmpw,11)
      call ticartes(zmidm,zmadm,mudm,kidmc,kadmc,11)
      call tilog(zmidm,zmadm,kidml,kadml,11)
      call tilog(zmidemp,zmademp,kideml,kademl,11)
      call ticartes(zmirate,zmarate,murate,kirate,karate,11)
      call tilog(zmilivee,zmalivee,kilivee,kalivee,11)
c
      call titles(11)
      rewind(unit=13)
c
      do 365 iter=1,npp
```

```
c
        read(13,5700) td, dm(1), delivee
c
        time = td/ttotd
        xmpw = mpi*(1.E0-qd*dm(1))
        xnewmp = presi + (xmpw-mpi)*facini
        xpre = dichot(xmpw)
        tpdt = (tprod + time)/time
        tdrx = 1000.
        demp = facini*abs(mpi-xmpw)
        rate = qsc(1)
        ynewtd = td/1000000.
c
        if(tpdt.GT.10000.) then
          tpdt = 10000.
        endif
c
        call trancar(time,itimec,kitimc,katimc)
        call tranlog(time,itimel,kitiml,katiml)
        call tranlog(ynewtd,itdl,kitdl,katdl)
        call tranlog(tpdt,itpdtl,kitpdt,katpdt)
        call tranlog(tdrx,itdrxl,kitrxl,katrxl)
        call trancar(xpre,ixprec,kipres,kapres)
        call trancar(xnewmp,ixmpwc,kixmpw,kaxmpw)
        call trancar(dm(1),idmc,kidmc,kadmc)
        call tranlog(dm(1),idml,kidml,kadml)
        call tranlog(demp,idempl,kideml,kademl)
        call trancar(rate,iratec,kirate,karate)
        call tranlog(delivee,ilivel,kilivee,kalivee)
c
        write(11,6100)itimec,itimel,itdl,itpdtl,itdrxl,ixprec,
     &          ixmpwc,idmc,idml,idempl,iratec,ilivel
c
  365 Continue
c     ********************************************************************
c     ********** End of Storage of Graphic File for the Active Well ****
c     ********************************************************************
      IF(fielcheck.EQ.1.) then
         call convert(type,ttotd,facini,dimra2,presi,qd)
      Endif
c
      IF(tycuck.EQ.1.) then
         call tycurve(NPP,mainame,Lcateg)
      Endif
c
  400 Stop
 5000 Format (A10)
 5100 Format (A3)
 5200 Format (F10.0)
 5300 Format (4F10.0)
 5400 Format (4E20.4)
 5500 Format (70A1)
 5600 Format (F10.0)
 5700 Format (E9.4,2x,E9.4,2x,e9.4)
 5800 Format (A12)
 5900 Format (A8)
 6000 format (I5)
 6100 format (12(I5,1X))
      End
      SUBROUTINE DELTC(NPTAB,NMBR,BETA,RED,QD,M,C2,C1INV,CRHO,CDERT,
     *PERMI,PERMSI,PRESI,MPI)

      IMPLICIT REAL*4(A-H,O-Z)

      COMMON /UR/ U(50)
      common /TABLE/ PTAB(1000),MPTAB(1000),VISCGT(1000)
      common /table/ ZTAB(1000),VISTAB(1000),derivt(1000)
      common /D/ DMP(50),DMITER(50),TERMJ(50)
      common /DELTA/ DELT(50),DELTP(50)

      REAL*4 MPI,MPTAB,MPAVE
c
      DO 200 I=2,NMBR

      IM1=I-1
      UAVE=(U(I)+U(IM1))/2.
      DMITAV=(DMITER(I)+DMITER(IM1))/2.

      MPAVE=MPI*(1.-QD*DMITAV)
      PRESAV = dichot(mpave)
      VISGAV = sookvi(presav)

      ZAVE = sookzz(presav)
      RHOAV=CRHO*PRESAV/ZAVE
      DERT=CDERT*VISGAV/RED**UAVE*(DMITER(I)-DMITER(IM1))

      IF(DERT.LT.0.) THEN
      DERT=0.
      ENDIF
c
      IF(I.LE.M) THEN
      GO TO 150
      ENDIF

      PERM=PERMI

      ARG=-1.201*LOG(PERM)+23.83
      BETA=EXP(ARG)
      B=VISGAV*C1INV/PERM
      VELOC=(-B+SQRT(B**2+4.*C2*BETA*DERT))/(2.*C2*BETA*RHOAV)
      DELT(IM1)=1./(1.+1.829744E-16*PERM*BETA*RHOAV*VELOC/VISGAV)
      GO TO 200
c
  150 PERMS=PERMSI

      ARG=-1.201*LOG(PERMS)+23.83
      BETAS=EXP(ARG)
      B=VISGAV*C1INV/PERMS
      VELOC=(-B+SQRT(B**2+4.*C2*BETAS*DERT))/(2.*C2*BETAS*RHOAV)
      DELT(IM1)=1./(1.+1.829744E-16*PERMS*BETAS*RHOAV*VELOC/VISGAV)
  200 CONTINUE
c
      RETURN
      END
C*********************************************************************
C*********************************************************************
      SUBROUTINE SPLINE(N,X,Y,B,C,D)
      INTEGER N
      Real*4 X(N),Y(N),B(N),C(N),D(N)
C
C     THE COEFFICIENTS B(I), C(I), AND D(I), I=1,2,...,N ARE COMPUTED
C     FOR A CUBIC INTERPOLATING SPLINE
C
C     S(X)=Y(I) + B(I)*(X-X(I)) + C(I)*(X-X(I))**2 +D(I)*(X-X(I))**3
C
C     FOR  X(I) .LE. X .LE. X(I+1)
C
C     INPUT..
```

```
C
C       N = THE NUMBER OF DATA POINTS OR KNOTS (N.GE.2)
C       X = THE ABSCISSAS OF THE KNOTS IN STRICTLY INCREASING ORDER
C       Y = THE ORDINATES OF THE KNOTS
C
C       OUTPUT..
C
C       B, C, D = ARRAYS OF SPLINE COEFFICIENTS AS DEFINED ABOVE.
C
C       USING P TO DENOTE DIFFERENTIATION,
C
C       Y(I) = S(X(I))
C       B(I) = SP(X(I))
C       C(I) = SPP(X(I))/2
C       D(I) = SPPP(X(I))/6   (DERIVATIVE FROM THE RIGHT)
C
C
        INTEGER NM1, IB, I
        Real*4 T
C
        NM1=N-1

        IF(N.LT.2) THEN
        RETURN
        ENDIF

        IF(N.LT.3) THEN
        GO TO 50
        ENDIF

C*******************************************************************************
C       SET UP TRIDIAGONAL SYSTEM**************************************************
C       B= DIAGONAL, D= OFFDIAGONAL,  C= RIGHT HAND SIDE.**************************
C*******************************************************************************

        D(1)=X(2)-X(1)
        C(2)=(Y(2)-Y(1))/D(1)

        DO 10 I=2,NM1
        D(I)=X(I+1)-X(I)
        B(I)=2.*(D(I-1)+D(I))
        C(I+1)=(Y(I+1)-Y(I))/D(I)
        C(I)=C(I+1)-C(I)
   10   CONTINUE

C*******************************************************************************
C       END CONDITIONS.  THIRD DERIVATIVES AT X(1) AND X(N)************************
C       OBTAINED FROM DIVIDED DIFFERENCES******************************************
C*******************************************************************************

        B(1)=-D(1)
        B(N)=-D(N-1)
        C(1)=0.
        C(N)=0.

        IF(N.EQ.3) THEN
        GO TO 15
        ENDIF

        C(1)=C(3)/(X(4)/X(2))-C(2)/(X(3)-X(1))
        C(N)=C(N-1)/(X(N)-X(N-2))-C(N-2)/(X(N-1)-X(N-3))
        C(1)=C(1)*D(1)**2/(X(4)-X(1))
        C(N)=-C(N)*D(N-1)**2/(X(N)-X(N-3))
C*******************************************************************************
C       FORWARD ELIMINATION*********************************************************
C*******************************************************************************
   15   DO 20 I=2,N
        T=D(I-1)/B(I-1)
        B(I)=B(I)-T*D(I-1)
        C(I)=C(I)-T*C(I-1)
   20   CONTINUE
C*******************************************************************************
C       BACK SUBSTITUITION**********************************************************
C*******************************************************************************
        C(N)=C(N)/B(N)

        DO 30 IB=1,NM1
        I=N-IB
        C(I)=(C(I)-D(I)*C(I+1))/B(I)
   30   CONTINUE
C*******************************************************************************
C       C(I) IS NOW THE SIGMA(I) OF THE TEXT*****************************************
C       COMPUTE POLYNOMIAL COEFFICIENTS*********************************************
C*******************************************************************************
        B(N)=(Y(N)-Y(NM1))/D(NM1)+D(NM1)*(C(NM1)+2.*C(N))

        DO 40 I=1,NM1
        B(I)=(Y(I+1)-Y(I))/D(I)-D(I)*(C(I+1)+2.*C(I))
        D(I)=(C(I+1)-C(I))/D(I)
        C(I)=3.*C(I)
   40   CONTINUE

        C(N)=3.*C(N)
        D(N)=D(N-1)

        RETURN
C
   50   B(1)=(Y(2)-Y(1))/(X(2)-X(1))
        C(1)=0.
        D(1)=0.

        RETURN
        END
C*******************************************************************************
C*******************************************************************************
        SUBROUTINE MATCOF(NMBR,M,TETA,CD,REDLG,DU,DTD,FS,ALFAP1,ALFA1,
       *NDIM,NSTEP,A,B,td)

        IMPLICIT REAL*4(A-H,O-Z)
        DIMENSION A(NDIM,3),B(NDIM)

        COMMON /G/ GAMA(50),GAMAP(50)
        common /D/ DMP(50),DMITER(50),TERMJ(50)
        common /DELTA/ DELT(50),DELTP(50)
C
C**** PURPOSE: THIS SUBROUTINE CALCULATES THE MATRIX COEFFICIENTS.
C
        N=NMBR-1
        NM1=N-1
        MM1=M-1
        MP1=M+1
C*******************************************************************************
c       INNER BOUNDARY CONDITION*****************************************
C*******************************************************************************
        A(1,1)=0.
        A1=3.*TETA*DELT(1)/(2.*DU)+REDLG*ALFA1*CD/(DTD*FS)
        A(1,2)=(TETA*DELT(1)/A1)*(3.*TETA*DELT(1)/(2.*DU)+TETA*
       *DELT(2)/(2.*DU))-(TETA*DELT(2)+TETA*DELT(1)+1./GAMA(2))
```

```
      A(1,3)=TETA*DELT(2)-(TETA*DELT(1)/A1)*TETA*DELT(2)/(2.*DU)

      IF(NSTEP.EQ.1.AND.CD.EQ.0.) THEN
      GO TO 10
      ENDIF

      TERMJ(1)=(1.-TETA)/(2.*DU*ALFAP1)*(3.*DELTP(1)*(DMP(2)-
     *DMP(1))-DELTP(2)*(DMP(3)-DMP(2)))+(1.-TETA)*REDLG/(FS*ALFAP1)
     *+REDLG*CD*DMP(1)/(FS*DTD)

      GO TO 20

   10 TERMJ(1)=0.

   20 TERMJ(2)=(1.-TETA)*GAMAP(2)*(DELTP(2)*(DMP(3)-DMP(2))-DELTP(1)
     **(DMP(2)-DMP(1)))+DMP(2)
      B(1)=-(A(1,2)*DMITER(2)+A(1,3)*DMITER(3)+(TETA*DELT(1)/A1)*
     *(TETA*REDLG/FS+ALFA1*TERMJ(1))+TERMJ(2)/GAMA(2))
C*********************************************************************
c     DAMAGED ZONE*****************************************************
C*********************************************************************
      IF(M.GE.NMBR) THEN
      GO TO 30
      ENDIF

      A(MM1,1)=TETA*DELT(MM1)
      A(MM1,2)=-(TETA*DELT(M)/FS+TETA*DELT(MM1)+1./GAMA(M))
      A(MM1,3)=TETA*DELT(M)/FS

      TERMJ(M)=(1.-TETA)*GAMAP(M)*(DELTP(M)/FS*(DMP(MP1)-DMP(M))-
     *DELTP(MM1)*(DMP(M)-DMP(MM1)))+DMP(M)
                                    *R
      B(MM1)=-(A(MM1,1)*DMITER(MM1)+A(MM1,2)*DMITER(M)+A(MM1,3)
     **DMITER(MP1)+TERMJ(M)/GAMA(M))
C*********************************************************************
c     OUTER BOUNDARY CONDITION*****************************************
C*********************************************************************
   30 A(N,1)=2.*TETA*DELT(N)
      A(N,2)=-(2.*TETA*DELT(N)+1./GAMA(NMBR))
      A(N,3)=0.

      TERMJ(NMBR)=-2.*(1.-TETA)*GAMAP(NMBR)*DELT(N)*(DMP(NMBR)-
     *DMP(N))+DMP(NMBR)

      B(N)=-(A(N,1)*DMITER(N)+A(N,2)*DMITER(NMBR)+TERMJ(NMBR)/GAMA(NMBR)
     *)
C*********************************************************************
C     GENERAL EQUATION*************************************************
C*********************************************************************
      DO 40 I=3,N
      IM1=I-1
      IP1=I+1

      IF(I.EQ.M) THEN
      GO TO 40
      ENDIF

      A(IM1,1)=TETA*DELT(IM1)
      A(IM1,2)=-(TETA*DELT(I)+TETA*DELT(IM1)+1./GAMA(I))
      A(IM1,3)=TETA*DELT(I)

      TERMJ(I)=(1.-TETA)*GAMAP(I)*(DELTP(I)*(DMP(IP1)-DMP(I))-
     *DELTP(IM1)*(DMP(I)-DMP(IM1)))+DMP(I)

      B(IM1)=-(A(IM1,1)*DMITER(IM1)+A(IM1,2)*DMITER(I)+A(IM1,3)*
     *DMITER(IP1)+TERMJ(I)/GAMA(I))

   40 CONTINUE
C
      RETURN
      END
C*********************************************************************
C*********************************************************************
      SUBROUTINE THOMAS(A,B,NN,NMAX)
      IMPLICIT REAL*4 (A-H,O-Z)
C*********************************************************************
C**** GENERAL TRIDIAGONAL NON-SYMETRIC MATRIX SOLVER
C     A(I,1),A(I,2),A(I,3) - THE COEFFICIENTS OF THE MATRIX.
C     B(I) - THE RIGHT HAND SIDE OF THE EQUATIONS.
C*********************************************************************
      DIMENSION A(NMAX,3),B(NMAX)


      B(1)=B(1)/A(1,2)
      A(1,2)=A(1,3)/A(1,2)

      DO 40 N=2,NN
         W=A(N,2)-A(N,1)*A(N-1,2)
         A(N,2)=A(N,3)/W
         ARG=B(N-1)

         IF(ABS(ARG).GT.1.D-30) GO TO 37

         B(N)=0.
         GO TO 40

   37    B(N)=(B(N)-A(N,1)*B(N-1))/W

   40 CONTINUE
      N=NN

   50 N=N-1

      IF(N.LT.1) RETURN

      B(N)=B(N)-A(N,2)*B(N+1)

      GO TO 50
      END
C*********************************************************************
C*********************************************************************
      SUBROUTINE SETDT(LGCYNM,TDMIN,DU,DTD,LGSTNM,NMSTLG,RAI)
      IMPLICIT REAL*4(A-H,O-Z)
C
C**** THIS SUBROUTINE CALCULATES TIME STEP SEQUENCE
C
      IF(LGCYNM.GE.3) THEN
      GO TO 20
      ENDIF

      NMSTLG=18
      LGSTNM=1
      DTD=0.5*TDMIN
      GO TO 50

   20 NMSTLG=90
      LGSTNM=1
      DTD=0.1*TDMIN
```

```
  50  RAI=DTD/(DU**2)

      RETURN
      END
C*********************************************************************
C*********************************************************************
      SUBROUTINE GAMACO(NPTAB,I,QD,MPI,VISCGI,DM,U,RED,REDLG2,FS,
     *RAI,NCDDAT,DERIVI,PRESI,GAMA)

      IMPLICIT REAL*4(A-H,O-Z)

      COMMON /TABLE/ PTAB(1000),MPTAB(1000),VISCGT(1000)
      common /table/ ZTAB(1000),VISTAB(1000),derivt(1000)

      REAL*4 MP,MPI,MPTAB
C
C     PURPOSE: THIS SUBROUTINE CALCULATES GAMA COEFFICIENTS*********
C
      MP=MPI*(1.-QD*DM)
      PRES = dichot(mp)
      VISCG = sookcg(pres)

      PERMR=1.

      ALFA=VISCG/VISCGI/PERMR
      REDEX=RED**(2.*U)
      GAMA=FS*RAI/(REDEX*REDLG2*ALFA)

      IF(I.GT.1) THEN
      GO TO 30
      ENDIF

C*********************************************************************
c     WHEN I=1:*****************************************************
C*********************************************************************
      DERIV = sookde(pres)
      GAMA=ALFA*DERIV/DERIVI

  30  RETURN
      END
C*********************************************************************
C*********************************************************************
      real*4 function sookmp(p)
      real*4 p,mptab
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
c*****This subroutine is a table-lookup*****************
c
      a = p/10.
      i = int(a)
      rlpart = abs(a-int(a))

      sookmp = mptab(i) + rlpart*(mptab(i+1)-mptab(i))

      return
      end
c*************************************************************
c*************************************************************
c
      real*4 function dichot(mp)
      real*4 mp, mptab
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
c*****this subroutine is a dichotomy scheme to get P. from m(P).
c
      npetit=1
      ngrand=1000
      milieu=500

   10 if(mptab(milieu).EQ.mp) then
         dichot = ptab(milieu)
         goto 30
      endif

      if((ngrand-npetit).EQ.1) then
         goto 20
      endif

      if(mptab(milieu).LT.mp) then
         npetit = milieu
         milieu = int((npetit+ngrand)/2)
         goto 10
      else
         ngrand=milieu
         milieu = int((ngrand+npetit)/2)
         goto 10
      endif

   20 Aratio = (mp-mptab(npetit))/(mptab(ngrand)-mptab(npetit))
      dichot = 10.*(Aratio + npetit)

   30 return
      end
c*************************************************************
C*************************************************************
      real*4 function sookcg(p)
      real*4 p
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
c*****This subroutine is a table-lookup*****************
c
      a = p/10.
      i = int(a)
      rlpart = abs(a-int(a))

      sookcg = viscgt(i) + rlpart*(viscgt(i+1)-viscgt(i))

      return
      end
c*************************************************************
C*************************************************************
      real*4 function sookzz(p)
      real*4 p
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
c*****This subroutine is a table-lookup*****************
c
      a = p/10.
      i = int(a)
```

```
      rlpart = abs(a-int(a))

      sookzz = ztab(i) + rlpart*(ztab(i+1)-ztab(i))

      return
      end
c*************************************************************
c*************************************************************
      real*4 function sookvi(p)
      real*4 p
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
c*****This subroutine is a table-lookup*****************
c
      a = p/10.
      i = int(a)
      rlpart = abs(a-int(a))

      sookvi = vistab(i) + rlpart*(vistab(i+1)-vistab(i))

      return
      end
c*************************************************************
c*************************************************************
      real*4 function sookde(p)
      real*4 p
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
c*****This subroutine is a table-lookup*****************
c
      a = p/10.
      i = int(a)
      rlpart = abs(a-int(a))

      sookde = derivt(i) + rlpart*(derivt(i+1)-derivt(i))

      return
      end
c*************************************************************
      subroutine gaspro(gasty3,xco2,xh2s,temwi,spg,wd,temti)
c
      implicit real*4(a-h,o-z)
      real*4 mptab
      CHARACTER*3 gasty3
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
      IF((gasty3.EQ.'CAL').or.(gasty3.eq.'cal')) THEN
          PCRIT = 677. + 15.*SPG - 37.5*(SPG*SPG)
          TCRIT = 168. + 325.*SPG - 12.5*(SPG*SPG)
          GOTO 10
      ENDIF

      PCRIT = 706. - 51.7*SPG - 11.1*(SPG*SPG)
      TCRIT = 187. + 330.*SPG - 71.5*(SPG*SPG)

   10 EPS = 120.*(XCO2+XH2S)**0.9 - 120.*(XCO2+XH2S)**1.6
     &+15.*(XCO2**0.5 - XH2S**4.0)
c
      TSTAR = TCRIT - EPS
      PSTAR = PCRIT*TSTAR/(TCRIT+XH2S*EPS*(1.-XH2S))
c
      call wellpro(tstar,pstar,wd,temwi,temti)
c
      TEMW = TEMWI + 460.
      TREDU = TEMW/TSTAR
c
      ptab(1) = 10.
      predu=ptab(1)/pstar
      call zfacgt(ptab(1),predu,tredu,ztab(1),cgt)
      call viscosity(spg,temw,ptab(1),ztab(1),vistab(1))
      viscgt(1) = vistab(1)*cgt
      mptab(1)=0.
c
      xold = ptab(1)/(ztab(1)*vistab(1))
      sum = 0.
      fac = 10.
c
      do 40 i=2,1000
c
         pab=i*10.
         predu=pab/pstar
         call zfacgt(pab,predu,tredu,zab,cgt)
         call viscosity(spg,temw,pab,zab,visab)
         xnew = pab/(zab*visab)
         sum = sum + fac*(xnew + xold)
         xold = xnew
c
            mptab(i)=sum
            ptab(i)=pab
            ztab(i)=zab
            vistab(i)=visab
            viscgt(i)=visab*cgt
c
   40 continue
      return
      END
c*************************************************************************
c*************************************************************************
      Subroutine zfacgt(ptab,predu,tredu,ztab,cgt)
c
      REAL*4 invers, invcgt
c
      AA = 0.06423
      BB = 0.5353*TREDU - 0.6123
      CC = 0.3151*TREDU - 1.0467 - 0.5783/TREDU**2.
      DD = TREDU
      EE = 0.6816/TREDU**2.
      FF = 0.6845
      GG = 0.27*PREDU
c
      DENRD = 0.27**PREDU/TREDU
c
      DO 10 ITER=1,20
c
      DENRD2 = DENRD*DENRD
      DENRD3 = DENRD2 * DENRD
      DENRD4 = DENRD2 * DENRD2
      DENRD5 = DENRD2 * DENRD3
      DENRD6 = DENRD3 * DENRD3
c
      FONC = AA*DENRD6 + BB*DENRD3 + CC*DENRD2 + DD*DENRD - GG
     &+EE*DENRD3*(1.+FF*DENRD2)*EXP(-FF*DENRD2)
```

```
c
      FONPRI = 6.*AA*DENRD5 + 3.*BB*DENRD2 +2.*CC*DENRD + 3D
     &+EE*DENRD2*(3.+FF*DENRD2*(3.-2.*FF*DENRD2))*EXP(-FF*DENRD2)
c
      DENRDN = DENRD - FONC/FONPRI
c
      IF(ABS(DENRDN-DENRD).LT.0.5E-04) THEN
      GOTO 20
      ENDIF
c
   10 DENRD = DENRDN
c
   20 ZTAB = (0.27*PREDU)/(DENRDN*TREDU)
c
      DENRD = DENRDN
      DENRD2 = DENRD*DENRD
      DENRD4 = DENRD2*DENRD2
      DENRD5 = DENRD4*DENRD
      FF2 = FF*FF
      INVERS = 1./(DENRD*TREDU)
c
      DEZDEN = INVERS*(5.*AA*DENRD5 + 2.*BB*DENRD2 + CC*DENRD
     &+2.*EE*DENRD2*(1.+FF*DENRD2-FF2*DENRD4)*EXP(-FF*DENRD2))
c
      INVCGT = PTAB*(1. + DENRD*DEZDEN/ZTAB)
      CGT = 1./INVCGT
c
      Return
      END
c*************************************************************************
c*************************************************************************
      Subroutine viscosity(spg,temw,ptab,ztab,vistab)
c
      xmolw = 28.9625*spg
      rho = ptab*xmolw*1.4935E-03/(ztab*temw)
      xk = (9.379+0.01607*xmolw)*temw**1.5
      xk = xk/(209.2+19.26*xmolw+temw)
      x = 3.448 + 986.4/temw + 0.01009*xmolw
      y = 2.447 - 0.2224*x
      vistab = 1.E-04*xk*exp(x*rho**y)
c
      return
      end
c*********************************************************************
      subroutine wellpro(tstar,pstar,wd,temwi,temti)
c
      Dimension A(4), PZ(4)
      Dimension PZI(1000), PZAVE(1000), B1(1000), C1(1000), D1(1000)
c
      common /table/ ptab(1000),mptab(1000),viscgt(1000)
      common /table/ ztab(1000),vistab(1000),derivt(1000)
c
      Data A /0.125, 0.375, 0.375, 0.125/
c
      NPRSWM = 1000
      PRSWI = 10.
c
      NPRSW = 1
      A1 = (TEMTI-TEMWI) / WD
c
   10 NP = 2
      D = WD / 3.
      DEPTH = WD
      PRSW = PRSWI
      TEMW = TEMWI
c*************************************************************************
c********at bottom hole condition*****************
c*************************************************************************
      predu=prsw/pstar
      tredu=(temw+460.)/tstar
      Call ZFACGT(PRSW,PREDU,TREDU,ZED,CJUNK)
      PTZ = (TEMW+460.) * ZED / PRSW
      PZ(1) = PRSW / ZED
      PZI(NPRSW) = PZ(1)
c*************************************************************************
c***********at various depth
c*************************************************************************
   20 ITER = 0
      TEMT = A1 * D + TEMW
      DEPTH = DEPTH - D
      GL = 0.01875 * SPG * D
      YY = A(1) * PTZ
      PRS = PRSW
      TEM = TEMW
      PRST = PRSW - GL * PRSW / ((TEMW+460.)*ZED)
      Y = YY
c
   30 ITER = ITER + 1
      PRSINC = (PRST-PRSW) / 3.
      TEMINC = (TEMT-TEMW) / 3.
c*************************************************************************
c************integration loop
c*************************************************************************
      Do 40 I = 2, 4
        PRS = PRS + PRSINC
        TEM = TEM + TEMINC
        predu=prs/pstar
        tredu=(tem+460.)/tstar
        Call ZFACGT(PRS,PREDU,TREDU,ZED,CJUNK)
        PTZ = (TEM+460.) * ZED / PRS
        Y = Y + A(I) * PTZ
   40 Continue
c*************************************************************************
c********newton raphson iteration
c*************************************************************************
      FUN = GL - (PRSW-PRST) * Y
      PRSN = PRST - FUN / PTZ
      ERROR = ABS(PRST-PRSN) / PRST
      PRST = PRSN
c
      If (ERROR.LT.0.002) goto 45
c
      PRS = PRSW
      TEM = TEMW
      Y = YY
      Go To 30
c
   45 PREDU = PRST/PSTAR
      TREDU = (TEM + 460.)/TSTAR
      Call ZFACGT(PRST,PREDU,TREDU,ZED,CJUNK)
      PTZ = (TEMT+460.) * ZED / PRST
      PZ(NP) = PRST / ZED
c
      NP = NP + 1
      If (NP.GT.4) goto 50
      TEMW = TEMT
      PRSW = PRST
```

```
      Go To 20
c**********************************************************************
c******************calculation of the average p/z
c**********************************************************************
   50 PZAVE(NPRSW)=A(1)*PZ(1) + A(2)*PZ(2) + A(3)*PZ(3) + A(4)*PZ(4)
c
c
      NPRSW = NPRSW + 1
      If (NPRSW.GT.NPRSWM) goto 60
c
        if(NPRSW.LE.NPRSWM) then
          prswi=prswi+10.
          goto 10
        endif
c**********************************************************************
c****************calculation of the derivative
c**********************************************************************
   60 Call SPLINE(NPRSWM,PZI,PZAVE,B1,C1,D1)
      do 70 i=1,NPRSWM
         derivt(i)=b1(i)
   70 continue
      return
      End
c**********************************************************************
c**********************************************************************
      subroutine ticartes(zmin,zmax,multip,jmin,jmax,key)
c
      Dimension natab(13)
c
      a=zmin/multip
      imin=int(a)
      jmin=multip*imin
c
      b=zmax/multip
      imax=int(b+1.)
      jmax=multip*imax
c
      interm=imax-imin-1
c
      if(interm.ge.14) then
         interm = 0
      endif
c
      if(interm.eq.0) then
         write(key,20) jmin, jmax, interm
         goto 40
      endif
c
      iint=imin
      do 10 iter=1,interm
         iint=iint+1
         natab(iter)=multip*iint
   10 Continue
c
      write(key,30) jmin, jmax, interm, (natab(iter),iter=1,interm)
c
   20 format(3(I5,1X))
   30 format(16(I5,1X))
   40 Return
      end
c**************************************************************
c**************************************************************
      subroutine tilog(zmin,zmax,imin,imax,key)
c
      character*5 ch5tab(13)
      character*5 ch5min, ch5max,ch5
c
      if(zmin.LT.1.) then
         zmin=zmin/10.
      endif
c
      a=log10(zmin)
      imin=int(a)
      ymin=10.**(imin*1.E0)
      call change(ymin,ch5)
      ch5min=ch5
c
      b=log10(zmax)
      imax=int(b+1.)
      ymax=10.**(imax*1.E0)
      call change(ymax,ch5)
      ch5max=ch5
c
      interm=imax-imin-1
c
      if(interm.ge.14) then
         interm=0
      endif
c
      if(interm.eq.0) then
         write(key,20) ch5min, ch5max, interm
         goto 40
      endif
c
      iint=imin
c
      do 10 iter=1,interm
         iint=iint+1
         yint=10.**(iint*1.E0)
         call change(yint,ch5)
         ch5tab(iter)=ch5
   10 continue
c
      write(key,30) ch5min, ch5max, interm, (ch5tab(iter),iter=1,interm)
c
   20 format(2(A5,1X),I5)
   30 format(2(A5,1X),I5,1X,13(A5,1X))
   40 Return
      End
c**********************************************************************
c**********************************************************************
      Subroutine change(a,ch5)
c
      character*5 ch5
      character*10 ch10
c
      write(ch10,10)a
      kindex = index(ch10,'.')
      kondex = index(ch10,'1')
      if(a.GE.1.) then
           ch5=ch10(1:5)
      else
           ch5=ch10(kindex:kondex)
      endif
c
   10 format(F10.4)
      Return
      End
```

```
c*************************************************************
c*************************************************************
      Subroutine Titles(key)
c
      character*5 title(12)
c
      Title(1)='timec'
      Title(2)='timel'
      Title(3)='dimtl'
      Title(4)='tpdtl'
      Title(5)='tdrxl'
      Title(6)='xprec'
      Title(7)='xmpwc'
      Title(8)='dimmc'
      Title(9)='dimml'
      Title(10)='dempl'
      Title(11)='ratec'
      title(12)='livel'
c
      write(key,10)  (title(iter),iter=1,12)
   10 format(12(A5,1X))
      return
      end
c*************************************************************
c*************************************************************
      subroutine tranlog(param,iparal,imin,imax)
c
      paralo=log10(param)
      parapo=paralo-imin
      iparal=nint(parapo*10000./(imax-imin))
c
      Return
      end
c*************************************************************
c*************************************************************
      subroutine trancar(param,iparac,jmin,jmax)
c
      parapo=param-jmin
      iparac=nint(parapo*10000./(jmax-jmin))
c
      return
      end
c*************************************************************
c
      subroutine convert (type,ttotd,facini,dimra2,presi,qd)
c
      implicit real*4 (a-h,o-z)
      real*4 time(500),xpre(500),xmpw(500),demp(500),dm(500)
      real*4 xnewmp(500),tpdt(500),tdrx(500),delivee(500),ynewtd(500)
      real*4 mpi
      dimension jval(500),kval(500)
      character*10 type
c
      common /dur/ dur(15)
c
      mpi=sookmp(presi)
      dummy = 200.
      if((type(1:2).EQ.'BU').OR.(type(1:2).EQ.'bu')) then
         tprod = dur(1)
      else
         tprod = dummy
      endif
c
      print *, 'Storing Field Data'
c
      mutime=100
      mupres=1000
      mudm=10
      muxmpw=1000
      murate=1000
c
      rewind (unit=17)
      read(17,*)zmitime,zmatime
      read(17,*)zmitd,zmatd
      read(17,*)zmitrx,zmatrx
      read(17,*)zmixpre,zmaxpre
      read(17,*)zmixmpw,zmaxmpw
      read(17,*)zmilivee,zmalivee
      read(17,*)zmidm,zmadm
      read(17,*)zmidemp,zmademp
      read(17,*)zmirate,zmarate
      read(17,*)zmitpdt,zmatpdt
      rewind (unit=17)
c
      write(17,5800) 'Field Data'
      npoint_printed=0
      rewind(unit=16)
      do 10 i=1,500
         read(16,*,end=20) time(i), xpre(i)
         npoint_printed = npoint_printed+1
c
         ynewtd(i)=time(i)*ttotd/1000000.
         tpdt(i)=(tprod+time(i))/time(i)
         tdrx(i)=1000.
         xmpw(i)=sookmp(xpre(i))
         dm(i)=(mpi-xmpw(i))/(mpi*qd)
         demp(i)=facini*abs(mpi-xmpw(i))
         xnewmp(i)=presi + facini*(xmpw(i)-mpi)
c
         if(tpdt(i).GT.10000.) then
            tpdt(i)=10000.
         endif
   10 continue
c
   20 write(17,6000)npoint_printed
c
c
      call ticartes(zmitime,zmatime,mutime,kitimc,katimc,17)
      call tilog(zmitime,zmatime,kitiml,katiml,17)
      call tilog(zmitd,zmatd,kitdl,katdl,17)
      call tilog(zmitpdt,zmatpdt,kitpdt,katpdt,17)
      call tilog(zmitrx,zmatrx,kitrxl,katrxl,17)
      call ticartes(zmixpre,zmaxpre,mupres,kipres,kapres,17)
      call ticartes(zmixmpw,zmaxmpw,muxmpw,kixmpw,kaxmpw,17)
      call ticartes(zmidm,zmadm,mudm,kidmc,kadmc,17)
      call tilog(zmidm,zmadm,kidml,kadml,17)
      call tilog(zmidemp,zmademp,kideml,kademl,17)
      call ticartes(zmirate,zmarate,murate,kirate,karate,17)
      call tilog(zmilivee,zmalivee,kilivee,kalivee,17)
c
      call titles(17)
c     ***************************************************
c     **********Satisfy condition in "Horne"page 49********
c     ***************************************************
      jval(1)=npoint_printed
      ibon=1
```

```
      reference=time(npoint_printed)
c
      do 30 i=npoint_printed-1,1,-1
         if(abs(log(reference)-log(time(i))).GE.0.2) then
           ibon=ibon+1
           jval(ibon)=i
           reference=time(i)
         endif
   30 continue
c     ********************************************************
c     **********Inverse jval into kval **********************
c     ********************************************************
      do 40 i=1,ibon
         kval(i)=jval(ibon+1-i)
   40 continue
c     ********************************************************
c     ********** Compute derivative curve ********************
c     ********************************************************
      do 50 iter=1,ibon-2
       li=kval(iter+1)
       limk=kval(iter)
       lipj=kval(iter+2)
       der11=log(time(li)/time(limk))*demp(lipj)
       der22=log(time(lipj)/time(li))*log(time(lipj)/time(limk))
       der33=log(time(lipj)*time(limk)/(time(li)*time(li)))*demp(li)
       der44=log(time(lipj)/time(li))*log(time(li)/time(limk))
       der55=log(time(lipj)/time(li))*demp(limk)
       der66=log(time(li)/time(limk))*log(time(lipj)/time(limk))
c
       delivee(li)=der11/der22 + der33/der44 - der55/der66
c
   50 continue
c
      do 70 i=1,npoint_printed
c
          call trancar(time(i),itimec,kitimc,katimc)
          call tranlog(time(i),itiml,kitiml,katiml)
          call tranlog(ynewtd(i),itdl,kitdl,katdl)
          call tranlog(tpdt(i),itpdtl,kitpdt,katpdt)
          call tranlog(tdrx(i),itdrxl,kitrxl,katrxl)
          call trancar(xpre(i),ixprec,kipres,kapres)
          call trancar(xnewmp(i),ixmpwc,kixmpw,kaxmpw)
          call trancar(dm(i),idmc,kidmc,kadmc)
          call tranlog(dm(i),idml,kidml,kadml)
          call tranlog(demp(i),idempl,kideml,kademl)
          call tranlog(delivee(i),illivel,kilivee,kalivee)
          iratec=0
          if(illivel.LT.0) then
             illivel=0
          endif
c
          write(17,6100)itimec,itiml,itdl,itpdtl,itdrxl,ixprec,
     &         ixmpwc,idmc,idml,idempl,iratec,illivel
   70 continue
 5800 format(A12)
 6000 format(I5)
 6100 format(12(I5,1X))
 7000 Return
      end
c     ********************************************************
c     ********************************************************
      Subroutine Tycurve(NPP,Mainame,Lcateg)
c
      Character*2 Yaxislab
      Character*6 Xaxislab
      Character*12 Mainame
      Character*30 Toplabel
c
      Common /typecu/ tcderiv(1000), tdtycu(1000), dmtycu(1000)
      Common /Kauto/ Kauto(4)
c
      Print *, 'Storing Type Curve'
      Toplabel = 'Type Curve from ' // Mainame
      Yaxislab = 'pD'
c
      IF(Lcateg.eq.1) then
         Xaxislab = ' tD '
      Endif
c
      IF(Lcateg.eq.2) then
         Xaxislab = 'tD/rD2'
      Endif
c
      IF(Lcateg.eq.4) then
         Xaxislab = 'tD/cD'
      Endif
c
      Rewind(unit=18)
      Write(18,*) Toplabel
      Write(18,*) Xaxislab
      Write(18,*) Yaxislab
c
      Do 10 i=1,4
         Write(18,6000) Kauto(i)
   10 Continue
c
      Write(18,6000) Lcateg
      Write(18,*) '0'
c
      IF(Lcateg.eq.2) then
         Write(18,*) '-1'
      Else
         Write(18,*) '0'
      Endif
c
      Write(18,*) '1'
      Write(18,6000) Npp
c
      Do 20 i=1,npp
         Write(18,6200) tdtycu(i), dmtycu(i)
   20 Continue
c
      Write(18,*) '0  10  5'
c
      IF(Lcateg.NE.2) then
         Write(18,*) '1'
         Write(18,6000) Npp
         Do 30 i=1,npp
            Write(18,6200) tdtycu(i), tcderiv(i)
   30    Continue
         Write(18,*) '0  10  5'
      Endif
 6000 Format(I5)
 6200 Format(2(E9.4,2X))
      Return
      End
c     ********************************************************
```

Appendix D.

Source Code of C Program for User Interface

RGF.EXE is a copy of FRAN.EXE which is described here. FRAN.EXE is built by using the NMK command under Windows SDK. The maintenance file called FRAN contains the following instructions:

```
all: fran.exe

fran.res: fran.rc fran.h filemgr.h input.h scope.h
     rc -r fran.rc

fran.obj: fran.c fran.h filemgr.h input.h scope.h
      cl -c -AM -Gsw -Od -Zipe -W3 fran.c

filemgr.obj: filemgr.c fran.h filemgr.h input.h scope.h
      cl -c -AM -Gsw -Od -Zipe -W3 filemgr.c


fran.exe: fran.obj filemgr.obj fran.def
      link  @fran.l
      rc -K fran.res

fran.exe: fran.res
      rc fran.res
```

This appendix presents the source code of the following C programs: FRAN.C, FRAN.H, FRAN.MAP, FRAN.L, FRAN.DLG, FRAN.RC, FRAN.DEF, FILEMGR.C, FILEMGR.H, INPUT.H, SCOPE.H, SCOPE.DLG in this order.

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

    PROGRAM: fran.c

    PURPOSE: Main source for application

    FUNCTIONS:

   WinMain() - calls initialization function, processes message loop
   InitApplication() - initializes window data and registers window
   InitInstance() - saves instance handle and creates main window
   MainWndProc() - processes messages
   About() - processes messages for "About" dialog box

    COMMENTS:

*****************************************************************************/

#include "windows.h"          /* required for all Windows applications */
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <math.h>
#include "fran.h"             /* specific to this program              */
#include "input.h"
#include "filemgr.h"
#include "scope.h"

#define  NUMPTS  1000

#define  XAXIS   1
#define  YAXIS   2

#define  LOG       1
#define  CARTESIAN 2

#define  FIELD 1
#define  REGULAR 2

HANDLE hInst;                 /* current instance                      */

POINT  Points[1000];

int    XInit;
int    YInit;
float  XScale;
float  YScale;

HWND   hWndMain;
HMENU  hMenu;
int    GraphType;

HANDLE hMemAsciiPts;
HANDLE hMemLoadPts;
LPSTR  pAsciiPts;
int far *  pIntPts;
LPSTR  pLoadPts;
LPSTR  pCurPts;
int    NumPts;
int    XNumTickPts;
int    YNumTickPts;
char   XTickValue[20][20];
char   YTickValue[20][20];

char   Col1Details[256];
char   Col2Details[256];
char   Col3Details[256];
char   Col4Details[256];
char   Col5Details[256];
char   Col6Details[256];
char   Col7Details[256];
char   Col8Details[256];
char   Col9Details[256];
char   Col10Details[256];
char   Col11Details[256];
char   Col12Details[256];

char   MDHDetails[80] = "Pseudo-pressure vs. time (hours)";
char   LogDetails[80] = "Delta mp vs. delta time (hours)";
char   LogDerivDetails[80] = "Delta mp vs. delta time (hours)";
char   CartesianDetails[80] ="Pseudo-pressure vs. time (hours)";
char   HornerDetails[80] = "Pseudo-pressure vs. (tp + delta t)/ delta t";
char   DimensionlessDetails[80] = "Dimensionless mp vs. dimensionless time";
char   PressureDetails[80] = "Pressure (psi) vs.  time (hours)";
char   LineDetails[80] = "Dimensionless pseudo-pressure vs. TD/rD^2";
char   RateDetails[80] = "Flow rate (Mcf/day) vs. time (hours)";


char   XMinLabel[20];
char   YMinLabel[20];
char   XMaxLabel[20];
char   YMaxLabel[20];

int    XDlogMouse;
int    YDLogMouse;

float  XMinFloat;
float  XMaxFloat;
float  YMinFloat;
float  YMaxFloat;

int    FileMode;

void   CreatePoints(void);
int    GetMaxY(void);
void   FormatFloat(char *, double);
void   DrawScales(HWND, int, int, double, double);
void   NextTick(HDC, int, int, int);
void   WriteInputFile(int, HWND);
void   WriteInputElement(int, HWND, int);
void   EraseWindow(HWND);
void   ReadInputFile(int, HWND);
void   ReadInputElement(int, HWND, int);
void   ReadDataFile(void);
int    ReadNextInt(int);
int    ReadNextPt(int);
void   SetPoints(void);
void   DrawPoints(HWND);
int    NumPoints(int);
void   GetDetailLine(int, char *);
void   SetupParms(int, char *);
int    GetNextToken(char *, char *);
void   DrawAxes(void);
void   DoStatusLine(HWND, LONG, int);
void   UpdateStatusBox(int);
float  ConvertPt(int, int, int);
void   MarkerDraw(HDC, int, int);
```

```
void    PrintVars(HDC);
void    PrintAxes(HDC);
void    PrintPoints(HWND);
HDC     GetDefPrinterDC(void);

extern char         OpenName[128];

/*****************************************************************************

    FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)

    PURPOSE: calls initialization function, processes message loop

    COMMENTS:


*****************************************************************************/

int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE          hInstance;   /* current instance                    */
HANDLE          hPrevInstance;/* previous instance                   */
LPSTR           lpCmdLine;   /* command line                        */
int             nCmdShow;    /* show-window type (open/icon)        */
{
    MSG             msg;     /* message                             */

    if (!hPrevInstance)      /* Other instances of app running?     */
       if (!InitApplication(hInstance)) /* Initialize shared things */
          return (FALSE);    /* Exits if unable to initialize       */

                             /* Perform initializations that apply to */
                             /* a specific instance                 */

    if (!InitInstance(hInstance, nCmdShow))
        return (FALSE);

                             /* Acquire and dispatch messages until a */
                             /*         WM_QUIT message is received. */

    while (GetMessage(&msg,  /* message structure                   */
           NULL,             /* handle of window receiving the message*/
           NULL,             /* lowest message to examine           */
           NULL))            /* highest message to examine          */
          {
          TranslateMessage(&msg);/* Translates virtual key codes     */
          DispatchMessage(&msg);/* Dispatches message to window      */
          }
    return (msg.wParam);     /* Returns the value from PostQuitMessage.*/
}


/*****************************************************************************

    FUNCTION: InitApplication(HANDLE)

    PURPOSE: Initializes window data and registers window class

    COMMENTS:


*****************************************************************************/

BOOL InitApplication(hInstance)
HANDLE          hInstance;   /* current instance                    */
{
    WNDCLASS  wc;

                             /* Fill in window class structure with  */
                             /* parameters that describe the         */
                             /* main window.                         */

    wc.style = NULL;         /* Class style(s).                     */
    wc.lpfnWndProc = MainWndProc;/* Function to retrieve messages for  */
                             /* windows of this class.              */
    wc.cbClsExtra = 0;       /* No per-class extra data.            */
    wc.cbWndExtra = 0;       /* No per-window extra data.           */
    wc.hInstance = hInstance;/* Application that owns the class.    */
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName =  "GenericMenu";/* Name of menu resource in .RC file. */
    wc.lpszClassName = "GenericWClass";/* Name used in call to CreateWindow. */

                             /* Register the window class and return */
                             /*                success/failure code. */

    return (RegisterClass(&wc));

}


/*****************************************************************************

    FUNCTION:  InitInstance(HANDLE, int)

    PURPOSE:  Saves instance handle and creates main window

    COMMENTS:

        This function is called at initialization time for every instance of
        this application.  This function performs initialization tasks that
        cannot be shared by multiple instances.

        In this case, we save the instance handle in a static variable and
        create and display the main program window.

*****************************************************************************/

BOOL InitInstance(hInstance, nCmdShow)
HANDLE          hInstance;   /* Current instance identifier.        */
int             nCmdShow;    /* Param for first ShowWindow() call.  */
{
    HWND            hWnd;    /* Main window handle.                 */
    int             cxDesk;
    int             cyDesk;

                             /* Save the instance handle in static */
                             /*   variable, which will be used in  */
                             /* many subsequence calls from this   */
                             /* application to Windows.            */

    hInst = hInstance;

                             /* Create a main window for this       */
                             /*            application instance.    */

    cyDesk = GetSystemMetrics(SM_CYSCREEN);
```

```
    cxDesk = GetSystemMetrics(SM_CXSCREEN);

    hWnd = CreateWindow(
        "GenericWClass",     /* See RegisterClass() call.          */
        "Real Gas Flow",/* Text for window title bar.      */
        WS_OVERLAPPEDWINDOW, /* Window style.                      */
        0,                   /* Default horizontal position.       */
        0,                   /* Default vertical position.         */
        cxDesk,              /* Default width.                     */
        cyDesk,              /* Default height.                    */
        NULL,                /* Overlapped windows have no parent. */
        NULL,                /* Use the window class menu.         */
        hInstance,           /* This instance owns this window.    */
        NULL                 /* Pointer not needed.                */
    );

                             /* If window could not be created,    */
                             /*                  return "failure" */

    if (!hWnd)
        return (FALSE);

    hWndMain = hWnd;

                             /* Make the window visible; update its*/
                             /*  client area; and return "success" */

    ShowWindow(hWnd, nCmdShow);/* Show the window                  */
    UpdateWindow(hWnd);      /* Sends WM_PAINT message             */
    return (TRUE);           /* Returns the value from PostQuitMessage*/

}

/****************************************************************************

    FUNCTION: MainWndProc(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages

    COMMENTS:

        To process the IDM_ABOUT message, call MakeProcInstance() to get the
        current instance address of the About() function.  Then call Dialog
        box which will create the box according to the information in your
        generic.rc file and turn control over to the About() function. When
        it returns, free the intance address.

****************************************************************************/

long FAR PASCAL MainWndProc(hWnd, message, wParam, lParam)
HWND            hWnd;       /* window handle                  */
unsigned        message;    /* type of message                */
WORD            wParam;     /* additional information         */
LONG            lParam;     /* additional information         */
{
   FARPROC          lpProcAbout;/* pointer to the "About" function */
   FARPROC          lpProcParam;/* pointer to Input Parameters function */
   FARPROC          lpScopeDlg;
   FARPROC          lpOpenDlg  ;
   BOOL             rc;
   int              delx;
   int              dely;
   WNDCLASS         wc;
   char             buf[256];

   switch (message)
      {
      case WM_CREATE:
         hMemAsciiPts = 0;
         XInit = 0;
         YInit = 0;
         XScale = 1.0;
         YScale = 1.0;

         delx = GetSystemMetrics(SM_CXFULLSCREEN);
         dely = GetSystemMetrics(SM_CYFULLSCREEN);

         hInputBut = CreateWindow("Button", "Input",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.02 * delx),
                        (int)(0.02 * dely),
                        75,
                        20,
                        hWnd,
                        IDC_INPUTBUT, hInst, NULL);

         hRunBut = CreateWindow("Button", "Run",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.14 * delx),
                        (int)(0.02 * dely),
                        75,
                        20,
                        hWnd,
                        IDC_RUNBUT, hInst, NULL);

         hSelectBut = CreateWindow("Button", "Output",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.26 * delx),
                        (int)(0.02 * dely),
                        75,
                        20,
                        hWnd,
                        IDC_SELECTBUT, hInst, NULL);

         hVarsBut = CreateWindow("Button", "Plot Menu",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.38  * delx),
                        (int)(0.02 * dely),
                        75,
                        20,
                        hWnd,
                        IDC_VARSBUT, hInst, NULL);

//       hScopeBut = CreateWindow("Button", "Scope",
//                      WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
//                      (int)(0.50  * delx),
//                      (int)(0.02 * dely),
//                      75,
//                      20,
//                      hWnd,
//                      IDC_SCOPEBUT, hInst, NULL);

         hGraphBut = CreateWindow("Button", "Graph",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.50  * delx),
                        (int)(0.02 * dely),
                        75,
                        20,
```

```
                        hWnd,
                        IDC_GRAPHBUT, hInst, NULL);
/*@@*/      hPrintBut = CreateWindow("Button", "Print",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.62 * delx),
                        (int)(0.02 * dely),
                        75,
                        20,
                        hWnd,
                        IDC_PRINTBUT, hInst, NULL);

          hEraseBut = CreateWindow("Button", "Erase",
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        (int)(0.74  * delx),
                        (int)(0.02 * dely),
//                      (int)(0.74  * delx),
//                      (int)(0.02 * dely),
                        75,
                        20,
                        hWnd,
                        IDC_ERASEBUT, hInst, NULL);

           wc.style = NULL;
           wc.lpfnWndProc = GraphProc;
           wc.cbClsExtra = 0;
           wc.cbWndExtra = 0;
           wc.hInstance = hInst;
           wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
           wc.hCursor = LoadCursor(NULL, IDC_ARROW);
           wc.hbrBackground = GetStockObject(WHITE_BRUSH);
           wc.lpszMenuName =  NULL;
           wc.lpszClassName = "GraphClass";

           RegisterClass(&wc);

           hGraphWnd =  CreateWindow(
                          "GraphClass",
                          "",
                          WS_CHILD | WS_BORDER | WS_VISIBLE,
                          50,
                          50,
                          550,
                          350,
                          hWnd,
                          NULL,
                          hInst,
                          NULL);
           hMenu = CreatePopupMenu();
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_MDH, (LPSTR)"Miller-Dyes-Hutchin
son Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_LOGLOG, (LPSTR)"Log-Log Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_LOGLOGDERIV, (LPSTR)"Log-Log Der
ivative Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_CARTESIAN, (LPSTR)"Cartesian Plo
t");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_HORNER, (LPSTR)"Horner Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_DIMENSIONLESS, (LPSTR)"Dimension
less Semi-Log Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_PRESSURE, (LPSTR)"Pressure Histo
ry Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_LINE, (LPSTR)"Line Source Soluti
on Plot");
           AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_RATEHIST, (LPSTR)"Rate History P
lot");
           break;

        case WM_COMMAND:      /*  command from application menu */
           GetWindowText(hWnd, buf, 256);
             strcpy(buf, "Real Gas Flow. ");
           switch (wParam)
              {
              case IDC_LOGLOG:
                 strcat(buf, "Log - Log Plot. ");
                 if (OpenName[0] != 0)
                    {
                    strcat (buf, "Output file : ");
                    strcat(buf, OpenName);
                    }
                 SetWindowText(hWnd, (LPSTR)buf);
                 GraphType = wParam;
                 break;

              case IDC_LOGLOGDERIV:
                 strcat(buf, "Log - Log Derivative Plot. ");
                 if (OpenName[0] != 0)
                    {
                    strcat (buf, "Output file : ");
                    strcat(buf, OpenName);
                    }
                 SetWindowText(hWnd, (LPSTR)buf);
                 GraphType = wParam;
                 break;
              case IDC_CARTESIAN:
                 strcat(buf, "Cartesian Plot. ");
                 if (OpenName[0] != 0)
                    {
                    strcat (buf, "Output file : ");
                    strcat(buf, OpenName);
                    }
                 SetWindowText(hWnd, (LPSTR)buf);
                 GraphType = wParam;
                 break;
              case IDC_HORNER:
                 strcat(buf, "Horner Plot. ");
                 if (OpenName[0] != 0)
                    {
                    strcat (buf, "Output file : ");
                    strcat(buf, OpenName);
                    }
                 SetWindowText(hWnd, (LPSTR)buf);
                 GraphType = wParam;
                 break;
              case IDC_DIMENSIONLESS:
                 strcat(buf, "Dimensionless Semi-log Plot. ");
                 if (OpenName[0] != 0)
                    {
                    strcat (buf, "Output file : ");
                    strcat(buf, OpenName);
                    }
                 SetWindowText(hWnd, (LPSTR)buf);
                 GraphType = wParam;
                 break;
              case IDC_PRESSURE:
                 strcat(buf, "Pressure History. ");
                 if (OpenName[0] != 0)
                    {
                    strcat (buf, "Output file :  ");
                    strcat(buf, OpenName);
```

```
        }
      SetWindowText(hWnd, (LPSTR)buf);
      GraphType = wParam;
      break;
   case IDC_LINE:
      strcat(buf, "Line Source Solution. ");
      if (OpenName[0] != 0)
         {
         strcat (buf, "Output file : ");
         strcat(buf, OpenName);
         }
      SetWindowText(hWnd, (LPSTR)buf);
      GraphType = wParam;
      break;
   case IDC_RATEHIST:
      strcat(buf, "Rate History. ");
      if (OpenName[0] != 0)
         {
         strcat (buf, "Output file: " );
         strcat(buf, OpenName);
         }
      SetWindowText(hWnd, (LPSTR)buf);
      GraphType = wParam;
      break;
   case IDC_MDH:
      strcat(buf, "Miller-Dyes-Hutchinson Plot. ");
      if (OpenName[0] != 0)
         {
         strcat (buf, "Output file : ");
         strcat(buf, OpenName);
         }
      SetWindowText(hWnd, (LPSTR)buf);
      GraphType = wParam;
      break;
   case IDM_ABOUT:
      lpProcAbout = MakeProcInstance(About, hInst);

      DialogBox(hInst,
                "AboutBox",
                hWnd,
                lpProcAbout);

      FreeProcInstance(lpProcAbout);
      break;

   case IDC_ERASEBUT:
      XDlogMouse = 0;
      EraseWindow(hWnd);
      break;

   case IDC_INPUTBUT:
      XDlogMouse = 0;
      lpProcParam = MakeProcInstance(InputParams, hInst);

      rc = DialogBox(hInst,
                     "PARAMS",
                     hWnd,
                     lpProcParam);
      FreeProcInstance(lpProcParam);
      break;

   case IDC_SELECTBUT:
      lpOpenDlg = MakeProcInstance((FARPROC) OpenSave, hInst);
      rc = DialogBox(hInst, "OPEN", hWnd, lpOpenDlg);
      FreeProcInstance(lpOpenDlg);
      if ((strstr(OpenName, ".FLD") == NULL) &&
          (strstr(OpenName, ".fld") == NULL))
         FileMode = REGULAR;
      else
         FileMode = FIELD;
      if (rc)
         {
         MessageBox (
                     hWndMain,
                     (LPSTR)"About to read & convert data points. This may requ
ire a few seconds to process.",
                     (LPSTR)"Output File",
                     MB_OK);
         ReadDataFile();
         strcpy (buf, "Real Gas Flow. ");
         if (OpenName[0] != 0)
            {
            strcat (buf, "Output file : ");
            strcat (buf, OpenName);
            }
         SetWindowText(hWndMain, (LPSTR)buf);
         }
      break;

   case IDC_SCOPEBUT:
      lpScopeDlg = MakeProcInstance((FARPROC) ScopeParams, hInst);
      DialogBox(hInst, "SCOPE", hWnd, lpScopeDlg);
      FreeProcInstance(lpScopeDlg);
      break;
/*@@*/
   case IDC_PRINTBUT:
      PrintIt = TRUE;
      DrawPoints(hGraphWnd);
      break;

   case IDC_GRAPHBUT:
      XDlogMouse = 0;

//    DrawWindow(hGraphWnd);
/*@@*/
      PrintIt = FALSE;
      DrawPoints(hGraphWnd);
//    DrawBorder(hWnd);
      break;

   case IDC_RUNBUT:
      WinExec((LPSTR)"FRANC.EXE", FALSE);
      break;

   case IDC_VARSBUT:
      GraphType = 0;
      DrawMenuBar(hWnd);
      TrackPopupMenu(hMenu, 0, 285, 52, 0, hWnd, (LPVOID)NULL);
      DrawMenuBar(hWnd);
      break;
   }
break;

case WM_DESTROY:      /*  window being destroyed */
   PostQuitMessage(0);
   break;
}
```

```
   return (DefWindowProc(hWnd, message, wParam, lParam));
}

/****************************************************************************

    FUNCTION: GraphProc()

    PURPOSE:  Processes messages

    COMMENTS:


****************************************************************************/

long FAR PASCAL GraphProc(hWnd, message, wParam, lParam)
HWND            hWnd;       /* window handle                    */
unsigned        message;    /* type of message                  */
WORD            wParam;     /* additional information           */
LONG            lParam;     /* additional information           */
{

    switch (message)
       {
       case WM_CREATE:
          break;

       case WM_MOUSEMOVE:
         if (wParam != MK_LBUTTON)
            break;

       case WM_LBUTTONUP:
                           /* Handle vertical status line info for current  */
                           /* position.                                     */
         DoStatusLine(hWnd, lParam, R2_NOT);
         break;
       }

   return (DefWindowProc(hWnd, message, wParam, lParam));
}


/****************************************************************************

    FUNCTION: About(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages for "About" dialog box

    COMMENTS:

        No initialization is needed for this particular dialog box, but TRUE
        must be returned to Windows.

        Wait for user to click on "Ok" button, then close the dialog box.

****************************************************************************/

BOOL FAR PASCAL About(hDlg, message, wParam, lParam)
HWND            hDlg;       /* window handle of the dialog box */
unsigned        message;    /* type of message                 */
WORD            wParam;     /* message-specific information    */
LONG            lParam;
{
   switch (message)
      {
      case WM_INITDIALOG:    /* message: initialize dialog box */
         return (TRUE);
         break;

      case WM_COMMAND:       /* message: received a command    */
         if (wParam == IDOK  /* "OK" box selected?             */
             || wParam == IDCANCEL)
            {                /* System menu close command?     */
            EndDialog(hDlg, TRUE);/* Exits the dialog box      */
            return (TRUE);
            }
         break;
      }

   return (FALSE);           /* Didn't process a message       */
}

/****************************************************************************

    FUNCTION: InputParams(HWND, unsigned, WORD, LONG)

    PURPOSE:

    COMMENTS:

****************************************************************************/

BOOL FAR PASCAL InputParams(hDlg, message, wParam, lParam)
HWND            hDlg;       /* window handle of the dialog box */
unsigned        message;    /* type of message                 */
WORD            wParam;     /* message-specific information    */
LONG            lParam;
{
   char            buf[80];
   char            buf2[80];
   OFSTRUCT        ofstruct;
   int             hdl;
   FARPROC         lpOpenDlg  ;
   BOOL            rc;
   int             len;

   switch (message)
      {
      case WM_CREATE:
          SendDlgItemMessage(hDlg, IDC_TYPE, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_RATE, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_OUTER, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_DAMAGED, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_THICKNESS, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_WELLDEPTH, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_TURBULENCE, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_MAIN, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_WELLHEAD, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_WELL, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_POROSITY, EM_LIMITTEXT, 40, NULL);


          SendDlgItemMessage(hDlg, IDC_HS2, EM_LIMITTEXT, 40, NULL);

          SendDlgItemMessage(hDlg, IDC_RESTEMP, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_SHUTIN, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_SKIN, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_SPECGRAV, EM_LIMITTEXT, 40, NULL);
          SendDlgItemMessage(hDlg, IDC_CO2, EM_LIMITTEXT, 40, NULL);
```

```
        SendDlgItemMessage(hDlg, IDC_CALIF, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hDlg, IDC_RESPRES, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hDlg, IDC_PERMAB, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hDlg, IDC_WELLBORE, EM_LIMITTEXT, 40, NULL);

        SendDlgItemMessage(hDlg, IDC_AUTO, EM_LIMITTEXT, 40, NULL);
        break;
    case WM_COMMAND:        /*  received a command    */
        switch (wParam)
            {
            case IDC_OPENFILE:
                lpOpenDlg = MakeProcInstance((FARPROC) OpenSave, hInst);
                rc = DialogBox(hInst, "OPEN", hDlg, lpOpenDlg);
                FreeProcInstance(lpOpenDlg);
                if (rc)
                    {
                    hdl = OpenFile((LPSTR)OpenName,
                                (LPOFSTRUCT)&ofstruct,OF_READWRITE);
                    if (hdl == -1)
                        {
                        strcpy(buf2, "Unable to create the file ");
                        strcat(buf2, OpenName);
                        MessageBox(hDlg, (LPSTR)buf2,
                             NULL, MB_OK | MB_ICONHAND);
                        EndDialog(hDlg, FALSE);
                        }
                    ReadInputFile(hdl, hDlg);
                    }
                break;

            case IDC_OK:
                if (ValidateEntries(hDlg))
                    {
                    GetDlgItemText(hDlg, IDC_MAIN, (LPSTR)buf, 80);
                    strcat(buf, ".MAI");
                    hdl = OpenFile((LPSTR)buf,
                                (LPOFSTRUCT)&ofstruct, OF_CREATE | OF_READWRITE);
                    if (hdl == -1)
                        {
                        strcpy(buf2, "Unable to create the file ");
                        strcat(buf2, buf);
                        MessageBox(hDlg, (LPSTR)buf2,
                             NULL, MB_OK | MB_ICONHAND);
                        EndDialog(hDlg, FALSE);
                        }
                    WriteInputFile(hdl, hDlg);
                    _lclose(hdl);


                    hdl = OpenFile((LPSTR)"INPUT.FIL",
                                (LPOFSTRUCT)&ofstruct, OF_CREATE | OF_READWRITE);
                    len = strlen(buf);
                    _lwrite(hdl, (LPSTR)buf, len+1);
                    _lclose(hdl);


                    EndDialog(hDlg, TRUE);/* Exits the dialog box      */
                    }
                return (TRUE);
                break;

            case IDC_CANCEL:
                EndDialog(hDlg, FALSE);/* Exits the dialog box      */
                return (TRUE);
                break;
            }
        break;
    }

    return (FALSE);          /* Didn't process a message       */
}

/*************************************************************************

    FUNCTION: ScopeParams(HWND, unsigned, WORD, LONG)

    PURPOSE:

    COMMENTS:

*************************************************************************/

BOOL FAR PASCAL ScopeParams(hDlg, message, wParam, lParam)
HWND            hDlg;       /* window handle of the dialog box */
unsigned        message;    /* type of message                 */
WORD            wParam;     /* message-specific information    */
LONG            lParam;
{
    char            buf[80];

    switch (message)
        {
        case WM_INITDIALOG:
            itoa(XInit, buf, 10);
            SetDlgItemText(hDlg, SCOPE_INITXENTRY, buf);
            itoa(YInit, buf, 10);
            SetDlgItemText(hDlg, SCOPE_INITYENTRY, buf);
            buf[0] = 0;
            FormatFloat(buf, XScale);
//          itoa(XScale, buf, 10);
            SetDlgItemText(hDlg, SCOPE_SCALEXENTRY, buf);
//          itoa(YScale, buf, 10);
            buf[0] = 0;
            FormatFloat(buf, YScale);
            SetDlgItemText(hDlg, SCOPE_SCALEYENTRY, buf);
            break;

        case WM_COMMAND:        /*  received a command    */
            switch (wParam)
                {
                case SCOPE_OK:
                    GetDlgItemText(hDlg, SCOPE_INITXENTRY, (LPSTR)buf, 80);
                    if (buf[0] != 0)
                        XInit = atoi(buf);
                    GetDlgItemText(hDlg, SCOPE_INITYENTRY, (LPSTR)buf, 80);
                    if (buf[0] != 0)
                        YInit = atoi(buf);
                    GetDlgItemText(hDlg, SCOPE_SCALEXENTRY, (LPSTR)buf, 80);
                    if (buf[0] != 0)
                        XScale = atof(buf);
                    GetDlgItemText(hDlg, SCOPE_SCALEYENTRY, (LPSTR)buf, 80);
                    if (buf[0] != 0)
                        YScale = atof(buf);
                    EndDialog(hDlg, TRUE);/* Exits the dialog box      */
                    return (TRUE);
                    break;

                case SCOPE_CANCEL:
```

```
            EndDialog(hDlg, FALSE);/* Exits the dialog box      */
            return (TRUE);
            break;
          }
       break;
     }

   return (FALSE);          /* Didn't process a message       */
}

/************************************************************************

   FUNCTION: ValidateEntries(void)

   PURPOSE:

   COMMENTS:

************************************************************************/

BOOL ValidateEntries(hDlg)
HWND            hDlg;
{
   if (!(CheckNoBlanks(hDlg)))
      return(FALSE);

   return(TRUE);
}

/************************************************************************

   FUNCTION: CheckNoBlanks(void)

   PURPOSE:

   COMMENTS:

************************************************************************/

BOOL CheckNoBlanks(hDlg)
HWND            hDlg;
{
// if (CheckText(hDlg, IDC_TITLE, "No title of test entered") == FALSE)
//    return(FALSE);

   if (CheckText(hDlg, IDC_MAIN, "No main file entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_RATE, "No rate file entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_TYPE, "No type of test entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_OUTER, "No outer boundary entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_RADIUS, "No boundary radius entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_DAMAGED, "No damaged zone radius entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_WELL, "No well radius entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_WELLDEPTH, "No well depth entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_THICKNESS, "No thickness entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_POROSITY, "No porosity entered") == FALSE)
      return(FALSE);



   if (CheckText(hDlg, IDC_SPECGRAV, "No specific gravity entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_HS2, "No H2S content entered") == FALSE)
      return(FALSE);


   if (CheckText(hDlg, IDC_CO2, "No CO2 content entered") == FALSE)
      return(FALSE);



   if (CheckText(hDlg, IDC_CALIF, "No California/condensate entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_RESTEMP, "No Res. temperature entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_WELLHEAD, "No well-head temperature entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_SHUTIN, "No Shut-in head pressure entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_RESPRES, "No initial res. pressure entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_SKIN, "No skin factor entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_PERMAB, "No initial permeability entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_TURBULENCE, "No turbulence indicator entered") == FALSE)
      return(FALSE);



   if (CheckText(hDlg, IDC_WELLBORE, "No Field Data File entered") == FALSE)
      return(FALSE);

   if (CheckText(hDlg, IDC_AUTO, "No automate curve indicator entered") == FALSE)
      return(FALSE);
```

```
   return(TRUE);
}


/*****************************************************************************

    FUNCTION: CheckText()

    PURPOSE:

    COMMENTS:

*****************************************************************************/

BOOL CheckText(hDlg, id, errstr)
HWND            hDlg;
int             id;
char *          errstr;
{
   char            buf[80];

   if (GetDlgItemText(hDlg, id, (LPSTR)buf,
                      80) == 0)
      {
      MessageBox (
                  hDlg,
                  (LPSTR)errstr,
                  (LPSTR)"Parameters Entry",
                  MB_OK | MB_ICONHAND);
      SetFocus(GetDlgItem(hDlg, id));
      return(FALSE);
      }

   return(TRUE);
}

/*****************************************************************************

    FUNCTION: DrawWindow()

    PURPOSE:

    COMMENTS:

*****************************************************************************/

void DrawWindow(hWnd)
HWND            hWnd;
{
   HDC             hDC;
   RECT            rect;
   int             delx;
   int             dely;
   int             maxY;
   float           xfact;
   float           yfact;
   HPEN            hPen;
   HPEN            hOldPen;

   hDC =  GetDC(hWnd);

   GetClientRect(hWnd, &rect);
   delx = rect.right - rect.left;
   dely = rect.bottom - rect.top;

// FillRect(hDC, &rect, GetStockObject(WHITE_BRUSH));

   CreatePoints();
   maxY = GetMaxY();

   SetMapMode(hDC, MM_ANISOTROPIC);
   xfact = 1000.0 / XScale;
   if (xfact < 0)
      xfact = 30000.0;
   yfact = ((float)(maxY - YInit)) / YScale;
   if (yfact < 0)
      yfact = 30000.0;
   SetWindowExt(hDC,  (int)xfact,  (int)yfact);
   SetWindowOrg(hDC, XInit, YInit);
   SetViewportExt(hDC, delx, -dely);
   SetViewportOrg(hDC, 0, dely);

   hPen = CreatePen(PS_SOLID, 1, RGB(0,0,255));
   hOldPen = SelectObject(hDC, hPen);

   Polyline(hDC, (LPPOINT)Points, NUMPTS);

   SelectObject(hDC, hOldPen);
   DeleteObject(hPen);

   DrawScales(hWndMain, XInit, YInit, xfact, yfact);

   ReleaseDC (hWnd, hDC);
}
/*@@*/
/*-------------------------------------------------------------------------.
|                                                                          |
| Function Name - PrintVars()                                              |
|                                                                          |
| Description   - Prints out an individual graph.                          |
|                                                                          |
| Parameters    - HDC        hDC    ; Handle of device context to the printer.|
|                                                                          |
 -------------------------------------------------------------------------*/
void PrintVars(hDC)
HDC             hDC;
{
   int far *       tmpptr;
   int             i;
   int             pgpts;
   int             j;
   int             k;
   int             ypt;
   BOOL            found;
   char            buf[255];
   HPEN            hPen;
   HPEN            hPenOrg;
   int             delx;
   int             winy;
   int             winx;
   int             dely;
   int             pg;
   BOOL            first;

                           /* Create pen handle                        */
                           /* graphs.                                  */
   hPen = GetStockObject(BLACK_PEN);

                           /* Start by selecting the       pen         */
```

```
   hPenOrg = SelectObject(hDC, hPen);
   SelectObject(hDC, hPenOrg);

   delx= GetDeviceCaps(hDC, HORZRES);
   dely= GetDeviceCaps(hDC, VERTRES);
   SetMapMode(hDC, MM_ANISOTROPIC);
   winx =  10000;
   winy =  10000;

                              /* Window is logical coordinates & extents      */
   SetWindowExt(hDC, winx , winy);
   SetWindowOrg(hDC, 0, 0);

   Escape(hDC, NEWFRAME, NULL, NULL, NULL);

                              /* Make view port slighlty less than printer    */
                              /* resolution                                   */
   SetViewportExt(hDC,(int)(0.9 * (float)delx),(int)(0.9 * (float)-dely));
   SetViewportOrg(hDC, (int)(0.05 * (float)delx), (int)( 0.95 * (float)dely));

   Rectangle(hDC, 0,0, winx, winy);

                              /* Coordinate systems set up. Print the points */
   first = TRUE;
   for (i = 0; i < NumPts - 1; i++)
       {
       if (first)
          {
          MoveTo(hDC , Points[i].x, Points[i].y);
          first = FALSE;
          }
       else
          LineTo(hDC,(int)Points[i].x, Points[i].y);
       }

                              /* Make view port bigger now                    */
   SetViewportExt(hDC,(int)((float)delx),(int)((float)-dely));
   SetViewportOrg(hDC, (int)0, (int)((float)dely));

   PrintAxes(hDC);

   SelectObject(hDC, hPenOrg);
}


/*@@*/
/*-------------------------------------------------------------------------.
|                                                                          |
| Function Name - GetDefPrinterDC()                                        |
|                                                                          |
| Description   -  Retrieves printer device from Win.ini and opens a context |
|                  to it.                                                  |
|                                                                          |
| Parameters    -  None.                                                   |
|                                                                          |
|                                                                          |
|                                                                          |
 -------------------------------------------------------------------------*/
HDC  GetDefPrinterDC()
{
   HDC             hDC;
   int             cch;
   int             i;
   PSTR            pDriver;
   PSTR            pPort;
   PSTR            pPrinter;
   char            buf[80];
   char            print_buf[80];

                              /* Get printer device reference from WIN.INI    */
   cch = GetProfileString("windows", "device", NULL, buf, 80);


                              /* Parse out details of port,printer & driver   */
   pDriver = pPrinter = &buf[0];

   for (i = 0; i < cch; i++, pDriver++)
      {
      if (*pDriver == ',')
         {
         *pDriver = '\0';
         pDriver++;
         i++;
         break;
         }
      }

   for (pPort = pDriver; i < cch; i++, pPort++)
      {
      if (*pPort == ',')
         {
         *pPort = '\0';
         pPort++;
         break;
         }
      }


                              /* Get handle to device                         */
   hDC = CreateDC(pDriver, pPrinter, pPort, NULL);


   return(hDC);
}

/*************************************************************************

    FUNCTION: PrintPoints()

    PURPOSE:

    COMMENTS:

*************************************************************************/
/*@@*/
void PrintPoints(hWnd)
HWND        hWnd;
{
   int             rc;
   float           sclx;
   float           scly;
   int             delx;
   int             dely;
   HDC             hDCPrint;
   BOOL            ret;


                              /* Set up a device handle to the printer        */
```

```
   hDCPrint = NULL;
   hDCPrint =  GetDefPrinterDC();
   if (hDCPrint == NULL)
      {
      MessageBox(hWnd, "Unable to print. GetDefPrinterDC() failed", "Print", MB_OK);
      return;
      }

                              /* Refer to Microsoft manual for printing       */
                              /* protocol.                                    */
                              /* First, do a "start document".                */
   rc = Escape(hDCPrint, STARTDOC, 6, "sample", NULL);
   if (rc <= 0)
      {
      MessageBox(hWnd, "Unable to print. Escape func(STARTDOC) failed", "Print", MB_OK);
      return;
      }

   delx= GetDeviceCaps(hDCPrint, HORZRES);
   dely= GetDeviceCaps(hDCPrint, VERTRES);

                              /* Print whatever variables are configured.     */
   PrintVars(hDCPrint);

                              /* All done, wrap things up.                    */

   rc = Escape(hDCPrint, NEWFRAME, NULL, NULL, NULL);
   rc = Escape(hDCPrint, ENDDOC, NULL, NULL, NULL);


   ret = DeleteDC (hDCPrint);
   if (ret == FALSE)
      {
      MessageBox(hWnd, "DeleteDC failed", "Print", MB_OK);
      }
}

/*****************************************************************************

    FUNCTION: DrawPoints()

    PURPOSE:

    COMMENTS:

*****************************************************************************/

void DrawPoints(hWnd)
HWND            hWnd;
{
   HDC             hDC;
   RECT            rect;
   int             delx;
   int             dely;
   int             maxX;
   int             maxY;
   float           xfact;
   float           yfact;
   HPEN            hPen;
   HPEN            hOldPen;
   int             i;
   int             j;

   GetClientRect(hWnd, &rect);
   delx = rect.right - rect.left;
   dely = rect.bottom - rect.top;

   SetPoints();

   if (PrintIt == TRUE)
      {
      PrintPoints(hWnd);
      return;
      }

   maxY = 10000;
   maxX = 10000;

   hDC = GetDC(hWnd);
   SetMapMode(hDC, MM_ANISOTROPIC);
   xfact = (float)(maxX) / XScale;
   yfact = (float)(maxY) / YScale;
   SetWindowExt(hDC,  (int)xfact,  (int)yfact);
   SetWindowOrg(hDC, 0, 0);
   SetViewportExt(hDC, delx, -dely);
   SetViewportOrg(hDC, 0, dely);

   hPen = CreatePen(PS_SOLID, 1, RGB(0,0,255));
   hOldPen = SelectObject(hDC, hPen);

// Polyline(hDC, (LPPOINT)Points, NumPts); /*10-02-91*/

   if (FileMode != FIELD)
      Polyline(hDC, (LPPOINT)Points, NumPts-1);
   else
      {
      for (i = 0; i < NumPts - 1; i++)
         {
         MoveTo(hDC , Points[i].x, Points[i].y);
         SetPixel(hDC, Points[i].x, Points[i].y, RGB(255, 0, 0));
         for (j = 0; j < 40; j++)
            SetPixel(hDC, Points[i].x + j, Points[i].y, RGB(255, 0, 0));

         for (j = 0; j < 40; j++)
            SetPixel(hDC, Points[i].x - j, Points[i].y, RGB(255, 0, 0));

         for (j = 0; j < 40; j++)
            SetPixel(hDC, Points[i].x, Points[i].y + j, RGB(255, 0, 0));

         for (j = 0; j < 40; j++)
            SetPixel(hDC, Points[i].x, Points[i].y - j, RGB(255, 0, 0));
         }
      }
   SelectObject(hDC, hOldPen);
   DeleteObject(hPen);

// DrawScales(hWndMain, XInit, YInit, xfact, yfact);
   DrawAxes();

   ReleaseDC (hWnd, hDC);
}

/*****************************************************************************

    FUNCTION: EraseWindow()

    PURPOSE:
```

```
    COMMENTS:

****************************************************************************/

void EraseWindow(hWnd)
HWND            hWnd;
{
   HDC             hDC;
   RECT            rect;
   int             delx;
   int             dely;
   int             maxY;
   float           xfact;
   float           yfact;

   hDC =  GetDC(hWnd);

   GetClientRect(hWnd, &rect);
   FillRect(hDC, &rect, GetStockObject(WHITE_BRUSH));
   InvalidateRect(hWnd, (LPRECT)NULL, FALSE);
   UpdateWindow(hWnd);

   ReleaseDC (hWnd, hDC);
}

/****************************************************************************

    FUNCTION: DrawBorder()

    PURPOSE:

    COMMENTS:

****************************************************************************/

void DrawBorder(hWnd)
HWND            hWnd;

{
   int             delx;
   int             dely;
   HDC             hDC;

   delx= GetDeviceCaps(hDC, HORZRES);
   dely= GetDeviceCaps(hDC, VERTRES);
   hDC =  GetDC(hWnd);
// Rectangle(hDC, (int)(0.1 * delx),(int)(0.15 * dely),
//                (int)(0.9 * delx),(int)(0.9 * dely));
   Rectangle(hDC, 50, 50, 620, 400);
   ReleaseDC (hWnd, hDC);
}

/****************************************************************************

    FUNCTION: CreatePoints()

    PURPOSE:

    COMMENTS:

****************************************************************************/

void CreatePoints()
{
   int   i;
   int   y;

   for (i = 0; i < NUMPTS; i++)
      {
      y = rand();
      Points[i].x = i;
      Points[i].y = y;
      }

}

/****************************************************************************

    FUNCTION: GetMaxY()

    PURPOSE:

    COMMENTS:

****************************************************************************/

int  GetMaxY()

{
   int   i;
   int   ymax;


   ymax = 0;
   for (i = 0; i < NUMPTS; i++)
      {
      if (ymax < Points[i].y)
         ymax = Points[i].y;
      }

   return(ymax);

}

/****************************************************************************

    FUNCTION: FormatFloat()

    PURPOSE:

    COMMENTS:

****************************************************************************/
void  FormatFloat(recstr, fltval)
char *          recstr;
double          fltval;
{
   char            utilbuf[80];
   char *          fptr;
   char *          utilptr;
   int             dec;
   int             sign;
   int             len;

                           /* Utility to format floating pt number  into   */
                           /* a string.                                    */
   fptr = fcvt((float)fltval, 2, &dec, &sign);
```

```
    if (fptr != 0)
       {
       if (sign)
          {
          utilbuf[0] = '-';
          utilptr = &utilbuf[1];
          }
       else
          utilptr = &utilbuf[0];

       len = strlen(fptr);
       if ((len != 2) && (len != 1))
          {
          memcpy (utilptr, fptr, len - 2);
          utilptr[len-2] = '.';
          utilptr[len-1] = 0;
          strcat(utilptr, &fptr[len -2]);
          strcat(recstr, utilbuf);
          }
       else
          {
          if (len == 1)
             {
             strcat(recstr, "0.0");
             strcat(recstr, fptr);
             }
          else
             {
             strcat(recstr, "0.");
             strcat(recstr, fptr);
             }
          }
       strcat(recstr, "  ");
       }

}

/****************************************************************************

    FUNCTION: DrawScales()

    PURPOSE:

    COMMENTS:

****************************************************************************/
void   DrawScales(hWnd, x1, y1, xfact, yfact)
HWND            hWnd;
int             x1;
int             y1;
double          xfact;
double          yfact;

{
   HDC             hDC;
   char            buf[80];
   int             xPt;
   int             yPt;
   int             xCur;
   int             yCur;
   int             i;

   hDC =  GetDC(hWnd);

   xCur = 45;
   xPt = x1;
   for (i = 0; i <= 10; i++)
       {
       NextTick(hDC, xCur, xPt, XAXIS);
       xCur += 55;
       xPt += (int)(xfact * 0.1);
       }

   yCur = 390;
   yPt = y1;
   for (i = 0; i <= 10; i++)
       {
       NextTick(hDC, yCur, yPt, YAXIS);
       yCur -= 35;
       yPt += (int)(yfact * 0.1);
       }

/*
   itoa(x1, buf, 10);
   TextOut(hDC, 45, 420, buf, strlen(buf));
   itoa(x1+(int)xfact, buf, 10);
   TextOut(hDC, 595,420, buf, strlen(buf));

   itoa(y1, buf, 10);
   TextOut(hDC, 10, 390, buf, strlen(buf));
   itoa(y1+(int)yfact, buf, 10);
   TextOut(hDC, 10, 40, buf, strlen(buf));
*/
   ReleaseDC (hWnd, hDC);
}

/****************************************************************************

    FUNCTION: PrintAxes()

    PURPOSE:

    COMMENTS:

****************************************************************************/
void   PrintAxes(hDC)
HDC      hDC;
{
   char            buf[80];
   int             xPt;
   int             yPt;
   int             xCur;
   int             yCur;
   int             xCur2;
   int             yCur2;
   int             i;
   int             delta;
   int             delta2;
   char *          xDetails;
   char *          yDetails;
   char *          hdrDetails;
   int             delx;
   int             dely;


   xDetails = Col2Details;
   yDetails = Col10Details;
```

```
switch (GraphType)
   {
   case IDC_LOGLOGDERIV:
      xDetails = Col2Details;
      yDetails = Col12Details;
      hdrDetails = LogDerivDetails;
      break;
   case IDC_LOGLOG:
      xDetails = Col2Details;
      yDetails = Col10Details;
      hdrDetails = LogDetails;
      break;
   case IDC_CARTESIAN:
      xDetails = Col1Details;
      yDetails = Col7Details;
      hdrDetails = CartesianDetails;
      break;
   case IDC_HORNER:
      xDetails = Col4Details;
      yDetails = Col7Details;
      hdrDetails = HornerDetails;
      break;
   case IDC_DIMENSIONLESS:
      xDetails = Col3Details;
      yDetails = Col8Details;
      hdrDetails = DimensionlessDetails;
      break;
   case IDC_PRESSURE:
      xDetails = Col1Details;
      yDetails = Col6Details;
      hdrDetails = PressureDetails;
      break;
   case IDC_LINE:
      xDetails = Col5Details;
      yDetails = Col9Details;
      hdrDetails = LineDetails;
      break;
   case IDC_RATEHIST:
      xDetails = Col1Details;
      yDetails = Col11Details;
      hdrDetails = RateDetails;

      break;
   case IDC_MDH:
      xDetails = Col2Details;
      yDetails = Col7Details;
      hdrDetails = MDHDetails;
      break;
   }

SetupParms(XAXIS, xDetails);
SetupParms(YAXIS, yDetails);

      /* FRANCOIS !!! - adjust x,y, delx, dely below as needed */

      /* Draws out title, xmin, xmax, ymin, and ymax values    */
xCur = 80;
yCur = 28;
TextOut(hDC, xCur, yCur, hdrDetails, strlen(hdrDetails));

xCur = 45;
yCur = 420;
TextOut(hDC, xCur, yCur, XMinLabel, strlen(XMinLabel));

   xCur = 600;
   yCur = 420;
   TextOut(hDC, xCur, yCur, XMaxLabel, strlen(XMaxLabel));

   xCur = 5;
   yCur = 390;
   TextOut(hDC, xCur, yCur, YMinLabel, strlen(YMinLabel));

   xCur = 5;
   yCur = 40;
   TextOut(hDC, xCur, yCur, YMaxLabel, strlen(YMaxLabel));

   delta = 555 / (XNumTickPts + 1);
   xCur = 45;
   yCur = 420;

   delx = 10000;
   dely = 10000;
   xCur2 = 0;

            /* Draws out X tick values, and vert line representing the tick */
   delta2 = delx / (XNumTickPts + 1);
   for (i = 0; i < XNumTickPts; i++)
      {
      xCur = xCur + delta;
      xCur2 = xCur2 + delta2;
      TextOut(hDC, xCur, yCur, XTickValue[i], strlen(XTickValue[i]));
      MoveTo(hDC, xCur2, 500);
      LineTo(hDC, xCur2, 9500);

      }

            /* Draws Y tick values, and horz line representing the tick    */
   xCur = 5;
   yCur = 390;
   yCur2 = 0;
   delta = -(350 / (YNumTickPts + 1));
   delta2 = -(dely / (YNumTickPts + 1));
   for (i = 0; i < YNumTickPts; i++)
      {
      yCur = yCur + delta;
      yCur2 = yCur2 + delta2;
      TextOut(hDC, xCur, yCur, YTickValue[i], strlen(YTickValue[i]));
      MoveTo(hDC, 500, yCur2);
      LineTo(hDC, 9500, yCur2);
      }

}

/*****************************************************************************

    FUNCTION: DrawAxes()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void   DrawAxes()

{
   HDC            hDC;
   HDC            hDC2;
   char           buf[80];
```

```
int             xPt;
int             yPt;
int             xCur;
int             yCur;
int             xCur2;
int             yCur2;
int             i;
HWND            hWnd;
int             delta;
int             delta2;
char *          xDetails;
char *          yDetails;
char *          hdrDetails;
RECT            rect;
int             delx;
int             dely;

hWnd = hWndMain;

hDC =  GetDC(hWnd);
hDC2 =  GetDC(hGraphWnd);

xDetails = Col2Details;
yDetails = Col10Details;

switch (GraphType)
   {
   case IDC_LOGLOGDERIV:
      xDetails = Col2Details;
      yDetails = Col12Details;
      hdrDetails = LogDerivDetails;
      break;
   case IDC_LOGLOG:
      xDetails = Col2Details;
      yDetails = Col10Details;
      hdrDetails = LogDetails;
      break;
   case IDC_CARTESIAN:
      xDetails = Col1Details;
      yDetails = Col7Details;
      hdrDetails = CartesianDetails;
      break;
   case IDC_HORNER:
      xDetails = Col4Details;
      yDetails = Col7Details;
      hdrDetails = HornerDetails;
      break;
   case IDC_DIMENSIONLESS:
      xDetails = Col3Details;
      yDetails = Col8Details;
      hdrDetails = DimensionlessDetails;
      break;
   case IDC_PRESSURE:
      xDetails = Col1Details;
      yDetails = Col6Details;
      hdrDetails = PressureDetails;
      break;
   case IDC_LINE:
      xDetails = Col5Details;
      yDetails = Col9Details;
      hdrDetails = LineDetails;
      break;
   case IDC_RATEHIST:
      xDetails = Col1Details;
      yDetails = Col11Details;
      hdrDetails = RateDetails;

      break;
   case IDC_MDH:
      xDetails = Col2Details;
      yDetails = Col7Details;
      hdrDetails = MDHDetails;
      break;
   }

SetupParms(XAXIS, xDetails);
SetupParms(YAXIS, yDetails);

xCur = 80;
yCur = 28;
TextOut(hDC, xCur, yCur, hdrDetails, strlen(hdrDetails));

xCur = 45;
yCur = 420;
TextOut(hDC, xCur, yCur, XMinLabel, strlen(XMinLabel));

xCur = 600;
yCur = 420;
TextOut(hDC, xCur, yCur, XMaxLabel, strlen(XMaxLabel));

xCur = 5;
yCur = 390;
TextOut(hDC, xCur, yCur, YMinLabel, strlen(YMinLabel));

xCur = 5;
yCur = 40;
TextOut(hDC, xCur, yCur, YMaxLabel, strlen(YMaxLabel));

delta = 555 / (XNumTickPts + 1);
xCur = 45;
yCur = 420;

GetClientRect(hGraphWnd, &rect);
delx = rect.right - rect.left;
dely = rect.bottom - rect.top;
xCur2 = rect.left;

delta2 = delx / (XNumTickPts + 1);
for (i = 0; i < XNumTickPts; i++)
   {
   xCur = xCur + delta;
   xCur2 = xCur2 + delta2;
   TextOut(hDC, xCur, yCur, XTickValue[i], strlen(XTickValue[i]));
   MoveTo(hDC2, xCur2, rect.bottom);
   LineTo(hDC2, xCur2, rect.top);

   }

xCur = 5;
yCur = 390;
yCur2 = rect.bottom;
delta = -(350 / (YNumTickPts + 1));
delta2 = -(dely / (YNumTickPts + 1));
for (i = 0; i < YNumTickPts; i++)
   {
   yCur = yCur + delta;
   yCur2 = yCur2 + delta2;
   TextOut(hDC, xCur, yCur, YTickValue[i], strlen(YTickValue[i]));
```

```
      MoveTo(hDC2, rect.left, yCur2);
      LineTo(hDC2, rect.right, yCur2);
      }
/*
   xPt = x1;
   for (i = 0; i <= 10; i++)
       {
       NextTick(hDC, xCur, xPt, XAXIS);
       xCur += 55;
       xPt += (int)(xfact * 0.1);
       }

   yCur = 390;
   yPt = y1;
   for (i = 0; i <= 10; i++)
       {
       NextTick(hDC, yCur, yPt, YAXIS);
       yCur -= 35;
       yPt += (int)(yfact * 0.1);
       }
*/
   ReleaseDC (hWnd, hDC);
   ReleaseDC (hGraphWnd, hDC2);
}

/*****************************************************************************

    FUNCTION: SetupParms()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void   SetupParms(axis, line)
int            axis;
char *         line;
{
   HDC            hDC;
   char           buf[80];
   int            xPt;
   int            yPt;
   int            xCur;
   int            yCur;
   int            i;
   char *         lptr;
   int            cnt;

   lptr = line;

   cnt = GetNextToken(lptr, buf);
   if (axis == XAXIS)
      {
      strcpy (XMinLabel, buf);
      XMinFloat = atof (XMinLabel);
      }
   else
      {
      strcpy (YMinLabel, buf);
      YMinFloat = atof (YMinLabel);
      }

   lptr = lptr + cnt;
   cnt = GetNextToken(lptr, buf);

   if (axis == XAXIS)
      {
      strcpy (XMaxLabel, buf);
      XMaxFloat = atof (XMaxLabel);
      }
   else
      {
      strcpy (YMaxLabel, buf);
      YMaxFloat = atof (YMaxLabel);
      }
   lptr = lptr + cnt;
   cnt = GetNextToken(lptr, buf);

   if (axis == XAXIS)
      XNumTickPts = atoi(buf);
   else
      YNumTickPts = atoi(buf);

   lptr = lptr + cnt;

   for (i = 0; i < XNumTickPts; i++)
      {
      cnt = GetNextToken(lptr, buf);
      lptr = lptr + cnt;
      if (axis == XAXIS)
         {
         strcpy(XTickValue[i], buf);
         }
      else
         {
         strcpy(YTickValue[i], buf);
         }
      }
}

/*****************************************************************************

    FUNCTION: GetNextToken()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
int   GetNextToken(curptr, resbuf)
char *         curptr;
char *         resbuf;
{

   char           buf[10];
   char *         ptr;
   int            cnt;

   ptr = curptr;
   cnt = 0;

   while(TRUE)
      {
      if ((*ptr != '-') &&
          (*ptr != '+') &&
          (*ptr != '.') &&
          (*ptr != 'E') &&
          (*ptr != '0') &&
          (*ptr != '1') &&
```

```
            (*ptr != '2') &&
            (*ptr != '3') &&
            (*ptr != '4') &&
            (*ptr != '5') &&
            (*ptr != '6') &&
            (*ptr != '7') &&
            (*ptr != '8') &&
            (*ptr != '9'))
           break;
        *resbuf = *ptr;
        resbuf++;
        ptr++;
        cnt++;
        }
    while (TRUE)
        {
        if ((*ptr != '-') &&
            (*ptr != '+') &&
            (*ptr != '.') &&
            (*ptr != 'E') &&
            (*ptr != '0') &&
            (*ptr != '1') &&
            (*ptr != '2') &&
            (*ptr != '3') &&
            (*ptr != '4') &&
            (*ptr != '5') &&
            (*ptr != '6') &&
            (*ptr != '7') &&
            (*ptr != '8') &&
            (*ptr != '9'))
           {
           cnt++;
           ptr++;
           }
        else
           break;
        }
    *resbuf = 0;

    return (cnt);

}
/*****************************************************************************

    FUNCTION: NextYTick()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void    NextTick(hDC, pos, val, axis)
HDC             hDC;
int             pos;
int             val;
int             axis;
{
    char            buf[80];

    itoa(val, buf, 10);
    if (axis == YAXIS)
       TextOut(hDC, 10, pos, buf, strlen(buf));
    if(axis == XAXIS)
       TextOut(hDC, pos, 420, buf, strlen(buf));
}

/*****************************************************************************

    FUNCTION: WriteInputFile()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void    WriteInputFile(hdl, hDlg)
int             hdl;
HWND            hDlg;
{

    char            buf[80];
/*
    WriteInputElement(hdl, hDlg, IDC_TITLE);
    WriteInputElement(hdl, hDlg, IDC_MAIN);
    WriteInputElement(hdl, hDlg, IDC_RATE);
    WriteInputElement(hdl, hDlg, IDC_TYPE);
    WriteInputElement(hdl, hDlg, IDC_OUTER);
    WriteInputElement(hdl, hDlg, IDC_RADIUS);
    WriteInputElement(hdl, hDlg, IDC_DAMAGED);
    WriteInputElement(hdl, hDlg, IDC_WELL);
    WriteInputElement(hdl, hDlg, IDC_WELLDEPTH);
    WriteInputElement(hdl, hDlg, IDC_THICKNESS);
    WriteInputElement(hdl, hDlg, IDC_POROSITY);

    WriteInputElement(hdl, hDlg, IDC_SPECGRAV);
    WriteInputElement(hdl, hDlg, IDC_HS2);
    WriteInputElement(hdl, hDlg, IDC_CO2);

    WriteInputElement(hdl, hDlg, IDC_CALIF);
    WriteInputElement(hdl, hDlg, IDC_RESTEMP);
    WriteInputElement(hdl, hDlg, IDC_SHUTIN);
    WriteInputElement(hdl, hDlg, IDC_RESPRES);
    WriteInputElement(hdl, hDlg, IDC_SKIN);
    WriteInputElement(hdl, hDlg, IDC_PERMAB);
    WriteInputElement(hdl, hDlg, IDC_TURBULENCE);

    WriteInputElement(hdl, hDlg, IDC_WELLHEAD);
*/


    WriteInputElement(hdl, hDlg, IDC_TITLE);
    WriteInputElement(hdl, hDlg, IDC_MAIN);
    WriteInputElement(hdl, hDlg, IDC_RATE);
    WriteInputElement(hdl, hDlg, IDC_TYPE);
    WriteInputElement(hdl, hDlg, IDC_OUTER);
    WriteInputElement(hdl, hDlg, IDC_RADIUS);
    WriteInputElement(hdl, hDlg, IDC_DAMAGED);
    WriteInputElement(hdl, hDlg, IDC_WELL);
    WriteInputElement(hdl, hDlg, IDC_WELLDEPTH);
    WriteInputElement(hdl, hDlg, IDC_THICKNESS);
    WriteInputElement(hdl, hDlg, IDC_POROSITY);

    WriteInputElement(hdl, hDlg, IDC_SPECGRAV);
    WriteInputElement(hdl, hDlg, IDC_HS2);
    WriteInputElement(hdl, hDlg, IDC_CO2);

    WriteInputElement(hdl, hDlg, IDC_CALIF);
    WriteInputElement(hdl, hDlg, IDC_RESTEMP);
```

```
   WriteInputElement(hdl, hDlg, IDC_WELLHEAD);
   WriteInputElement(hdl, hDlg, IDC_SHUTIN);
   WriteInputElement(hdl, hDlg, IDC_RESPRES);
   WriteInputElement(hdl, hDlg, IDC_SKIN);
   WriteInputElement(hdl, hDlg, IDC_PERMAB);
   WriteInputElement(hdl, hDlg, IDC_TURBULENCE);


   WriteInputElement(hdl, hDlg, IDC_WELLBORE);
   WriteInputElement(hdl, hDlg, IDC_AUTO);

   _lclose(hdl);

}

/****************************************************************************

    FUNCTION: WriteInputElement()

    PURPOSE:

    COMMENTS:

****************************************************************************/
void   WriteInputElement(hdl, hDlg, id)
int             hdl;
HWND            hDlg;
int             id;
{

   char            buf[80];
   int             len;

   GetDlgItemText(hDlg, id, (LPSTR)buf, 80);
   if (buf[0] != 0)
      {
//    strcat(buf, "\n\r");
      strcat(buf, "\n");
      }
   else
      {
//    strcpy(buf, "\n\r");
      strcpy(buf, "\n");
      }

   len = strlen(buf);
   _lwrite(hdl, (LPSTR)buf, len);

}

/****************************************************************************

    FUNCTION: ReadInputFile()

    PURPOSE:

    COMMENTS:

****************************************************************************/
void   ReadInputFile(hdl, hDlg)
int             hdl;
HWND            hDlg;
{
   char            buf[80];
/*
   ReadInputElement(hdl, hDlg, IDC_TITLE);
   ReadInputElement(hdl, hDlg, IDC_MAIN);
   ReadInputElement(hdl, hDlg, IDC_RATE);
   ReadInputElement(hdl, hDlg, IDC_TYPE);
   ReadInputElement(hdl, hDlg, IDC_OUTER);
   ReadInputElement(hdl, hDlg, IDC_RADIUS);
   ReadInputElement(hdl, hDlg, IDC_DAMAGED);
   ReadInputElement(hdl, hDlg, IDC_WELL);
   ReadInputElement(hdl, hDlg, IDC_WELLDEPTH);
   ReadInputElement(hdl, hDlg, IDC_THICKNESS);
   ReadInputElement(hdl, hDlg, IDC_POROSITY);

   ReadInputElement(hdl, hDlg, IDC_SPECGRAV);
   ReadInputElement(hdl, hDlg, IDC_HS2);
   ReadInputElement(hdl, hDlg, IDC_CO2);

   ReadInputElement(hdl, hDlg, IDC_CALIF);
   ReadInputElement(hdl, hDlg, IDC_RESTEMP);
   ReadInputElement(hdl, hDlg, IDC_SHUTIN);
   ReadInputElement(hdl, hDlg, IDC_RESPRES);
   ReadInputElement(hdl, hDlg, IDC_SKIN);
   ReadInputElement(hdl, hDlg, IDC_PERMAB);
   ReadInputElement(hdl, hDlg, IDC_TURBULENCE);

   ReadInputElement(hdl, hDlg, IDC_PROPERTIES);
*/
   ReadInputElement(hdl, hDlg, IDC_TITLE);
   ReadInputElement(hdl, hDlg, IDC_MAIN);
   ReadInputElement(hdl, hDlg, IDC_RATE);
   ReadInputElement(hdl, hDlg, IDC_TYPE);
   ReadInputElement(hdl, hDlg, IDC_OUTER);
   ReadInputElement(hdl, hDlg, IDC_RADIUS);
   ReadInputElement(hdl, hDlg, IDC_DAMAGED);
   ReadInputElement(hdl, hDlg, IDC_WELL);
   ReadInputElement(hdl, hDlg, IDC_WELLDEPTH);
   ReadInputElement(hdl, hDlg, IDC_THICKNESS);
   ReadInputElement(hdl, hDlg, IDC_POROSITY);

   ReadInputElement(hdl, hDlg, IDC_SPECGRAV);
   ReadInputElement(hdl, hDlg, IDC_HS2);
   ReadInputElement(hdl, hDlg, IDC_CO2);

   ReadInputElement(hdl, hDlg, IDC_CALIF);
   ReadInputElement(hdl, hDlg, IDC_RESTEMP);
   ReadInputElement(hdl, hDlg, IDC_WELLHEAD);
   ReadInputElement(hdl, hDlg, IDC_SHUTIN);
   ReadInputElement(hdl, hDlg, IDC_RESPRES);
   ReadInputElement(hdl, hDlg, IDC_SKIN);
   ReadInputElement(hdl, hDlg, IDC_PERMAB);
   ReadInputElement(hdl, hDlg, IDC_TURBULENCE);


   ReadInputElement(hdl, hDlg, IDC_WELLBORE);
   ReadInputElement(hdl, hDlg, IDC_AUTO);


   _lclose(hdl);

}

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

```
    FUNCTION: ReadInputElement()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void   ReadInputElement(hdl, hDlg, id)
int            hdl;
HWND           hDlg;
int            id;
{

   char            buf[80];
   char            buf2[80];
   int             len;
   int             cnt;

   buf[0];
   cnt = 0;
   while (TRUE)
      {
      len = _lread(hdl,buf, 1);
//    if ((buf[0] == '\r') || (buf[0] == '\n') ||(len == 0))
//    if ((buf[0] == '\r') ||(len == 0))
      if ((buf[0] == '\n') ||(len == 0))
         {
         break;
         }
      if (buf[0] != ' ')
         {
         buf2[cnt] = buf[0];
         cnt++;
         }
      }
   buf2[cnt] = 0;
   SetDlgItemText(hDlg, id, buf2);

}

/*****************************************************************************

    FUNCTION: ReadDataFile()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void   ReadDataFile()
{
   int             hdl;
   OFSTRUCT        OfStruct;
   int             cnt;
   int             datapt;
   int             i;
   int             j;
   BOOL            set;
   char            buf[20];
   int             len;
   LONG            rembytes;
   DWORD           allocamt;
   LONG            k;

   set = FALSE;

   hdl = OpenFile((LPSTR)OpenName, (LPOFSTRUCT)&OfStruct, OF_READ);
   if (hdl == -1)
      MessageBox (
                  hWndMain,
                  (LPSTR)"hdl == NULL",
                  (LPSTR)"Alloc",
                  MB_OK | MB_ICONHAND);

// if ((strstr(OpenName, ".FLD") == NULL) &&
//     (strstr(OpenName, ".fld") == NULL))
//    FileMode == REGULAR;
// else
//    FileMode == FIELD;

   NumPts = NumPoints(hdl);

   GetDetailLine(hdl, Col1Details);
   GetDetailLine(hdl, Col2Details);
   GetDetailLine(hdl, Col3Details);
   GetDetailLine(hdl, Col4Details);
   GetDetailLine(hdl, Col5Details);
   GetDetailLine(hdl, Col6Details);
   GetDetailLine(hdl, Col7Details);
   GetDetailLine(hdl, Col8Details);
   GetDetailLine(hdl, Col9Details);
   GetDetailLine(hdl, Col10Details);
   GetDetailLine(hdl, Col11Details);
   GetDetailLine(hdl, Col12Details);

   if (hMemAsciiPts)
      {
      GlobalUnlock(hMemAsciiPts);
      GlobalFree(hMemAsciiPts);
      }

   hMemAsciiPts = GlobalAlloc(GMEM_MOVEABLE,
                              ((DWORD)NumPts * (DWORD)24) + (DWORD)100);
   if (hMemAsciiPts == NULL)
      MessageBox (
                  hWndMain,
                  (LPSTR)"hMemAsciiPts NULL",
                  (LPSTR)"Alloc",
                  MB_OK | MB_ICONHAND);
   pAsciiPts = GlobalLock(hMemAsciiPts);
   if (pAsciiPts == NULL)
      MessageBox (
                  hWndMain,
                  (LPSTR)"pAsciiPts NULL",
                  (LPSTR)"Alloc",
                  MB_OK | MB_ICONHAND);

   rembytes = (LONG)((LONG)NumPts * (LONG)12 * (LONG)6);
   allocamt = (DWORD)rembytes + (DWORD)1500;
   hMemLoadPts = GlobalAlloc(GMEM_MOVEABLE | GMEM_NOT_BANKED, allocamt);

   if (hMemLoadPts == NULL)
      MessageBox (
                  hWndMain,
                  (LPSTR)"hMemLoadPts NULL",
                  (LPSTR)"Alloc",
                  MB_OK | MB_ICONHAND);
   pLoadPts = GlobalLock(hMemLoadPts);
```

```
   if (pLoadPts == NULL)
      MessageBox (
                  hWndMain,
                  (LPSTR)"pLoadPts NULL",
                  (LPSTR)"Alloc",
                  MB_OK | MB_ICONHAND);


   pIntPts = (int far *)pAsciiPts;

   pCurPts = pLoadPts;

   len = _lread(hdl, (LPSTR)pLoadPts, ((NumPts) * 12 * 6) + 1000);

   for (i = 0; i < NumPts; i++)
      {
      for (j = 0; j < 12; j++)
         {
         datapt = ReadNextPt(hdl);

         *pIntPts = datapt;
         pIntPts++;

         }
      }

   GlobalUnlock(hMemLoadPts);
   GlobalFree(hMemLoadPts);

   _lclose(hdl);

   MessageBox (
               hWndMain,
               (LPSTR)"Completed reading of output file",
               (LPSTR)" ",
               MB_OK);
}

/*****************************************************************************

    FUNCTION: ReadNextInt()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
int   ReadNextInt(hdl)
int             hdl;
{
   char             tmpbuf[2];
   char             buf[80];
   int              i;
   int              val;

   while(TRUE)
      {
      _lread(hdl, tmpbuf, 1);
      if ((tmpbuf[0] >= '0') && (tmpbuf[0] <= '9'))
         break;
      }

   buf[0] = tmpbuf[0];

   i = 1;
   while(TRUE)
      {
      _lread(hdl, tmpbuf, 1);
      if ((tmpbuf[0] < '0') || (tmpbuf[0] > '9'))
         break;
      buf[i] = tmpbuf[0];
      i++;
      }
   buf[i] = 0;

   val = atoi(buf);

   return(val);
}

/*****************************************************************************

    FUNCTION: ReadNextPt()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
int   ReadNextPt(hdl)
int             hdl;
{
   char             tmpbuf[2];
   char             buf[80];
   int              i;
   int              val;

   while(TRUE)
      {
      tmpbuf[0] = *pLoadPts;
      pLoadPts++;
      if ((tmpbuf[0] >= '0') && (tmpbuf[0] <= '9'))
         break;
      }

   buf[0] = tmpbuf[0];

   i = 1;
   while(TRUE)
      {
      tmpbuf[0] = *pLoadPts;
      pLoadPts++;
      if ((tmpbuf[0] < '0') || (tmpbuf[0] > '9'))
         break;
      buf[i] = tmpbuf[0];
      i++;
      }
   buf[i] = 0;

   val = atoi(buf);

   return(val);
}

/*****************************************************************************

    FUNCTION: GetDetailLine()
```

```
    PURPOSE:

    COMMENTS:

*****************************************************************************/
void  GetDetailLine(hdl, outline)
int             hdl;
char  *         outline;
{
   char            buf[256];
   char            tmpbuf[2];
   int             i;

   while(TRUE)
      {
      _lread(hdl, tmpbuf, 1);
      if ( ((tmpbuf[0] <= '9') && (tmpbuf[0] >= '0')) ||
           (tmpbuf[0] == '.'))
         break;
      }

   buf[0] = tmpbuf[0];

   i = 1;
   while(TRUE)
      {
      _lread(hdl, tmpbuf, 1);
      if (tmpbuf[0] == '\r')
         break;
      buf[i] = tmpbuf[0];
      i++;
      }
   buf[i] = 0;
   strcpy(outline, buf);
/*
   MessageBox (
            hWndMain,
            (LPSTR)outline,
            (LPSTR)"Alloc",
            MB_OK | MB_ICONHAND);
*/
}


/*****************************************************************************

    FUNCTION: SetPoints()

    PURPOSE:

    COMMENTS:

*****************************************************************************/
void  SetPoints()
{
   int             i;
   static int      graph;
   int             xoffset;
   int             yoffset;
   char            buf[20];
   int             xpt;
   int             ypt;

   xoffset = 1;
   yoffset = 7;

   switch (GraphType)
      {
      case IDC_LOGLOG:
         xoffset = 2 ;
         yoffset =10 ;
         break;
      case IDC_LOGLOGDERIV:
         xoffset = 2 ;
         yoffset =12 ;
         break;
      case IDC_CARTESIAN:
         xoffset = 1 ;
         yoffset = 7 ;
         break;
      case IDC_HORNER:
         xoffset = 4 ;
         yoffset = 7 ;
         break;
      case IDC_DIMENSIONLESS:
         xoffset = 3 ;
         yoffset = 8 ;
         break;
      case IDC_PRESSURE:
         xoffset = 1 ;
         yoffset = 6 ;
         break;
      case IDC_LINE:
         xoffset = 5 ;
         yoffset = 9 ;
         break;
      case IDC_RATEHIST:
         xoffset = 1 ;
         yoffset = 11 ;
         break;
      case IDC_MDH:
         xoffset = 2 ;
         yoffset = 7 ;
         break;
      }

   pIntPts = (int far *)pAsciiPts;


   for (i = 0; i < NumPts; i++)
       {
       xpt  = (int)(((float)(*(pIntPts + xoffset - 1))) * XScale);
       ypt  = (int)(((float)(*(pIntPts + yoffset - 1))) * YScale);
       Points[i].x = xpt + Xinit;
       Points[i].y = ypt + Yinit;



       pIntPts = pIntPts + 12;
       }

// GlobalUnlock(hMemAsciiPts);
}

*****************************************************************************

    FUNCTION: NumPoints()
```

```
    PURPOSE:

    COMMENTS:

*****************************************************************************/
int   NumPoints(hdl)
int             hdl;
{
   return(ReadNextInt(hdl));
}

/*****************************************************************************

  Function Name - DoStatusLine()

  Description   - Draw vertical line indicating current position as reflected
                  by the mouse.

  Parameters    - HWND        hWnd   ; Window handle to draw into.
                  LONG        lParam ; Value containing x- and y- mouse pos.
                  int         rop    ; Indicates logical drawing mode.

*****************************************************************************/
void DoStatusLine(hWnd, lParam, rop)
HWND            hWnd;
LONG            lParam;
int             rop;
{
   HDC             hDC2;
   RECT            rect2;
// HPEN            hPen;
// HPEN            hOldPen;

                           /* Get device context to draw into.            */
   hDC2 =  GetDC(hWnd);

// hPen = CreatePen(PS_SOLID, 1, RGB(255,0, 0));
// hOldPen = SelectObject(hDC2, hPen);

                           /* Set drawing mode as reflected by input      */
   SetROP2(hDC2, rop);

// SetROP2(hDC2, R2_MASKNOTPEN);

   GetClientRect(hWnd, &rect2);
   if (XDlogMouse != 0)
      {
                           /* Move to bottom x-pos on bottom of rect, then  */
                           /* draw a vertical line to top of drawing rect.  */
                           /* so as to erase the old line.                  */
      MoveTo(hDC2, XDlogMouse,0);
//    LineTo(hDC2, XDlogMouse, YDLogMouse);
      LineTo(hDC2, XDlogMouse, rect2.bottom);
      }

                           /* Same as above, but draw in the new line.      */
   XDlogMouse = LOWORD(lParam);
   YDLogMouse = HIWORD(lParam);


// SetROP2(hDC2, R2_MASKPEN);

   MoveTo(hDC2, XDlogMouse,0);
   YDLogMouse = rect2.bottom;
   LineTo(hDC2, XDlogMouse, YDLogMouse);

// SelectObject(hDC2, hOldPen);
// DeleteObject(hPen);

   ReleaseDC(hWnd, hDC2);

                           /* Write out digit values reflecting graph inter- */
                           /* sections with new vertical line.               */
   UpdateStatusBox(XDlogMouse);
}

/*****************************************************************************

  Function Name - UpdateStatusBox()

  Description   - Draw vertical line indicating current position as reflected
                  by the mouse.

  Parameters    - HWND        hWnd   ; Window handle to draw into.
                  LONG        lParam ; Value containing x- and y- mouse pos.
                  int         rop    ; Indicates logical drawing mode.

*****************************************************************************/
void UpdateStatusBox(xpt)
int             xpt;
{
   int          dp;
   float        xConv;
   float        yConv;
   int          i;
   HDC          hDC;
   char         buf[40];
   char         buf2[40];
   int          x;
   int          y;
   int          xmode;
   int          ymode;

   dp = (int)(((float)xpt * (float)10000) / ((float)550));
   for (i = 0; i < NumPts; i++)
      {
      if ( ((Points[i].x > dp) && (GraphType != IDC_HORNER)) ||
           ((Points[i].x < dp) && (GraphType == IDC_HORNER)) )
         {
         switch (GraphType)
            {
            case IDC_LOGLOG:
               xmode = LOG;
               ymode = LOG;
               break;
            case IDC_LOGLOGDERIV:
               xmode = LOG;
               ymode = LOG;
               break;
            case IDC_CARTESIAN:
               xmode = CARTESIAN;
               ymode = CARTESIAN;
               break;
            case IDC_HORNER:
               xmode = LOG;
               ymode = CARTESIAN;
               break;
            case IDC_DIMENSIONLESS:
```

```
                xmode = LOG;
                ymode = CARTESIAN;
                break;
             case IDC_PRESSURE:
                xmode = CARTESIAN;
                ymode = CARTESIAN;
                break;
             case IDC_LINE:
                xmode = LOG;
                ymode = LOG;
                break;
             case IDC_RATEHIST:
                xmode = CARTESIAN;
                ymode = CARTESIAN;
                break;
             case IDC_MDH:
                xmode = LOG;
                ymode = CARTESIAN;
                break;
             }
          xConv = ConvertPt(XAXIS, xpt, xmode);
          yConv = ConvertPt(YAXIS, Points[i].y, ymode);

          hDC =  GetDC(hWndMain);

          strcpy(buf, "Cur. X : ");
//        itoa(xConv, buf2, 10);
//        buf2[0] = 0;
//        FormatFloat(buf2, xConv);
          gcvt(xConv, 6, buf2);
          strcat(buf, buf2);
          x = 475;
          y = 10;
          TextOut(hDC, x, y, (LPSTR)"                                    ", 40);
          TextOut(hDC, x, y, (LPSTR)buf, strlen(buf));

          strcpy(buf, "Cur. Y : ");
//        itoa(yConv, buf2, 10);
//        buf2[0] = 0;
//        FormatFloat(buf2, yConv);
          gcvt(yConv, 6, buf2);
          strcat(buf, buf2);
          y = 30;
          TextOut(hDC, x, y, (LPSTR)"                                    ", 40);
          TextOut(hDC, x, y, (LPSTR)buf, strlen(buf));

          ReleaseDC (hWndMain, hDC);
          break;
          }
       }
}

/*****************************************************************************

  Function Name - ConvertPt()

  Description   -


  Parameters    -

*****************************************************************************/
float ConvertPt(axis, pt, mode)
int             axis;
int             pt;
int             mode;
{
   float            conval;
   int              intval;
   float            delmaxmin;
   float            delmaxminlog;
   double           a;
   float            logval;

   if (mode == CARTESIAN)
      {
      if (axis == XAXIS)
         {
         delmaxmin = XMaxFloat - XMinFloat;
         conval = ((((float)pt * delmaxmin)/ (float)550) +
                        XMinFloat);
         }
      else
         {
         delmaxmin = YMaxFloat - YMinFloat;
         conval = ((((float)pt * delmaxmin)/ (float)10000) +
                        YMinFloat);
         }
      return (conval);
      }

   if (mode == LOG)
      {
      if (axis == XAXIS)
         {
         delmaxminlog = log10(XMaxFloat) - log10(XMinFloat);
         a = ((((float)pt * delmaxminlog)/ (float)550) +
                        log10(XMinFloat));
         }
      else
         {
         delmaxminlog = log10(YMaxFloat) - log10(YMinFloat);
         a = ((((float)pt * delmaxminlog)/ (float)10000) +
                        log10(YMinFloat));
         }
      logval = (float)pow(10, a);
      return(logval);
      }

}

/*****************************************************************************

  Function Name - MarkerDraw()

  Description   -


  Parameters    -

*****************************************************************************/
void  MarkerDraw(hDC, x, y)
HDC             hDC;
int             x;
int             y;
{
   DWORD dwColor;
   int i;
```

```
   dwColor = GetPixel (hDC, x, y);
   dwColor = ~dwColor;
   SetPixel (hDC, x, y, dwColor);

   for (i = 1; i < 5; i++)
      {
      dwColor = GetPixel (hDC, x, y);
      dwColor = ~dwColor;
      SetPixel (hDC, x + i, y,dwColor);

      dwColor = GetPixel (hDC, x - i, y);
      dwColor = ~dwColor;
      SetPixel (hDC, x - i, y,dwColor);

      dwColor = GetPixel (hDC, y + i, y);
      dwColor = ~dwColor;
      SetPixel (hDC, y + i, y,dwColor);

      dwColor = GetPixel (hDC, y - i, y);
      dwColor = ~dwColor;
      SetPixel (hDC, y - i, y,dwColor);

      }
}

\032
```

```
#define IDM_ABOUT 100
#define IDM_PARAM 101

#define IDC_INPUTBUT  110
#define IDC_SELECTBUT 111
#define IDC_RUNBUT    112
#define IDC_GRAPHBUT  113
#define IDC_SCOPEBUT  114
#define IDC_VARSBUT   115
#define IDC_ERASEBUT  116
/*@@*/
#define IDC_PRINTBUT  117

#define IDC_LOGLOG  120
#define IDC_LOGLOGDERIV  121
#define IDC_CARTESIAN 122
#define IDC_HORNER  123
#define IDC_DIMENSIONLESS 124
#define IDC_PRESSURE  125
#define IDC_LINE    126
#define IDC_RATEHIST 127
#define IDC_MDH      128

HWND      hInputBut;
HWND      hSelectBut;
HWND      hRunBut;
HWND      hScopeBut;
HWND      hGraphBut;
HWND      hVarsBut;
HWND      hEraseBut;
/*@@*/
HWND      hPrintBut;
HWND      hGraphWnd;

/*@@*/
BOOL      PrintIt;

int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
long FAR PASCAL MainWndProc(HWND, unsigned, WORD, LONG);
long FAR PASCAL GraphProc(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL About(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL InputParams(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL ScopeParams(HWND, unsigned, WORD, LONG);
BOOL ValidateEntries(HWND);
BOOL CheckNoBlanks(HWND);
BOOL CheckText(HWND, int, char *);
void EraseWindow(HWND);
void DrawWindow(HWND);

\032
```

```
fran

Start     Length     Name                   Class
0001:0000 03D7DH     FRAN_TEXT              CODE
0001:3D7E 004FBH     FILEMGR_TEXT           CODE
0001:4280 02866H     _TEXT                  CODE
0002:0000 00010H     NULL                   BEGDATA
0002:0010 00E32H     _DATA                  DATA
0002:0E42 0000EH     CDATA                  DATA
0002:0E50 00000H     XIFB                   DATA
0002:0E50 00000H     XIF                    DATA
0002:0E50 00000H     XIFE                   DATA
0002:0E50 00000H     XIB                    DATA
0002:0E50 00008H     XI                     DATA
0002:0E58 00000H     XIE                    DATA
0002:0E58 00000H     XPB                    DATA
0002:0E58 00000H     XP                     DATA
0002:0E58 00000H     XPE                    DATA
0002:0E58 00000H     XCB                    DATA
0002:0E58 00000H     XC                     DATA
0002:0E58 00000H     XCE                    DATA
0002:0E58 00000H     XCFB                   DATA
0002:0E58 00000H     XCF                    DATA
0002:0E58 00000H     XCFE                   DATA
0002:0E58 0000CH     DBDATA                 DATA
0002:0E64 00002H     XIOC                   DATA
0002:0E66 00088H     CONST                  CONST
0002:0EEE 00019H     HDR                    MSG
0002:0F07 00361H     MSG                    MSG
0002:1268 00002H     PAD                    MSG
0002:126A 00001H     EPAD                   MSG
0002:1270 00188H     _BSS                   BSS
0002:13F8 00000H     XOB                    BSS
0002:13F8 00000H     XO                     BSS
0002:13F8 00000H     XOE                    BSS
0002:1400 022EAH     c_common               BSS

Origin   Group
0002:0   DGROUP

Address   Export                 Alias

0001:0C32 About                  About
0001:0BB6 GraphProc              GraphProc
0001:0C9F InputParams            InputParams
0001:01B3 MainWndProc            MainWndProc
0001:3D7E OpenSave               OpenSave
0001:10F2 ScopeParams            ScopeParams
0001:4786 ___ExportedStub        ___ExportedStub

Program entry point at 0001:4291
```

```
/NOD /CO +
fran +
filemgr
,,,mlibcew libw, fran.def

\032
```

```
ABOUTBOX DIALOG LOADONCALL MOVEABLE DISCARDABLE 22, 17, 144, 75
CAPTION "About "
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU | DS_MODALFRAME
BEGIN
    CONTROL "Microsoft Windows", -1, "static", SS_CENTER | WS_GROUP | WS_CHILD, 0, 5, 14
4, 8
    CONTROL "Francois' Application", -1, "static", SS_CENTER | WS_GROUP | WS_CHILD, 0, 1
4, 144, 8
    CONTROL "Version 3.0", -1, "static", SS_CENTER | WS_GROUP | WS_CHILD, 0, 34, 144, 8
    CONTROL "OK", 1, "button", BS_DEFPUSHBUTTON | WS_GROUP | WS_TABSTOP | WS_CHILD, 53,
59, 32, 14
END

OPEN DIALOG LOADONCALL MOVEABLE DISCARDABLE 75, 60, 148, 112
CAPTION "Open "
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_SYSMENU | DS_MODALFRAME
BEGIN
    CONTROL "Open File &Name:", 400, "static", SS_LEFT | WS_GROUP | WS_CHILD, 4, 4, 100,
 10
    CONTROL "", 401, "edit", ES_LEFT | ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP | WS_CHIL
D, 4, 16, 100, 12
    CONTROL "&Files in", 402, "static", SS_LEFT | WS_GROUP | WS_CHILD, 4, 40, 32, 10
    CONTROL "", 404, "listbox", LBS_NOTIFY | WS_BORDER | WS_VSCROLL | WS_TABSTOP | WS_CH
ILD, 4, 52, 70, 56
    CONTROL "", 403, "static", SS_LEFT | WS_GROUP | WS_CHILD, 40, 40, 100, 10
    CONTROL "&Open", 1, "button", BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 87, 50, 50,
14
    CONTROL "Display", 405, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 87, 70, 50,
 14
    CONTROL "Cancel", 2, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 87, 90, 50, 14
END

SCOPE DIALOG LOADONCALL MOVEABLE DISCARDABLE 28, 41, 240, 128
CAPTION "Data Scope"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Origin X:", 501, "static", SS_RIGHT | WS_CHILD, 18, 17, 37, 9
    CONTROL "Origin Y:", 502, "static", SS_RIGHT | WS_CHILD, 21, 36, 34, 8
    CONTROL "Scale X:", 503, "static", SS_RIGHT | WS_CHILD, 21, 54, 34, 8
    CONTROL "Scale Y:", 504, "static", SS_RIGHT | WS_CHILD, 19, 72, 36, 8
    CONTROL "OK", 509, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 44, 109, 24, 14
    CONTROL "Cancel", 510, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 145, 109, 24
, 14
    CONTROL "", 505, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 15, 32, 12
    CONTROL "", 506, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 33, 32, 12
    CONTROL "", 507, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 53, 32, 12
    CONTROL "", 508, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 71, 32, 12
END

SCOPE DIALOG LOADONCALL MOVEABLE DISCARDABLE 28, 41, 240, 128
CAPTION "Data Scope"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Init X :", 501, "static", SS_RIGHT | WS_CHILD, 18, 17, 37, 9
    CONTROL "Init Y:", 502, "static", SS_RIGHT | WS_CHILD, 21, 36, 34, 8
    CONTROL "Scale X:", 503, "static", SS_RIGHT | WS_CHILD, 21, 54, 34, 8
    CONTROL "Scale Y:", 504, "static", SS_RIGHT | WS_CHILD, 19, 72, 36, 8
    CONTROL "OK", 509, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 44, 109, 24, 14
    CONTROL "Cancel", 510, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 145, 109, 24
, 14
    CONTROL "", 505, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 15, 32, 12
    CONTROL "", 506, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 33, 32, 12
    CONTROL "", 507, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 53, 32, 12
    CONTROL "", 508, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 71, 32, 12
END

SCOPE DIALOG LOADONCALL MOVEABLE DISCARDABLE 28, 41, 240, 128
CAPTION "Data Scope"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Init X :", 501, "static", SS_RIGHT | WS_CHILD, 18, 17, 37, 9
    CONTROL "Init Y:", 502, "static", SS_RIGHT | WS_CHILD, 21, 36, 34, 8
    CONTROL "Scale X:", 503, "static", SS_RIGHT | WS_CHILD, 21, 54, 34, 8
    CONTROL "Scale Y:", 504, "static", SS_RIGHT | WS_CHILD, 19, 72, 36, 8
    CONTROL "OK", 509, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 44, 109, 24, 14
    CONTROL "Cancel", 510, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 145, 109, 24,
 14
    CONTROL "", 505, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 15, 32, 12
    CONTROL "", 506, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 33, 32, 12
    CONTROL "", 507, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 53, 32, 12
    CONTROL "", 508, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 71, 32, 12
END

SCOPE DIALOG LOADONCALL MOVEABLE DISCARDABLE 28, 41, 240, 128
CAPTION "Data Scope"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Init X :", 501, "static", SS_RIGHT | WS_CHILD, 18, 17, 37, 9
    CONTROL "Init Y:", 502, "static", SS_RIGHT | WS_CHILD, 21, 36, 34, 8
    CONTROL "Scale X:", 503, "static", SS_RIGHT | WS_CHILD, 21, 54, 34, 8
    CONTROL "Scale Y:", 504, "static", SS_RIGHT | WS_CHILD, 19, 72, 36, 8
    CONTROL "OK", 509, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 44, 109, 24, 14
    CONTROL "Cancel", 510, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 145, 109, 24,
 14
    CONTROL "", 505, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 15, 32, 12
    CONTROL "", 506, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 33, 32, 12
    CONTROL "", 507, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 53, 32, 12
    CONTROL "", 508, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 61, 71, 32, 12
END

PARAMS DIALOG LOADONCALL MOVEABLE DISCARDABLE 3, 18, 311, 191
CAPTION "Parameter Input"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_VISIBLE | DS_MODALFRAME | WS_POPUP
BEGIN
    CONTROL "", IDC_TITLE, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 30, 3, 18
2, 12
    CONTROL "", IDC_TYPE, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63, 54, 32
, 12
    CONTROL "", IDC_RATE, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63, 40, 32
, 12
    CONTROL "", IDC_OUTER, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63, 67, 3
```

```
2, 12
    CONTROL "", IDC_DAMAGED, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 64, 95
, 32, 12
    CONTROL "", IDC_THICKNESS, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63,
135, 32, 12
    CONTROL "", IDC_WELLDEPTH, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63,
122, 32, 12
    CONTROL "Type of test:", 111, "static", SS_RIGHT | WS_CHILD, 15, 56, 44, 8
    CONTROL "Rate data file:", 112, "static", SS_LEFT | WS_CHILD, 15, 43, 46, 8
    CONTROL "", IDC_TURBULENCE, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 260
, 68, 32, 12
    CONTROL "", IDC_MAIN, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63, 25, 3
2, 12
    CONTROL "", IDC_WELLHEAD, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 156,
110, 32, 12
    CONTROL "", IDC_RADIUS, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63, 81,
 32, 12
    CONTROL "", IDC_WELL, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 64, 108,
32, 12
    CONTROL "", IDC_POROSITY, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 63, 1
49, 32, 12


    CONTROL "Title:", 143, "static", SS_RIGHT | WS_CHILD, 3, 5, 24, 11
    CONTROL "OK", IDC_OK, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 102, 177, 24,
 14
    CONTROL "Cancel", IDC_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 196,
177, 24, 14
    CONTROL "Damaged zone", 154, "static", SS_RIGHT | WS_CHILD, 9, 94, 51, 8
    CONTROL "Thickness (ft):", 155, "static", SS_LEFT | WS_CHILD, 13, 136, 47, 8
    CONTROL "Well depth (ft):", 156, "static", SS_RIGHT | WS_CHILD, 0, 123, 59, 8
    CONTROL "Turbulence (y/n):", 157, "static", SS_RIGHT | WS_CHILD, 201, 69, 55, 8
    CONTROL "Main file:", 158, "static", SS_RIGHT | WS_CHILD, 11, 28, 49, 8
    CONTROL "Well-head", 159, "static", SS_RIGHT | WS_CHILD, 101, 110, 53, 8
    CONTROL "Boundary", 160, "static", SS_RIGHT | WS_CHILD, 7, 79, 52, 8
    CONTROL "Well radius (ft):", 161, "static", SS_RIGHT | WS_CHILD, 1, 112, 59, 8
    CONTROL "Porosity (fr):", 162, "static", SS_RIGHT | WS_CHILD, 3, 151, 56, 8


    CONTROL "", IDC_HS2, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 156, 39, 3
2, 12

    CONTROL "", IDC_RESTEMP, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 156, 9
6, 32, 12
    CONTROL "", IDC_SHUTIN, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 157, 12
6, 32, 12
    CONTROL "", IDC_SKIN, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 259, 41,
32, 12
    CONTROL "", IDC_SPECGRAV, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 155,
26, 32, 12
    CONTROL "", IDC_CO2, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 156, 53, 3
2, 12
    CONTROL "", IDC_CALIF, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 156, 81,
 32, 12
    CONTROL "", IDC_RESPRES, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 259, 2
8, 32, 12
    CONTROL "", IDC_PERMAB, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 259, 54
, 32, 12
    CONTROL "H2S (fr):", 146, "static", SS_RIGHT | WS_CHILD, 109, 42, 45, 8

    CONTROL "Reservoir", 148, "static", SS_RIGHT | WS_CHILD, 99, 95, 54, 8
    CONTROL "Wellbore", 149, "static", SS_RIGHT | WS_CHILD, 99, 126, 54, 8
    CONTROL "Skin factor:", 150, "static", SS_RIGHT | WS_CHILD, 219, 43, 37, 8
    CONTROL "Gas gravity:", 151, "static", SS_RIGHT | WS_CHILD, 97, 28, 58, 8
    CONTROL "CO2  (fr):", 152, "static", SS_RIGHT | WS_CHILD, 98, 55, 57, 8
    CONTROL "California or", 153, "static", SS_RIGHT | WS_CHILD, 98, 78, 55, 8
    CONTROL "Initial", 166, "static", SS_RIGHT | WS_CHILD, 191, 26, 65, 8
//  CONTROL "Initial", 167, "static", SS_RIGHT | WS_CHILD, 209, 53, 46, 8
    CONTROL "volume:", 169, "static", SS_RIGHT | WS_CHILD, 106, 133, 47, 8
    CONTROL "Open...", IDC_OPENFILE, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 250
, 3, 34, 14
    CONTROL "Outer boundary:", 168, "static", SS_RIGHT | WS_CHILD, 2, 67, 58, 8
    CONTROL "radius (ft):", 170, "static", SS_RIGHT | WS_CHILD, 19, 87, 41, 8
    CONTROL "radius (ft):", 171, "static", SS_RIGHT | WS_CHILD, 12, 102, 49, 8
    CONTROL "condensate gas:", 172, "static", SS_RIGHT | WS_CHILD, 98, 87, 55, 8
    CONTROL "Temperature (F):", 173, "static", SS_RIGHT | WS_CHILD, 98, 102, 55, 8
    CONTROL "Temperature (F):", 174, "static", SS_RIGHT | WS_CHILD, 98, 118, 55, 8

    CONTROL "res. press. (psia):", 176, "static", SS_RIGHT | WS_CHILD, 195, 33, 62, 8
    CONTROL "permeability (mD):", 177, "static", SS_RIGHT | WS_CHILD, 196, 58, 61, 9


    CONTROL "", IDC_WELLBORE, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 260, 1
08, 32, 12
    CONTROL "Field data file:", 178, "static", SS_RIGHT | WS_CHILD, 194, 109, 64, 8
    CONTROL "Automate curve :", 180, "static", SS_RIGHT | WS_CHILD, 189, 122, 71, 9
    CONTROL "", IDC_AUTO, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 261, 122,
32, 12
END


\032
```

```
#include "windows.h"
#include "fran.h"
#include "input.h"
#include "filemgr.h"
#include "scope.h"
#include "fran.dlg"
#include "scope.dlg"

//GenericMenu MENU
//BEGIN
//  POPUP        "Input..."
//  BEGIN
//      MENUITEM "Input Parameters...", IDM_PARAM
//  END
//    POPUP        "&Help"
//    BEGIN
//        MENUITEM "&About...", IDM_ABOUT
//    END

//END

\032
```

```
; module-definition file for generic -- used by LINK.EXE

NAME       Fran   ; application's module name

DESCRIPTION  'Microsoft Windows Application'

EXETYPE      WINDOWS   ; required for all Windows applications

STUB         'WINSTUB.EXE' ; Generates error message if application
                           ; is run without Windows

;CODE can be moved in memory and discarded/reloaded
CODE  PRELOAD MOVEABLE DISCARDABLE

;DATA must be MULTIPLE if program can be invoked more than once
DATA  PRELOAD MOVEABLE MULTIPLE


HEAPSIZE     8048
STACKSIZE    10240   ;


; All functions that will be called by any Windows routine
; MUST be exported.

EXPORTS
MainWndProc   @1   ; name of window processing function
About         @2   ; name of "About" processing function
InputParams   @3   ; input parameters
OpenSave      @4   ; input parameters
GraphProc     @5   ;
ScopeParams   @6   ;

\032
```

```
#include <windows.h>
#include <io.h>
#include <string.h>
#include "filemgr.h"


char                FileName[128];
char                PathName[128];
char                OpenName[128];
char                SaveAsName[128];
char                DefPath[128];
char                DefSpec[13] = "*.*";
char                DefExt[] = ".txt";
char                str[255];

BOOL                DeletingFiles;


/*--------------------------------------------------------------------------.
|                                                                           |
| Function Name - OpenSave()                                                |
|                                                                           |
| Description   -                                                           |
|                                                                           |
| Parameters    -                                                           |
|                                                                           |
|                                                                           |
 ---------------------------------------------------------------------------*/

HANDLE FAR PASCAL OpenSave(hDlg, message, wParam, lParam)
HWND                hDlg;
unsigned            message;
WORD                wParam;
LONG                lParam;
{
/*
    WORD            index;
    PSTR             pTptr;
*/
    HANDLE          hFile=1;     /* Temp value for return */
    char *          ptr;
    int             item;

    switch (message)
       {
       case WM_COMMAND:
            switch (wParam)
               {

               case IDC_DISPLAY:
                  GetDlgItemText(hDlg, IDC_EDIT, OpenName, 128);
                   if (!OpenName[0])
                      {
                      MessageBox(hDlg, "No filename specified.",
                           NULL, MB_OK | MB_ICONHAND);
                      return (TRUE);
                      }
                  return (TRUE);
                  break;

               case IDC_LISTBOX:
                 switch (HIWORD(lParam))
                    {
                    case LBN_SELCHANGE:
                          /* If item is a directory name, append "*.*" */
                       if (DlgDirSelect(hDlg, str, IDC_LISTBOX))
                          strcat(str, DefSpec);

                       SetDlgItemText(hDlg, IDC_EDIT, str);
                       SendDlgItemMessage(hDlg,
                              IDC_EDIT,
                              EM_SETSEL,
                              NULL,
                              MAKELONG(0, 0x7fff));
                       break;

                    case LBN_DBLCLK:
                       goto openfile;
                    }
                 return (TRUE);

               case IDOK:
openfile:
                  GetDlgItemText(hDlg, IDC_EDIT, OpenName, 128);
                  if (strchr(OpenName, '*') || strchr(OpenName, '?'))
                     {
                     SeparateFile(hDlg, (LPSTR) str, (LPSTR) DefSpec,
                              (LPSTR) OpenName);
                     if (str[0])
                        strcpy(DefPath, str);
                     ChangeDefExt(DefExt, DefSpec);
                     UpdateListBox(hDlg);
                     return (TRUE);
                     }

                     if (!OpenName[0]) {
                         MessageBox(hDlg, "No filename specified.",
                             NULL, MB_OK | MB_ICONHAND);
                         return (TRUE);
                         }

                     AddExt(OpenName, DefExt);

                     EndDialog(hDlg, hFile);
                     return (TRUE);

               case IDCANCEL:
                     OpenName[0];
                     EndDialog(hDlg, NULL);
                     return (FALSE);
               }
             break;

        case WM_INITDIALOG:                         /* message: initialize    */
           OpenName[0] = 0;

           UpdateListBox(hDlg);

           SetDlgItemText(hDlg, IDC_EDIT, DefSpec);
           SendDlgItemMessage(hDlg,
                  IDC_EDIT,
                  EM_SETSEL,
                  NULL,
                  MAKELONG(0, 0x7fff));
           SetFocus(GetDlgItem(hDlg, IDC_EDIT));
           ShowWindow(GetDlgItem(hDlg, IDC_DISPLAY), SW_HIDE);
```

```
        return (FALSE); /* Indicates the focus is set to a control */
    }
    return FALSE;
}
/*-------------------------------------------------------------------------*
 |                                                                         |
 | Function Name - UpdateListBox()                                         |
 |                                                                         |
 | Description   -                                                         |
 |                                                                         |
 | Parameters    -                                                         |
 |                                                                         |
 |                                                                         |
 *-------------------------------------------------------------------------*/

void UpdateListBox(hDlg)
HWND            hDlg;
{
    strcpy(str, DefPath);
    strcat(str, DefSpec);
    DlgDirList(hDlg, str, IDC_LISTBOX, IDC_PATH, 0x4010);

    /* To ensure that the listing is made for a subdir. of
     * current drive dir...
     */
    if (!strchr (DefPath, ':'))
        DlgDirList(hDlg, DefSpec, IDC_LISTBOX, IDC_PATH, 0x4010);

    /* Remove the '..' character from path if it exists, since this
     * will make DlgDirList move us up an additional level in the tree
     * when UpdateListBox() is called again.
     */
    if (strstr (DefPath, ".."))
        DefPath[0] = '\0';

    SetDlgItemText(hDlg, IDC_EDIT, DefSpec);
}

/*-------------------------------------------------------------------------*
 |                                                                         |
 | Function Name - ChangeDefExt()                                         |
 |                                                                         |
 | Description   -                                                         |
 |                                                                         |
 | Parameters    -                                                         |
 |                                                                         |
 |                                                                         |
 *-------------------------------------------------------------------------*/

void ChangeDefExt(Ext, Name)
PSTR            Ext, Name;
{
    PSTR            pTptr;

    pTptr = Name;
    while (*pTptr && *pTptr != '.')
        pTptr++;
    if (*pTptr)
        if (!strchr(pTptr, '*') && !strchr(pTptr, '?'))
            strcpy(Ext, pTptr);
}


/*-------------------------------------------------------------------------*
```

```
 |                                                                         |
 | Function Name - SeperateFile()                                          |
 |                                                                         |
 | Description   -                                                         |
 |                                                                         |
 | Parameters    -                                                         |
 |                                                                         |
 |                                                                         |
 *-------------------------------------------------------------------------*/
void SeparateFile(hDlg, lpDestPath, lpDestFileName, lpSrcFileName)
HWND            hDlg;
LPSTR           lpDestPath, lpDestFileName, lpSrcFileName;
{
    LPSTR           lpTmp;
    char            cTmp;

    lpTmp = lpSrcFileName + (long) lstrlen(lpSrcFileName);
    while (*lpTmp != ':' && *lpTmp != '\\' && lpTmp > lpSrcFileName)
        lpTmp = AnsiPrev(lpSrcFileName, lpTmp);
    if (*lpTmp != ':' && *lpTmp != '\\') {
        lstrcpy(lpDestFileName, lpSrcFileName);
        lpDestPath[0] = 0;
        return;
    }
    lstrcpy(lpDestFileName, lpTmp + 1);
    cTmp = *(lpTmp + 1);
    lstrcpy(lpDestPath, lpSrcFileName);
     *(lpTmp + 1) = cTmp;
    lpDestPath[(lpTmp - lpSrcFileName) + 1] = 0;
}


/*-------------------------------------------------------------------------*
 |                                                                         |
 | Function Name - AddExt()                                                |
 |                                                                         |
 | Description   -                                                         |
 |                                                                         |
 | Parameters    -                                                         |
 |                                                                         |
 |                                                                         |
 *-------------------------------------------------------------------------*/
void AddExt(Name, Ext)
PSTR            Name, Ext;
{
    PSTR            pTptr;

    pTptr = Name;
    while (*pTptr && *pTptr != '.')
        pTptr++;
    if (*pTptr != '.')
        strcat(Name, Ext);
}


\032
```

```
/* Control IDs */

#define     IDC_FILENAME  400
#define     IDC_EDIT      401
#define     IDC_FILES     402
#define     IDC_PATH      403
#define     IDC_LISTBOX   404
#define     IDC_DISPLAY   405

HANDLE FAR PASCAL OpenSave(HWND, unsigned, WORD, LONG);
void SeparateFile(HWND, LPSTR, LPSTR, LPSTR);
void UpdateListBox(HWND);
void AddExt(PSTR, PSTR);
void ChangeDefExt(PSTR, PSTR);
\032
```

```
#define IDC_AUTO            182
#define IDC_CALIF           252
#define IDC_CANCEL          202
#define IDC_CO2             251
#define IDC_DAMAGED         207
#define IDC_HOWMANY         220
#define IDC_HS2             245
#define IDC_KI              243
#define IDC_KI1             231
#define IDC_KI2             232
#define IDC_KI3             233
#define IDC_KI4             234
#define IDC_KI5             235
#define IDC_MAIN            211
#define IDC_N2              246
#define IDC_OK              201
#define IDC_ONWHAT          219
#define IDC_OPENFILE        257
#define IDC_OUTER           206
#define IDC_PERM            217
#define IDC_PERMAB          254
#define IDC_PERMFILE        258
#define IDC_PI              241
#define IDC_PI1             221
#define IDC_PI2             222
#define IDC_PI3             223
#define IDC_PI4             224
#define IDC_PI5             225
#define IDC_POROSITY        215
#define IDC_PROD            216
#define IDC_RADIUS          213
#define IDC_RATE            205
#define IDC_RESPRES         253
#define IDC_RESTEMP         247
#define IDC_RQ              244
#define IDC_RQ1             236
#define IDC_RQ2             237
#define IDC_RQ3             238
#define IDC_RQ4             239
#define IDC_RQ5             240
#define IDC_S               242
#define IDC_S1              226
#define IDC_S2              227
#define IDC_S3              228
#define IDC_S4              229
#define IDC_S5              230
#define IDC_SENSITIVITY     218
#define IDC_SHUTIN          248
#define IDC_SKIN            249
#define IDC_SPECGRAV        250
#define IDC_STORAGE         209
#define IDC_THICKNESS       208
#define IDC_TITLE           203
#define IDC_TURBULENCE      210
#define IDC_TYPE            204
#define IDC_WELL            214
#define IDC_WELLBORE        259
#define IDC_WELLDEPTH       256
#define IDC_WELLHEAD        212
```

```
#define SCOPE_INITXTEXT      501
#define SCOPE_INITYTEXT      502
#define SCOPE_SCALEXTEXT     503
#define SCOPE_SCALEYTEXT     504
#define SCOPE_INITXENTRY     505
#define SCOPE_INITYENTRY     506
#define SCOPE_SCALEXENTRY    507
#define SCOPE_SCALEYENTRY    508
#define SCOPE_OK             509
#define SCOPE_CANCEL         510
```

```
SCOPE DIALOG LOADONCALL MOVEABLE DISCARDABLE 28, 41, 240, 128
CAPTION "Data Scope"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Init X :", SCOPE_INITXTEXT, "static", SS_RIGHT | WS_CHILD, 18, 17, 37, 9
    CONTROL "Init Y:", SCOPE_INITYTEXT, "static", SS_RIGHT | WS_CHILD, 21, 36, 34, 8
    CONTROL "Scale X:", SCOPE_SCALEXTEXT, "static", SS_RIGHT | WS_CHILD, 21, 54, 34, 8
    CONTROL "Scale Y:", SCOPE_SCALEYTEXT, "static", SS_RIGHT | WS_CHILD, 19, 72, 36, 8
    CONTROL "OK", SCOPE_OK, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 44, 109, 24
, 14
    CONTROL "Cancel", SCOPE_CANCEL, "button", BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD, 145
, 109, 24, 14
    CONTROL "", SCOPE_INITXENTRY, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 6
1, 15, 32, 12
    CONTROL "", SCOPE_INITYENTRY, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD, 6
1, 33, 32, 12
    CONTROL "", SCOPE_SCALEXENTRY, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD,
61, 53, 32, 12
    CONTROL "", SCOPE_SCALEYENTRY, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP | WS_CHILD,
61, 71, 32, 12
END


\032
```