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:

Judith J. Richardson
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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedication</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>iv</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td><strong>1. Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Previous Work</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Turbulent Flow</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Wellbore Storage</td>
<td>4</td>
</tr>
<tr>
<td><strong>2. Mathematical Model</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 Turbulence Effect</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Formation Damage</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Initial Condition</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Inner Boundary Condition</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Outer Boundary Condition</td>
<td>8</td>
</tr>
<tr>
<td>2.6 Dimensionless Quantities</td>
<td>8</td>
</tr>
<tr>
<td>2.7 Resulting Equations</td>
<td>10</td>
</tr>
<tr>
<td>2.8 Comments on the Finite-Difference Simulation</td>
<td>11</td>
</tr>
<tr>
<td><strong>3. Programing Language and Development Features</strong></td>
<td>13</td>
</tr>
<tr>
<td><strong>4. User Manual for REAL GAS FLOW</strong></td>
<td>14</td>
</tr>
<tr>
<td>4.1 Installation</td>
<td>14</td>
</tr>
<tr>
<td>4.1.1 Configuration</td>
<td>14</td>
</tr>
<tr>
<td>4.1.2 Setting Up</td>
<td>15</td>
</tr>
<tr>
<td>4.1.3 Starting Up</td>
<td>16</td>
</tr>
<tr>
<td>4.1.4 A Quick Guide to REAL GAS FLOW</td>
<td>16</td>
</tr>
<tr>
<td>4.2 Overview</td>
<td>17</td>
</tr>
<tr>
<td>4.2.1 Methodology</td>
<td>17</td>
</tr>
<tr>
<td>4.2.2 Main Menu Bar</td>
<td>19</td>
</tr>
<tr>
<td>4.2.3 File Types</td>
<td>19</td>
</tr>
<tr>
<td>4.2.3.1 Files Created by a User</td>
<td>20</td>
</tr>
<tr>
<td>4.2.3.2 Files Created by REAL GAS FLOW</td>
<td>20</td>
</tr>
<tr>
<td>4.3 Setting Up the Data</td>
<td>21</td>
</tr>
<tr>
<td>4.3.1 Main Data File</td>
<td>21</td>
</tr>
<tr>
<td>4.3.1.1 Opening an Old Main File</td>
<td>22</td>
</tr>
<tr>
<td>4.3.1.2 Creating a New Main File</td>
<td>25</td>
</tr>
<tr>
<td>4.3.1.3 Detailed Description of Each Field</td>
<td>27</td>
</tr>
<tr>
<td>4.3.2 Flow Rate File</td>
<td>31</td>
</tr>
<tr>
<td>4.4 Running the Simulation</td>
<td>32</td>
</tr>
</tbody>
</table>
4.4.1 Running the Right Case 32
4.4.2 During the Simulation 32

4.5 Plotting Results 33
4.5.1 Plot Menu 33
4.5.2 Overview 33
4.5.2.1 Opening an Output File 33
4.5.2.2 Plotting the Data 36
4.5.2.3 Using the Cursor Utility 38

4.6 Special Features 40
4.6.1 Using Field Data 40
4.6.1.1 Setting Up the Field Data 40
4.6.1.2 Plotting Field Data 41
4.6.2 Creating Type Curves 42
4.6.2.1 Background 43
4.6.2.2 Creating One Type Curve 43
4.6.2.3 Building a Library of Type Curves 44

4.7 Printing 47
4.7.1 Printing Directly 47
4.7.2 Printing with CLIPBOARD 47

4.8 Leaving REAL GAS FLOW 47

4.9 Computing Aids 48

5. Using RGF as a Well Test Interpretation Tool 49

6. Conclusions 56

Nomenclature 57
References 60

Appendix A. Computation of Gas Properties 64
Appendix B. Flow Chart of Fortran Code 69
Appendix C. Source Code of Fortran Program for Flow Simulation 72
Appendix D. Source Code of C Program for User Interface 88
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Simulation Time on Various Computers</td>
<td>14</td>
</tr>
<tr>
<td>4.2</td>
<td>Summary of Main Input Data File</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Data File for Library of Type Curves Generated by RGF</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Pressure History for Synthetic Field Data</td>
<td>51</td>
</tr>
<tr>
<td>List of Figures</td>
<td>page</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Fig. 2.1 Radial Flow Model</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.1 Main Structure of REAL GAS FLOW</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.2 Input Parameter Window to Create and Display Main Input Files</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.3 OPEN Window to Access Files on Disks</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.4 Example of File for Sensitivity Analysis, with k=10mD</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.5 Example of File for Sensitivity Analysis, with k=100mD</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.6 Choice of Available Plots for Viewing Results</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.7 OPEN Window to Access Output Files Stored on Disks</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.8 Typical Log-Log Plot with REAL GAS FLOW</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.9 Typical Graph of Both Log-Log and Derivative Plots</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.10 Superposition of Log-Log Plots from Two Different Output Files</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.11 Using the Cursor Utility to Pick Points on a Graph</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.12 Superposition of Field and Synthetic Data</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Fig. 4.13 Example of Combination of Five Type Curves to Build a Library</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Fig. 5.1 Plot Fitting Field Data and Simulated Data by Trial and Error, Log-Log Plot</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Fig. 5.2 Plot Fitting Field Data and Simulated Data by Trial and Error, Semi-Log Plot</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Fig. 5.3 Non-Linear Regression Fit of Synthetic Field Data on Generated Type Curves</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Fig. A.1 Wellbore derivative $\frac{\partial[p/z(p)]}{\partial[p/z(p)]}$ Vs Bottom Hole Pressure for $\gamma_s = 0.9$</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>
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 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 $\beta$) 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 $\beta p v^2$ calculates less pressure drop than would occur, and "velocity coefficient" to describe $\beta$ 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:

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.
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).

\[ u_r = -\delta_r \frac{k}{\mu(p)} \frac{\partial p}{\partial r} \]  

(2.1)

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

\[ \delta_r = \frac{1}{1 + \frac{\beta \rho(p) u_r k}{\mu(p)}} \]  

(2.2)

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

\[ \beta = \frac{e^{23.83}}{k^{1.201}} \]  

(2.3)

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

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

(2.4)
2.2 Formation Damage

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

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

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:

$$p = p_i; t=0, \quad r_w \leq r \leq r_e \quad (2.6)$$

Or, in term of pseudopressure:

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

2.4 Inner Boundary Condition

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

$$q_{sc} = q_{wb} + q_{sf} \quad (2.8)$$
in which:

$$q_{wb} = -\frac{T_{sc}}{p_{sc} T} V_{wb} \left[ C_g(p) \frac{T}{T_f} \frac{1}{k} \frac{\mu(p)}{2} \frac{\partial f(m(p))}{\partial t} \right]$$

(2.9)

and:

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

(2.10)

### 2.5 Outer Boundary Condition

The outer boundary is circular.

The condition for the infinite outer boundary is:

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

(2.11)

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

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

(2.12)

### 2.6 Dimensionless Quantities

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

\[ t_D = \frac{k}{\phi \mu(p_i)c_g(p_i)r_w^2} \tau \]  

(2.13)

Dimensionless pseudo-pressure for constant mass rate:

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

(2.14)

Dimensionless rate:

\[ q_D = \frac{q_{sc} p_{sc} T}{\pi h T_{sc} m(p_i)} \]  

(2.15)

Dimensionless wellbore storage:

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

(2.16)

Dimensionless distance:

\[ r_D = \frac{r}{r_w} \]  

(2.17)

Dimensionless diffusivity:

\[ \alpha_D = \frac{\mu(p)c_g(p)}{\mu(p_i)c_g(p_i)} \]  

(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:

\[
\frac{1}{r_D} \frac{\partial}{\partial r_D} \left[ \delta_{r_D} \frac{\partial m_{ID}(r_D, t_D)}{\partial r_D} \right] = \frac{k}{k_1} \alpha_{ID}[m_{ID}(r_D, t_D)] \frac{\partial m_{ID}(r_D, t_D)}{\partial t_D} \tag{2.20}
\]

Initial condition:

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

At the interface between damaged and undamaged zones:

\[
\delta_1 k_1 \frac{\partial m_{ID}(r_{ID}, t_D)}{\partial r_D} = \delta k \frac{\partial m_D(r_{ID}, t_D)}{\partial r_D} \tag{2.22}
\]
The inner boundary condition is:

\[
\alpha_D \left[ m_D(1, t_D) \right] \frac{\partial [p/z(p)]}{\partial [p/z(p)]} \left[ \frac{\partial [p/z(p)]}{\partial t} \right] _i \cdot C_D \cdot \frac{\partial [m_D(1, t_D)]}{\partial t_D} - \frac{k_1}{k} r_D \delta_1 \frac{\partial m_D(r_D, t_D)}{\partial r_D} \bigg| _{t=1} = 1 \quad (2.23)
\]

The outer boundary conditions are:

For the infinite acting case:

\[
\lim_{t_0 \to t_D} m_D(r, t_D) = m_D(r_D, 0) \quad (2.24)
\]

For the no-flow condition:

\[
\frac{\partial m_D(r_D, t_D)}{\partial r_D} = 0 \quad (2.25)
\]

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:

\[ v = \frac{\ln r_D}{\ln r_e} \]  

(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.]

<table>
<thead>
<tr>
<th></th>
<th>286-6 Mhz</th>
<th>386-16Mhz</th>
<th>486-25Mhz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darcy Flow</td>
<td>8mins</td>
<td>4mins</td>
<td>2mins</td>
</tr>
<tr>
<td>Non-Darcy Flow</td>
<td>12mins</td>
<td>7mins</td>
<td>3mins</td>
</tr>
</tbody>
</table>

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.

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.

![Image of Parameter Input Window]

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 referred 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.
<table>
<thead>
<tr>
<th>Parameter Input</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Sensitivity analysis</td>
</tr>
<tr>
<td>Main file:</td>
<td>xyz10</td>
</tr>
<tr>
<td>Rate data file:</td>
<td>xyz</td>
</tr>
<tr>
<td>Type of test:</td>
<td>drawdo</td>
</tr>
<tr>
<td>Outer boundary:</td>
<td>infinite</td>
</tr>
<tr>
<td>Boundary radius [ft]:</td>
<td>0</td>
</tr>
<tr>
<td>Damaged zone radius [ft]:</td>
<td>3</td>
</tr>
<tr>
<td>Well radius [ft]:</td>
<td>0.33</td>
</tr>
<tr>
<td>Well depth [ft]:</td>
<td>10000</td>
</tr>
<tr>
<td>Thickness [ft]:</td>
<td>27</td>
</tr>
<tr>
<td>Porosity [fr]:</td>
<td>0.1</td>
</tr>
<tr>
<td>Prod / obs distance [ft]:</td>
<td></td>
</tr>
<tr>
<td>Gas gravity:</td>
<td>0.9</td>
</tr>
<tr>
<td>H2S [fr]:</td>
<td>0</td>
</tr>
<tr>
<td>CO2 [fr]:</td>
<td>0</td>
</tr>
<tr>
<td>N2 [fr]:</td>
<td>0</td>
</tr>
<tr>
<td>California or condensate gas:</td>
<td>califoni</td>
</tr>
<tr>
<td>Reservoir Temperature [F]:</td>
<td>255</td>
</tr>
<tr>
<td>Wellhead Temperature [F]:</td>
<td>75</td>
</tr>
<tr>
<td>Wellbore volume:</td>
<td>5000</td>
</tr>
<tr>
<td>Initial res. press. (psia):</td>
<td>5000</td>
</tr>
<tr>
<td>Skin factor:</td>
<td>3</td>
</tr>
<tr>
<td>Initial permeability (mD):</td>
<td>10</td>
</tr>
<tr>
<td>Turbulence [y/n]:</td>
<td>yes</td>
</tr>
<tr>
<td>Perm = f[P] [y/n]:</td>
<td>.</td>
</tr>
<tr>
<td>Own perm file:</td>
<td>.</td>
</tr>
<tr>
<td>Field data file:</td>
<td>.</td>
</tr>
<tr>
<td>Automate curve [y/n]:</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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:

\[ \lim_{r \to \infty} m(p) = m(p_i) ; \quad t > 0 \]  \hspace{1cm} (4.1)

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

\[ \frac{\partial m(p)}{\partial r} \bigg|_{r=a} = 0 \quad ; \quad t > 0 \]  \hspace{1cm} (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:

\[
C_D = \frac{V_{wb}}{2\pi r_h h \phi} \left[ \frac{T}{T} \frac{\partial [\tilde{p}/z(p)]}{\partial p} \right]
\]

\[
(4.3)
\]
(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:

$$s = \left[ \frac{k}{k_1} - 1 \right] \ln \frac{r_1}{r_w}$$  \hspace{1cm} \text{(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 $\phi$ 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.

![Plot Menu Screen](image)

**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 Δt log(Δmp/Δt) 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:

\[ P_{Pn} = P_i + \frac{\mu_i z_i}{P_i} \int_P^{P_i} \frac{P dP}{\mu z} \] \hspace{1cm} (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:**

(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:**

(8) **Line source solution:** Not accessible

(9) **Rate history:**

Cartesian plot of rate in Mcf/D versus 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.

![Typical Log-Log Plot with REAL GAS FLOW](image)

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 psia²/cp and time=0.23 hours.

![Graph of Delta mp vs. Delta time (hours)](image)

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:

<table>
<thead>
<tr>
<th>Time (hr)</th>
<th>Pressure (psia)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>4507</td>
</tr>
<tr>
<td>0.05</td>
<td>4505</td>
</tr>
<tr>
<td>0.1</td>
<td>4490</td>
</tr>
<tr>
<td>0.2</td>
<td>4485</td>
</tr>
<tr>
<td>0.8</td>
<td>4480</td>
</tr>
<tr>
<td>1.0</td>
<td>4473</td>
</tr>
<tr>
<td>10</td>
<td>4460</td>
</tr>
<tr>
<td>100</td>
<td>4450</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
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:

\[
\frac{\partial P}{\partial t_{i,j}} = \left[ \frac{\partial P}{\partial \ln t_{i,j}} \right]
\]

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

(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 \( p_D \) versus \( t_D \) plot
"2" for a storage \( p_D \) 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 \( C_D.e^{2s} \) 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](image)

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

Data points for the curves: \( t_D, p_{D1}, p_{D2}, \text{etc.} \)

All curves must have the same number of points and share the same time values.

Parameters related to the curve labels.

Data point for the derivative curves.

They must have the same number of points as the \( p_D \) curves.

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 reusing 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.pr s *.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.
<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Pressure (psi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8000</td>
</tr>
<tr>
<td>0.001</td>
<td>7997</td>
</tr>
<tr>
<td>0.0027</td>
<td>7991</td>
</tr>
<tr>
<td>0.0044</td>
<td>7986</td>
</tr>
<tr>
<td>0.006</td>
<td>7981</td>
</tr>
<tr>
<td>0.0077</td>
<td>7976</td>
</tr>
<tr>
<td>0.012</td>
<td>7962</td>
</tr>
<tr>
<td>0.025</td>
<td>7923</td>
</tr>
<tr>
<td>0.04</td>
<td>7886</td>
</tr>
<tr>
<td>0.052</td>
<td>7851</td>
</tr>
<tr>
<td>0.066</td>
<td>7816</td>
</tr>
<tr>
<td>0.08</td>
<td>7783</td>
</tr>
<tr>
<td>0.1</td>
<td>7716</td>
</tr>
<tr>
<td>0.24</td>
<td>7460</td>
</tr>
<tr>
<td>0.37</td>
<td>7264</td>
</tr>
<tr>
<td>0.5</td>
<td>7110</td>
</tr>
<tr>
<td>0.64</td>
<td>6985</td>
</tr>
<tr>
<td>0.77</td>
<td>6881</td>
</tr>
<tr>
<td>1.91</td>
<td>6440</td>
</tr>
<tr>
<td>3.58</td>
<td>6243</td>
</tr>
<tr>
<td>5.24</td>
<td>6174</td>
</tr>
<tr>
<td>6.91</td>
<td>6139</td>
</tr>
<tr>
<td>8.24</td>
<td>6120</td>
</tr>
<tr>
<td>10.8</td>
<td>6095</td>
</tr>
<tr>
<td>24</td>
<td>6030</td>
</tr>
<tr>
<td>37.5</td>
<td>5996</td>
</tr>
<tr>
<td>50.8</td>
<td>5973</td>
</tr>
<tr>
<td>64</td>
<td>5956</td>
</tr>
<tr>
<td>77.5</td>
<td>5942</td>
</tr>
<tr>
<td>224</td>
<td>5865</td>
</tr>
<tr>
<td>458</td>
<td>5813</td>
</tr>
<tr>
<td>691</td>
<td>5784</td>
</tr>
</tbody>
</table>

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:

![Plot-Fitting Field Data and Simulated Data by Trial and Error, Semi-Log Plot](image)

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.
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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cr</td>
<td>dimensionless wellbore storage</td>
</tr>
<tr>
<td>cg</td>
<td>gas compressibility, psi$^{-1}$</td>
</tr>
<tr>
<td>h</td>
<td>formation thickness, ft</td>
</tr>
<tr>
<td>k</td>
<td>formation permeability, md</td>
</tr>
<tr>
<td>k1</td>
<td>damaged annular region permeability, md</td>
</tr>
<tr>
<td>M</td>
<td>molecular weight, lb$_m$/mole</td>
</tr>
<tr>
<td>m(p)</td>
<td>real gas potential, psi$^2$/cp</td>
</tr>
<tr>
<td>p</td>
<td>pressure, psia</td>
</tr>
<tr>
<td>p̅</td>
<td>average wellbore flowing pressure, psia</td>
</tr>
<tr>
<td>Pi</td>
<td>initial formation pressure, psia</td>
</tr>
<tr>
<td>Psc</td>
<td>standard pressure, psia</td>
</tr>
<tr>
<td>Ppn</td>
<td>normalized pseudopressure, psia</td>
</tr>
<tr>
<td>Ppc</td>
<td>pseudocritical pressure, psia</td>
</tr>
<tr>
<td>P*c</td>
<td>corrected critical pressure, psia</td>
</tr>
<tr>
<td>Pr</td>
<td>reduced pressure, dimensionless</td>
</tr>
<tr>
<td>qsc</td>
<td>gas production rate, Mcf/D</td>
</tr>
<tr>
<td>qsf</td>
<td>sandface gas flow rate, Mcf/D</td>
</tr>
<tr>
<td>qwb</td>
<td>wellbore rate, Mcf/D</td>
</tr>
<tr>
<td>r</td>
<td>radial distance from wellbore center, ft</td>
</tr>
<tr>
<td>r1</td>
<td>damaged annular region radius, ft</td>
</tr>
<tr>
<td>re</td>
<td>external radius, ft</td>
</tr>
<tr>
<td>rw</td>
<td>wellbore radius, ft</td>
</tr>
<tr>
<td>R</td>
<td>universal gas constant</td>
</tr>
<tr>
<td>s</td>
<td>real skin effect</td>
</tr>
</tbody>
</table>
\[ t = \text{time, hrs} \]
\[ T = \text{formation temperature, } ^\circ \text{R} \]
\[ \overline{T} = \text{average wellbore flowing temperature, } ^\circ \text{R} \]
\[ T_{\text{sc}} = \text{standard temperature, } ^\circ \text{R} \]
\[ T_{\text{pc}} = \text{pseudocritical temperature, } ^\circ \text{R} \]
\[ T_r = \text{reduced temperature, dimensionless} \]
\[ T^*_C = \text{corrected critical temperature, } ^\circ \text{R} \]
\[ u = \text{macroscopic gas velocity} \]
\[ V_{\text{wb}} = \text{wellbore volume, cu ft} \]
\[ z = \text{real gas law deviation factor} \]
\[ \alpha = \text{diffusivity parameter} \]
\[ \beta = \text{turbulence parameter, } \text{md}^{-1} \]
\[ \gamma_s = \text{specific gas gravity (to air)} \]
\[ \delta_i = \text{Darcy's law correction factor for damaged region} \]
\[ \delta_r = \text{radial Darcy's law correction factor} \]
\[ v = \text{dimensionless logarithm ratio} \]
\[ \mu(p) = \text{pressure dependent gas viscosity, } \text{cp} \]
\[ \phi = \text{porosity, fraction} \]
\[ \rho(p) = \text{pressure dependent gas density, } \text{lb}_{\text{m/cu ft}} \]

Subscripts:

\[ 1 = \text{damaged around the well} \]
\[ D = \text{dimensionless} \]
\[ e = \text{external} \]
\( g \) = gas

\( i \) = initial

\( r \) = distance \( r \) from the wellbore center

\( sc \) = standard condition

\( sf \) = sandface

\( w, \text{wb} \) = wellbore
References


Appendix A.

Computation of Gas Properties

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

For California gases:

$$p_{pc} = 677 + 15\gamma_g - 37.5\gamma_g^2$$  \hspace{1cm} (A.1)

$$T_{pc} = 168 + 325\gamma_g - 12.5\gamma_g^2$$  \hspace{1cm} (A.2)

For Condensate gases:

$$p_{pc} = 706 - 51.7\gamma_g - 11.1\gamma_g^2$$  \hspace{1cm} (A.3)

$$T_{pc} = 187 + 330\gamma_g - 71.5\gamma_g^2$$  \hspace{1cm} (A.4)

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

$$e = 120(\gamma_{CO_2} + \gamma_{H_2S})^{0.9} - 120(\gamma_{CO_2} + \gamma_{H_2S})^{1.6} + 15(\gamma_{H_2S}^{0.5} - \gamma_{H_2S}^{4})$$  \hspace{1cm} (A.5)

$$p^* = \frac{p_c(T_c - e)}{T_c + \gamma_{H_2S}(1 - \gamma_{H_2S})e}$$  \hspace{1cm} (A.6)

$$T^* = T_c - e$$  \hspace{1cm} (A.7)

The reduced pressure and temperature are:

$$p_r = \frac{p}{p_{pc}}$$  \hspace{1cm} (A.8)

$$T_r = \frac{T_R}{T_{pc}}$$  \hspace{1cm} (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:

\[ \rho_r^{k+1} = \rho_r^k - \frac{f(\rho_r^k)}{f'(\rho_r^k)} \]  \hspace{1cm} (A.10)

where:

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

\[ f'(\rho_r) = 6a\rho_r^5 + 3b\rho_r^2 + 2c\rho_r + d + e\rho_r^2(3 + \rho_r^2[3 - 2\rho_r^2])\exp[-\rho_r^2] \]  \hspace{1cm} (A.12)

where:

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

and:

\[ z = \frac{0.27\rho_r}{\rho_rT_r} \]  \hspace{1cm} (A.13)
Gas compressibility is computed from:

\[
\begin{align*}
 c_g &= \frac{1}{p_c p_r \left[ 1 + \frac{\rho_r}{z} \frac{\partial z}{\partial \rho_r} \right]} \\
\end{align*}
\]  

(A.14)

where:

\[
\frac{\partial z}{\partial \rho_r} = \frac{1}{\rho_T r} \left[ 5a \rho_r^5 + 2b \rho_r^2 + c \rho_r + 2e \rho_r^2 (1 + f \rho_r^2 - f^2 \rho_r^4) \exp[-f \rho_r^2] \right]
\]

(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:

\[
\mu_g = K \cdot 10^{-4} \cdot \exp(X \rho^y)
\]

(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 \( \frac{\partial[p/z(p)]}{\partial[p/z(p)]} \) 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:

\[
0.01875 \gamma_g H = \int_{p}^{p'} \frac{T_z(p')dp'}{p'}
\]

(A.17)
where $H$ is the length of the section along which the integration is performed, and $P_t$ 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 $\frac{\partial [P/z(p)]}{\partial [P/z(p)]}$ as a function of the bottom hole pressure for a wellhead temperature of 75° F, a temperature gradient of 20° 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 \( \frac{\partial [P/z(P)]}{\partial [P/z(P)]} \) Vs Bottom Hole Pressure for \( \gamma_g = 0.9 \)
Appendix B. Flow Chart of Fortran Code

Start

Get name of main file; Read "input.fil"

Create and Open all files

Creation of gas properties tables

Compute initial values in reservoir at each point

\[ t_D = t_D + dt_D \]

Iter = Iter + 1

Calculate Gama coefficient at each point

Subroutines:
- gaspro,
- zfacgt,
- viscosity,
- wellpro.

Calculate Turbulence coeff. for each bloc

Set up matrix coefficients

Solve matrix equation by Thomas Algorithm

Subroutines:
- subroutine Gamaco
- subroutine Deltc
- subroutine Matcoef
- subroutine Thomas
Convergence reached?

Yes

Storage of Data for plots and type curves

No

iter > item?

Yes

{ New iteration }

No

Is outer boundary felt?

Yes

Squaring Re/Rw
Rearrangement of matrix

No

New log cycle of dimensionless time?

Yes

Set up new time step

Subroutine Setdt

No

{ New time step }

{ New time step }
Subroutine Ticartes, Tilog.

Subroutine Trancar, Tranlog.

Prepare data for graphics

Set up scales of graphics

Set up pressure and time values into "pixel" form

Field data?

Set up field data for graphics

Type curve?

Create type curve file

STOP
Appendix C.

Source Code of Fortran Program for Flow Simulation
Franc.for

50 RID = RI / NW
If ( (CODEREA(1:1).NE.'Y').AND.(CODEREA(1:1).NE.'Y') ) Then
   COBRET = 0.
   Go To 60
End If

COBRET = 1.

*********** initial values ***********

60 MINTER = 36
RDIM = 50
ITEM = 50

ERROR = 0.00001
DTUL = 0.00001
SIGMA = 1.
TETA = 1.

MFP = 0
MNPATRO = 0
STO = STDI
RED = REDI
M = INT( (MINTER+1) * LOG(10) / LOG(10^0.1) + 1 )

******* EVALUATION OF BASIC PARAMETERS:*******

MP = sookwp(presl)
VISCGR = sookcp(presl)
VISCAL = sookvi(presl)
X = sooks(presl)
Dervi = sookdv(presl)
Facini = (vistl1+stini)/presl

DU = 1. / MINTER
RMBR = MINTER + 1
Do 80 T = 1, RMBR
   UI(T) = DU * (I-1)
80 Continue

*********** compute dimensionless values ***********

QO = 14.7 * T/E / (thick * perm / 1.52 * 1.9875 - 5)
CD = Dervi / vistl1 / temp / (temp / temp + 3.1415 * thick * poros / rw) * rw
print *, CD
TDMIN = STO
FSTI = 1.
RAI = STO / (DU**2)
REDLG = LOG(RED)
REDLG22 = REDLG ** 2
STOUD = (0.0000254 * PERMI) / (POROS * NW**2 * VISCGR)

*********** more initial values ***********

ZMITMS = 100.
ZMITP = 1000000.
ZMITUP = 1.1
ZMITRX = 100.

IMXPRE = presl
IMXMPW = 1.04
IMXMD = 100.
IMXMDMP = 1.04
IMXRATE = 1000000.

UNIT1 = 10000.

ZMAP = 0.01
ZMAD = 1.
ZMAPTO = 9999.
ZMAPTRD = 1000.
ZMAPPRE = presl
ZMAPMPW = 1.04
ZMAM = 10.
ZMAMMP = 1.04
ZMAMRATE = 1.
ZMAMLE = 1.

IF (CID,NE.1) Then
   Go To 90
End If

FS = 1.
Go To 100

90 RIDLG = LOG(RID)
FS = 1. / (5*RIDLG+1.)
PERM = PERMI * FS

PARAMETERS FOR CALCULATING THE TURBULENCE COEFFICIENT:**********

100 IF (CORDER.EQ.1.) Then
   Go To 110
End If

BETAI = 0.
PERHW = 0.
Go To 120

110 CIDNY = 158.02219
C2 = 2.89838E-14
CMXHD = 28.86 * SPI / (10.7328E+TDMW)
CMXCP = CMXHD / NTI * CD / (2.5*MPW*EDLG0)-
ARG = -1.201 * LOG(PERMI) + 23.83
BETAI = EXP(ARG)

*********** more initial values ***********

120 MXFEP = 1
BRED = 1
LSTNM = 0
MHTL = 9
LGSTHM = 1
GPRD = 0.
TO = STO

STARTING VALUES OF DMP,UNIT1,DELT,DELF,GMAP ************
```fortran
500 DO 150 I = 1, NMNR
   DMP(1) = 0.
   DMTER(I) = 0.
   IF (QO.GT.0.03) DMTER(I) = 0.35
   IF (QO.GT.0.06) DMTER(I) = 0.65
   IF (I.EQ.WMNR) THEN
      Go To 130
   END IF
   DELT(I) = 1.
   DELT(I) = 1.
130 IF (I.GT.0) THEN
   Go To 140
END IF
   CALL GAMACO (NPETB, I, OD, MPI, VISCGL, DMP(1), U(I), RED, REDLG2, FS, RAI,
   & NCDAT, DERIV, PRESI, GAMAP(I))
   Go To 150
140 CALL GAMACO (NPETB, I, OD, MPI, VISCGL, DMP(1), U(I), RED, REDLG2, FS1,
   & NCDAT, DERIV, PRESI, GAMAP(I))
150 GAMAI(I) = GAMAP(I)
160 Continue
   ALFAI = GAMAI(I)
   DMITI = DMTER(I)
   CALL DETERMINE THE INITIAL GUESS VECTOR
Currectionary Procedure
C
170 ITER = 1
   IF (WSTEP.EQ.1) THEN
      Go To 190
   END IF
   CALL DETERMINE OF THE INITIAL GUESS VECTOR
C
180 Continue
   DMITI = DMTER(I) + SIGMA * (DM(1)-DMP(1))
   DMP(1) = DMP(1)
   GAMAP(I) = GAMAI(I)
   IF (I.EQ.WMNR) THEN
      Go To 180
   END IF
   DELT(I) = DELT(I)
190 Continue
   DMITI = DMTER(I)
   ALFAI = GAMAP(I)
   CALL CALCULATION OF GAMAI TERMS
C
If (I.GT.0) Then
   Go To 200
End If
   CALL GAMACO (NPETB, I, OD, MPI, VISCGL, DMITI(I), U(I), RED, REDLG2, FS,
   & RAI, NCDAT, DERIV, PRESI, GAMAI(I))
   Go To 210
200 CALL GAMACO (NPETB, I, OD, MPI, VISCGL, DMITI(I), U(I), RED, REDLG2, FS1,
   & RAI, NCDAT, DERIV, PRESI, GAMAI(I))
210 Continue
   ALFAI = GAMAI(I)
   CALL CALCULATE THE TURBULENCE COEFFICIENT (DELT):=
C
220 IF (BETA.EQ.0.) Then
   Go To 220
End If
   CALL DELTE (NPETB, WMNR, BETA, RED, OD, M, C2, C1NF, CHNO, CDERT, PERNI,
   & PERSI, PRESI, MPI)
C
230 CALL SAVING THE MATRIX COEFFICIENTS
C
240 Call MATCOF (WMNR, M, TETA, CD, REDLG, OD, DTD, FS, ALFAI, ALFAI, WMNR,
   & WSTEP, A, B, TD)
C
250 CALL SOLVE THE EQUATION BY THOMAS ALGORITHM
C
260 CALL THOMAS (A, B, WINTER, WMNR)
270 Do 230 I = 1, WINTER
   IF (WSTEP.EQ.1) THEN
      IPI = I + 1
      DEL(IPI) = B(I)
   END IF
   DMITI = DMITI(I) + DEL(I)
   DMITI = DMITI(I) + DEL(I)
280 Do 180 I = 1, NMNR
   DMTER(I) = DMTER(I) + SIGMA * (DM(I)-DM(I))
   DMP(1) = DMP(1)
   GAMAP(I) = GAMAI(I)
   IF (I.EQ.WMNR) THEN
      Go To 180
   END IF
   DELT(I) = DELT(I)
   180 Continue
   DMITI = DMTER(I)
   ALFAI = GAMAP(I)
C
C
190 Do 190 I = 1, NMNR
C
```

franx.for

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 380

Re-Calculation of QMTER *******************

250 Do 260 I = 2, NMMR
  DMTER(I) = DMTER(I) + DEL(I)
260 Continue

DMMPL = DMTER(1)
Go To 190

270 Do 280 I = 1, NMMR
  DM(I) = DMTER(I)
280 Continue

***************************************************************************************
*****Transformation to plotting values************************************************
***************************************************************************************

If (MPLP.EQ.2) then
  XMPREM=DMPL
  TIPR=0.
end if

TIME = TD / TOTD
XMPW = MPLP*(1.-QD+DM(I))
XPREM = DMPL*(1.-QD)+PLS
XN = DMPL*XPREM
XW = DMPL*XPREW
XWP = DMPL*(1.-QD)

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

go to 290

End If

NQMTO = NQMTO + 1
If (NQMTO.EQ.5) Then
  NQMTO = 1
End If

If (NQMTO.EQ.1) Then
  Go To 290
End if

Write (13,5700) TD, DM(1), DELV
NPP = NPP + 1

DIMR2=10000.
XPRE = XPREM
TDX = (TDX+TIME) / TIME
TDX = TD / DIMR2
RATE = QCF(1)

****************************************************************************
Preparation of data for input curve
*****************************************************************************

End If

If (YCUCK.EQ.1) then
  If (LCTLG.EQ.1) then
    TDYCYC(DMPL)=DELYC(FACNL*PLS*(TD))
    TDYCYC(DMPL)=TD
    TDYCYC(DMPL)=DMPL
  Endif
  If (LCTLG.EQ.2) then
    TDYCYC(DMPL)=TD
    TDYCYC(DMPL)=TDX
  Endif
  If (LCTLG.EQ.3) then
    TDYCYC(DMPL)=DELV
    TDYCYC(DMPL)=DMPL
  Endif
  If (LCTLG.EQ.4) then
    TDYCYC(DMPL)=DELV
    TDYCYC(DMPL)=DMPL
  Endif
End If

End if

End if

Computation of MIN and MAX values ********************************

INITIME = AMIN(INITIME, TIME)
ZMATIME = AMAX(ZMATIME, TIME)
INITO = AMIN(INITO, TD)
ZMATO = AMAX(ZMATO, TD)

IMAXPRE = AMAX(IMAXPRE, XPRE)
IMAXDMP = AMAX(IMAXDMP, XDMP)

MINL = AMIN(MINL, DELV, DMP)

MINLM = AMIN(MINL, DELV, DMP)

MAXL = AMAX(MAXL, DELV, DMP)

MAXLM = AMAX(MAXLM, DELV, DMP)

IF (TPU.TE.1.E-07) Then
  Go To 340
End If

NSTEP = NSTEP + 1
If (TD.GT.TOT) Then
  Go To 360
End If

End if

End if

******************************************************************************
Re-arrangement of 'DD' and 'GAMA' to fit an increase in REV
******************************************************************************

If (DMHM.DLT.1.E-7) Then
  Go To 340
End If

NRED = NRED + 1
If (NRED.NE.4) Then
  Go To 340
5

End If
RED = RED + RED
REDIG = LOG(RED)
REDIG2 = REDIG ** 2

If (s.eq.0.0 .OR. R1D.EQ.1.) Then
  Go To 300
End If
M = nint(nint(red/log(red)) + 1)

300 If (posd.GE.1.) Then
  mobs = nint(nint(log(posd)/log(red)) + 1)
  If (mobs.nint(nint+1)) Then
    mobs = nint + 1
  Endif
Endif

NW = (NMNR-1) / 2 - 1
Do 310 I = 2, NW
DM(I) = DM(1+I)
310 Continue

NW = NW + 1
Do 320 I = NW, NMNR
DM(I) = 0.
GAMN(I) = 0.
320 Continue

If (CDERT.EQ.0.) Then
  Go To 340
End If
Do 330 I = 1, NMNR
DMTRN(I) = DM(I)
330 Continue

CDERT = -CHRO * MPI * GD / (2.*NM+REDIG*OU)
Call DEUT(DPTAB, NMNR, BETA, RED, GD, C2, CHV, CHRO, CDERT, PERN, & PERN1, FRESI, MP1)

C INCREASE IN TIME STEP:******************************
C
340 LGIENM = LGIENM + 1
If (LGTMN, LE, NGESTLGE) Then
  Go To 350
End If
LGTMN = LGTMN + 1
TOMIN = 10. * TOMIN
Call SETP(LGTMN, TOMIN, D0, DTD, LGTMN, NGESTL, RAI)

Print *, 'Calculating'
350 TD = TD + DTD
Go To 170

C *********************************************************
C
C *********************************************************

360 If (tyc1t.EQ.1.) Then
  Kauto(1) = int(log10(redm)+1)
  Kauto(2) = int(log10(redm))
Endif

C *********************************************************
C
C *********************************************************

362 Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
Call ticles(tmlxt, tmlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt, zmtlxt)
INTEGER NMIN, NM1, I, J
REAL T, S, X(I), Y(I), Z(I)

DATA X(1), Y(1), Z(1) / 1.0, 2.0, 3.0 /

FORWARD ELIMINATION

DO 20 I=2,N
   T=0.1/(X(I)-X(I-1))
   B(I)=B(I-1)*T
   C(I)=C(I-1)*T
20 CONTINUE

BACK SUBSTITUTION

DO 30 I=N,1,-1
   B(I)=B(I+1)+T*B(I+1)
   C(I)=C(I+1)+T*C(I+1)
30 CONTINUE

CALL MATCH(NMIN, NM1, X, Y, Z)

SUBROUTINE MATCH(NMIN, NM1, X, Y, Z)

IMPLICIT REAL*(8)
DIMENSION X(NMIN, NM1), Y(NMIN, NM1), Z(NMIN, NM1)

COMMON/AM/ C(NMIN, NM1)
COMMON/CM/ T(NMIN, NM1)

C*** PURPOSE: THIS SUBROUTINE CALCULATES THE MATRIX COEFFICIENTS.

C*** INNER BOUNDARY CONDITION

C(1,1)=0
C(I,J)=T(I,J)*C(I,J-1)+T(I,J-1)*C(I,J-2)
C(I,J)=C(I,J)*T(I,J)

C*** END CONDITIONS. THIRD DERIVATIVES AT X(1) AND X(NM1)

C*** OBTAINED FROM DIVIDED DIFFERENCES

C(I,J)=C(I,J)/T(I,J)

RETURN
END
A(I,3)=TETA*DELT(2) - (TETA*DELT(1)/A(1))*TETA*DELT(2)/12. *DU)

IF (H.GE.0.001) THEN
GO TO 10
ENDIF

TERM1(1) = (1.-TETA) / (1.+DU*ALPHA1) * (3.*DELT(1) + *DELT(2) + *DELT(3) + *DELT(4) + *DELT(5) + 1.)/GAMA(11) + *DELT(1)/GAMA(11)

GO TO 20

TERM1(1)=0.

B(I)=A(I,1)*OMIETER(I)+A(I,2)*OMIETER(I)+A(I,3)*OMIETER(I)+A(I,4)*OMIETER(I)+A(I,5)*OMIETER(I)

END

DIMENSION A(01.),1.1.,1.2.,1.3. - THE COEFFICIENTS OF THE MATRIX.
C
B(I) - THE RIGHT HAND SIDE OF THE EQUATIONS.
C
A(I,1),A(I,2),A(I,3) - THE COEFFICIENTS OF THE MATRIX.
C
A(I,1)=B(I)/A(I,1)
A(I,2)=A(I,2)/A(I,1)
A(I,3)=A(I,3)/A(I,1)

DO 40 N=2,NN

A(N-2)=A(N-1)*A(N-2)
A(N-1)=A(N-2)/A(N-1)
N=N-1

IF (N.GT.10) THEN
GO TO 37
ENDIF

B(N)=A(N-1)*B(N-1)/N

END

General Triangular Non-Symmetric Matrix Solver
C
B(I) - THE RIGHT HAND SIDE OF THE EQUATIONS.
C
A(I,1),A(I,2),A(I,3) - THE COEFFICIENTS OF THE MATRIX.
C
A(I,1)=B(I)/A(I,1)
A(I,2)=A(I,2)/A(I,1)
A(I,3)=A(I,3)/A(I,1)

DO 40 N=2,NN

A(N-2)=A(N-1)*A(N-2)
A(N-1)=A(N-2)/A(N-1)
N=N-1

IF (N.GT.10) THEN
GO TO 37
ENDIF

B(N)=A(N-1)*B(N-1)/N

END}

This subroutine calculates time step sequence
C
IF (LCYNN.GE.3) THEN
GO TO 20
ENDIF

MNSTLG=18
LCYNN=1
DTE0=0.1*TDMIN

GO TO 20

MNSTLG=90
LCYNN=1
DTE0=0.1*TDMIN
50 RAI-DTD/(QU**2)
RETURN
END

C*************************************************************************
C SUBROUTINE CALACO1(PMTAB, IP, MP, VISCCT, DM, U, RED, RED1G2, FS,
C *RAI, MODNAME, DERIV, PRES, GAM)
C*************************************************************************
IMPLICIT REAL*4(A-H, O-I)
COMMON /TABLE/ PTAB(1000), MPTAB(1000), VSTAB(1000)
COMMON /TABLE/ STAB(1000), VISTAB(1000), DERIV(1000)
REAL*4 MF, MP, MPTAB

C PURPOSE: THIS SUBROUTINE CALCULATES GAMMA COEFFICIENTS

MF-MP**2*(1.-QD/DM)
PRESS = dichot(MP)
VISCCT = soakct(PRESS)
PERM=1.
ALFA=VISCCT/VSTAB/PERM
RED1G2=RED**2*(Z.E-1)
GAMA=FS*RAI/(RED1G2*ALFA)

IF(IGT.1) THEN
GO TO 30
ENDIF

C*************************************************************************
C WHEN I=1: ******************************************************
C DERIV = soakd(press)
C GAMMA=ALFA*DERIV/DERIV

30 RETURN
END

C*************************************************************************
C real*4 function soakct(p)
C real*4 p, mptab
C
C*************************************************************************
C real*4 function dichot(m)
C real*4 m, mptab

C*************************************************************************
C this subroutine is a dichotomy scheme to get P. from m(P).
C
10 IF(mptab(millieu),EQ.mp) THEN
11 dichot = ptab(millieu)
12 goto 30
13 ENDIF
14 IF(mgrand-npetit,EQ.1) THEN
15 goto 70
16 ENDIF
17 IF(mptab(millieu),LT.mp) THEN
18 npetit = millieu
19 millieu = INT((npetit+ngrand)/2)
20 gato 10
21 ELSE
22 ngrand = millieu
23 millieu = INT((ngrand+npetit)/2)
24 gato 10
25 ENDIF
26 ARATIO = (mp-mptab(npetit))/(mptab(ngrand)-mptab(npetit))
27 dichot = 10.*(ARATIO + npetit)
30 RETURN
END

C*************************************************************************
C real*4 function soakct(p)
C real*4 p
C
C*************************************************************************
C real*4 function dichot(m)
C real*4 m, mptab

C*************************************************************************
C this subroutine is a table-lookup
C
a = p/10.
1 = INT(a)
ripart = abstra-INT(a)
soakmp = mptab[1] + ripart*(mptab[1+1]-mptab[1])
RETURN
END

C*************************************************************************
C real*4 function dichot(m)
C real*4 m, mptab

C*************************************************************************
C this subroutine is a table-lookup
C
a = p/10.
1 = INT(a)
Franc.for

TSTAR = TCRIT - EPS
PSTAR = PCRIT*TSTAR/(TCRIT+KH2S*EPS*(1.-KH2S))
call wellpro(TSTAR,PSTAR,WD,TMW,TEMPI,TEMPI)

TEMW = TEMW + 460.
TREDU = TEMW/TSTAR
c
ptab(1) = 10.
pedu=ptab(1)/pstar
call xfracg(ptab(1),pedu,tredu,stab(1),cqt)
call viscosity(spg,temw,ptab(1),stab(1),visab)
visab(i) = visstab(i)*cqt
mtab(i)=0.
c
mold = ptab(i)/(stab(i)*visstab(i))
sum = 0.
fac = 10.
c
do 40 i=2,1000
c
pab=i*10.
pedu=pab/pstar
call xfracg(pab,pedu,tredu,stab,ab,egt)
call viscosity(spg,temw,stab,visab)
snum = pab/(stab*visab)
snum = sum + fac*snex + mold
mold = snum
c
c
mtab(i)=sum
c
ptab(i)=pab
c
stab(i)=stab vi stab(i)-visstab vi stab(i)*cqt
c
c
40 continue
return
END

Routine xfracg(ptab,pedu,tredu,stab)

REAL*4 invers, invcqt

AA = 0.06423
BB = 0.5533*TREDU - 0.6123
CC = 0.3151*TREDU - 1.0467 - 0.5785/TREDU**2,
DD = TREDU
EE = 0.6816/TREDU**2.
FF = 0.6845
GG = 0.273*TREDU

DERHO2 = 0.27*PREDU/TREDU

DO 10 ITER=1,20

c
DERHO2 = DERHO2*DERHO
DERHO4 = DERHO4*DERHO2
DERHO6 = DERHO6*DERHO4
DERHO8 = DERHO8*DERHO6

END

c
FUNC = AA*DERHO4 + BB*DERHO2 + CC*DERHO4 + DD*DERHO2 - GG
+4*ELDERHO2*(1. + FF*DERHO2)*EXP(-FF*DERHO2);
** franc.for **

```plaintext
C    FONPRI = 6. AA*DENR95 + 3. BB*DENR2 + 2. CC*DENR + 30
        4.4*DENR2 + (3. AA*DENR1 + (3. AA - 2. BB)*DENR2)*EXP(-AA*DENR2)
C    DEMR9 = DENR2 - FONC/FONPRI
C    IF (ARS(DENR9-DENR9),LT,0. SE-04) THEN
GO'TO 20
ENDIF
C 10 DENR9 = DENR9
C 20 STAB = (0.27*FREDU)/(DENR9*FREDU)
C    DENR2 = DENR9
C    DENR2 = DENR2*DENR2
C    DENR9 = DENR9*DENR9
C    FF2 = FF*FF
C    INVER = 1./DENR9
C    DEIDER = INVER*(3. AA + BB*DENR2 + CC*DENR2 + 4.4*DENR2)*EXP(-AA*DENR2)
C    INVCT = PTAB + (1. + DENR9 + DEIDER/STAB)
C    COT = 1./INVCT
C    Return
C******************************************************************************
C    Subroutine viscosity(npq,temv,ptab,tab,vstab)
C    zmolv = 28. 0625*npq
C    rho = ptab/zmolv*1.4955-03/(temv*(temv)
C    x = (9.37*90.015*temv)/(temv*1.5
C    x = x(205.219.2*temv)
C    z = 3.44 + 986.4/(temv + 0.0105*temv)
C    y = 2.447 + 0.2224*x
C    vstab = 3.8-04*x*exp(x*rho+y)
C    return
C******************************************************************************
C    Subroutine wellpro(tstar,ptab,wd,temv,temt1)
C    Dimension A(4), PZ(4)
C    Dimension PZ(1000), PAVE(1000), BI(1000), CI(1000), Di(1000)
C    common /tabl/ ptab(1000),ptab(1000),viscopt(1000)
C    common /tabl/ stab(1000),stab(1000),viscopt(1000)
C    Data A /0.125, 0.375, 0.375, 0.125/
C    MPWSM = 1000
C    PRSW = 10.
C    MPWSM = 1
A1 = (TEMV-TEMZ)/WD
C 10 HP = 1.
D = WD / 3.
DEPHT = WD
PRSW = PRSW
```
calculation of the average p/r

50 PAVE(NPRSWM)=A(1)*PZ(1)+A(2)*PZ(2)+A(3)*PZ(3)+A(4)*PZ(4)

if (NPRSW.M.E.NPRSWM) then
  pcal=pave+10,
  goto 10
endif

Call SPLINE(NPRSWM,P2Z,PAVE,B1,C1,D1)
do 70 i=1,NPRSWM
  deriv(i)=b(i)
70 continue
return
End

Subroutine ticartes(zmin,mmax,mz,mn,key)
Dimension ntab(13)
a=mz/multip
mmin=int(a)
mz=multip*mmin
b=mmax/multip
mmax=int(b+1)
mmax=multip*mmax

if (mmax==mmin) then
  goto 40
endif

if (mmax==mmax) then
  write(key,20) nzmin, ch5max, interm
  goto 40
endif

nzmin=0
do 10 iter=1,interm
  nzmin=nzmin+1
  ntab(iter)=nmtab
10 Continue

write(key,30) nzmin, ch5max, interm, (ch5tab(iter),iter=1,interm)

Subroutine change(ch5)
c5 character*5 ch5
character*90 ch10

c5 write(ch10,10)
  kindex = index(ch10,'A')
  kindex = index(ch10,'A')
  if (a.GE.1) then
    ch5=ch10(i)
  else
    ch5=ch10(kindex:kindex)
  endif

10 format (F10.4)
return
End
c***********************************************************************
c Subroutine Titles(key)
c character*5 title(12)
c
Title1='timec'
Title2='time1'
Title3='dimension'
Title4='tdpt1'
Title5='tdrx1'
Title6='xprec'
Title7='xmapc'
Title8='dimc'
Title9='dimension'
Title10='xmapi1'
Title11='ratec'
Title12='live1'

c write(key,10) (title(iter),iter=1,12)
c 10 format(12(A5,3X))
c return

c***********************************************************************
c Subroutine tranlog(par,iparcal,imin,imax)
c
par=nlog10(par)
c parcal=parcal-imin
iparcal=imin(parcal*10000./(imax-imin))
c return
end

c Subroutine tranac(par,iparcal,imin,imax)
c
par=parcal-imin
iparcal=imin(parcal*10000./(imax-imin))
c return
end

c Subroutine convert (type,totd,facini,dimea,prei,qs)
c
implicit real*4 (a-h,o-z)
c real*4 time(500),xpre(500),xmap(500),dimp(500),dm(500)
c real*4 znewmap(500),tpdt(500),tdrx(500),dnewriv(500),ypewtd(500)
c real*4 mpl
c dimension jval(500),kwal(500)
c character*10 type

c common /dur/ dur(15)
c
mpl=slookmp(prei)
dummy = 200.
if(type(1:2).eq.'BU').or.(type(1:2).eq.'bu') then
  tprod = dur(1)
else
  tprod = dummy
endf
character = Xaxislab
character = Yaxislab
character = Toplabel

if (lcatag == 2) then
  Write(18, *) Toplabel
  Write(18, *) Xaxislab
  Write(18, *) Yaxislab
  Do 10 1 = 1, 6000 Kauto(1)
  10 Continue
  Write(18, 6000) Lcatag
  Write(18, *) '0'
  If (lcatag == 2) then
    Write(18,*) '-1'
  Else
    Write(18,*) '0'
  Endif
  Write(18,*) '1'
  Write(18, 6000) Mpp
  Do 20 1 = 1, mpp
    Write(18, 6200) ldtycu(1), ldtecy(1)
  20 Continue
  Write(18,*) '0 10 5'
  If (lcatag == 2) then
    Write(18,*) '1'
    Write(18, 6000) Mpp
    Do 10 1 = 1, mpp
      Write(18, 6200) ldtycu(1), ldtecy(1)
  10 Continue
  Write(18,*) '0 10 5'
  Endif
  Write(18, 6000) Format(15)
  6200 Format(2(29.4,2X)) Return
End
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
IniApplication() - initializes window data and registers window
IniInstance() - 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 */
define NUMPTS 1000
define TAXIS 1
define TAXIS 2
define LOG 1
define CARTESIAN 2
define FIELD 1
define REGULAR 2
HANDLE hinst; /* current instance */

POINT Points[1000];

int XInfinity;
int YInfinity;
float XScale;
float YScale;

HWND hWndMain;
HMENU hMenu;
int GraphType;

HANDLE hMenuMainPts;
HANDLE hMenuLoadPts;
LPSTR pMenuPts;
int far * pIntPts;
LPSTR pLoadPts;
LPSTR pCurFts;
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 MXDetails[80] = "Pseudo-pressure vs. time (hours)";
char LogDetails[80] = "Delta mp vs. delta time (hours)";
char LogDeltaDetails[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/C2";
char RateDetails[80] = "Flow rate (Mcfd) vs. time (hours)";

char wWinLabel[120];
char wWinLabel[120];
char wMaxYLabel[120];
char wMaxYLabel[120];
int XDingHouse;
int YDingHouse;
float wXMinFloat;
float wXMaxFloat;
float wYMinFloat;
float wYMaxFloat;
int FileMode;

void CreatePoints(void);
int GetMax(void);
void FormatFloat(char *, double);
void DrawScales(HWND, int, int, double, double);
void NextTick(HWND, 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(int);
int ReadNextInt(int);
int ReadNextPt(int);
int SetPoints(void);
void DrawPoints(HWND);
int NumPoints(int);
void GetDetailLine(int, char *);
void SetupForms(int, char *);
int GetNextToken(char *, char *);
void DrawArea(void);
void DoStatusLine(HWND, LONG, int);
void UpdateStatusBox(int);
int ConvertPt(int, int, int);
void MarkerDraw(HWND, int, int);
void PrintVar(HDC);  
void PrintAxes(HDC);  
void PrintPoints(HWND);  
HDC GetDefPrinterDC(void);  

extern char OpenName[128];  

 mús FUnCt10n: 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 messages */  
    NULL, /* max window 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. */  
  }
}

 mús FUnCt10n: 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 window. */  

hInst = hInstance;  

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

cyDesk = GetSystemMetrics(SM_CXSCREEN);  

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.lpfnMenuName = "GenericMenu"; /* Name of menu resource in .RC files. */  
wc.lpfnClassName = "GenericWClass"; /* Name used in call to CreateWindow. */  
/* Register the window class and return */  
/* success/failure code. */  
return (RegisterClass(&wc));  

 mús FUnCt10n: MainWndProc(HWND, int)  
PURPOSE: handles WM_COMMAND messages  
COMMENTS:  

In this case, we post a WM_COMMAND message to the main window,  
which is processed by the MainWndProc(). It is responsible for  
handling all user input and updating the display.  

hInst = hInstance;  

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

cyDesk = GetSystemMetrics(SM_CXSCREEN);  

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.lpfnMenuName = "GenericMenu"; /* Name of menu resource in .RC files. */  
wc.lpfnClassName = "GenericWClass"; /* Name used in call to CreateWindow. */  
/* Register the window class and return */  
/* success/failure code. */  
return (RegisterClass(&wc));  

 mús FUnCt10n: MainWndProc(HWND, int)  
PURPOSE: handles WM_COMMAND messages  
COMMENTS:  

In this case, we post a WM_COMMAND message to the main window,  
which is processed by the MainWndProc(). It is responsible for  
handling all user input and updating the display.  

hInst = hInstance;  

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

cyDesk = GetSystemMetrics(SM_CXSCREEN);
```c
switch (message)
{
    case WM_CREATE:
        hWndClient = 0;
        xInit = 0;
        yInit = 0;
        xScale = 1.0;
        yScale = 1.0;
        dels = GetSystemMetrics(SM_CXFULLSCREEN);
        dey = GetSystemMetrics(SM_CYFULLSCREEN);
        hWndBut = CreateWindow("Button", "Input",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            (int)(0.02 * dels),
            (int)(0.02 * dey),
            75,
            20,
            hWnd,
            IDC_INPUTBUT, hinst, NULL);
        hWndBut = CreateWindow("Button", "Run",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            (int)(0.14 * dels),
            (int)(0.02 * dey),
            75,
            20,
            hWnd,
            IDC_RUNBUT, hinst, NULL);
        hWndBut = CreateWindow("Button", "Select",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            (int)(0.38 * dels),
            (int)(0.02 * dey),
            75,
            20,
            hWnd,
            IDC_SELECTBUT, hinst, NULL);
        hWndBut = CreateWindow("Button", "Select",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            (int)(0.38 * dels),
            (int)(0.02 * dey),
            75,
            20,
            hWnd,
            IDC_SELECTBUT, hinst, NULL);
        hWndBut = CreateWindow("Button", "Select",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            (int)(0.38 * dels),
            (int)(0.02 * dey),
            75,
            20,
            hWnd,
            IDC_SELECTBUT, hinst, NULL);
        hWndBut = CreateWindow("Button", "Select",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            (int)(0.38 * dels),
            (int)(0.02 * dey),
            75,
            20,
            hWnd,
            IDC_SELECTBUT, hinst, NULL);
```
hMenu = CreatePopupMenu();

AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_MDH, (LPSTR)"Miller-Drey-Hutchinson Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_LOGLOG, (LPSTR)"Log-Log Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_LOGLOGDERIV, (LPSTR)"Log-Log Derivative Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_CARTESIAN, (LPSTR)"Cartesian Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_HORNER, (LPSTR)"Horner Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_DIMENSIONLESS, (LPSTR)"Dimensionless Semi-Log Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_PRESSURE, (LPSTR)"Pressure History Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_LINE, (LPSTR)"Line Source Solution Plot");
AppendMenu(hMenu, MF_ENABLED | MF_STRING, IDC_RATERHIST, (LPSTR)"Rate History Plot");

break;

case WM_COMMAND: /* command from application menu */
    GetWindowText(hWnd, buf, 256);
    strpbrk(buf, "Miller-Drey-Hutchinson Plot.");
    switch (wParam) {
    case IDC_MDH:
        strcat(buf, "Miller-Drey-Hutchinson Plot.");
        if (OpenName[0] == 0)
            strcat(buf, "Open Name: ");
        strcat(buf, OpenName);
        SetWindowText(hWnd, (LPSTR)buf);
        GraphType = wParam;
        break;
    case IDC_LOGLOG:
        strcat(buf, "Log-Log Plot.");
        if (OpenName[0] == 0)
            strcat(buf, "Open Name: ");
        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, "Open Name: ");
        strcat(buf, OpenName);
        SetWindowText(hWnd, (LPSTR)buf);
        GraphType = wParam;
        break;
    case IDC_CARTESIAN:
        strcat(buf, "Cartesian Plot.");
        if (OpenName[0] == 0)
            strcat(buf, "Open Name: ");
        strcat(buf, OpenName);
        SetWindowText(hWnd, (LPSTR)buf);
        GraphType = wParam;
        break;
    case IDC_HORNER:
        strcat(buf, "Horner Plot.");
        if (OpenName[0] == 0)
            strcat(buf, "Open Name: ");
        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, "Open Name: ");
        strcat(buf, OpenName);
        SetWindowText(hWnd, (LPSTR)buf);
        GraphType = wParam;
        break;
    case IDC_PRESSURE:
        strcat(buf, "Pressure History.");
        if (OpenName[0] == 0)
            strcat(buf, "Open Name: ");
        strcat(buf, OpenName);
        break;
fran.c

FreeProcInstance(lpOpenDlg);
if ((strl(OpenName, "FLD") == NULL) &
   (strl(OpenName, "FSD") == NULL))
   FileMode = REGULAR;
else
   FileMode = FIELD;
if (rc)
   
   MessageBox (hWndMain,
   (LPSTR) "About to read & convert data points. This may require a few seconds to process."
   (LPSTR) "Output file",
   MB_OK);
   ReadDataFile();
   strcpy (buf, "Real Gas Flow. ");
   if (OpenName[0] == 0)
      
   strcpy (buf, "Output file: ");
   strcpy (buf, OpenName);
   hWndMain, (LPSTR)buf);
   break;
   case IDC_BATONIST:
   strcat (buf, "Rate History. ");
   if (OpenName[0] == 0)
      
   strcat (buf, "Output file: ");
   strcat (buf, OpenName);
   hWndMain, (LPSTR)buf);
   break;
   case IDC_MDI:
   strcat (buf, "Miller-Dyes-Hutchinson Plot. ");
   if (OpenName[0] == 0)
      
   strcat (buf, "Output file: ");
   strcat (buf, OpenName);
   hWndMain, (LPSTR)buf);
   break;
   case IDC_ABOUT:
   ifProcAbout = MakeProcInstance(About, hInst);
   DialogBox(hInst,
   "AboutBox",
   hWnd,
   ifProcAbout);
   FreeProcInstance(ifProcAbout);
   break;
   case IDC_ERASEBUT:
   XDialogBox = 0;
   EraseWindow(hWnd);
   break;
   case IDC_INPUTBUT:
   XDialogBox = 0;
   ifProcParam = MakeProcInstance(InputParam, hInst);
   rc = DialogBox(hInst,
   "PARAMS",
   hWnd,
   ifProcParam);
   FreeProcInstance(ifProcParam);
   break;
   case IDC_SELECTBUT:
   lpOpenDlg = MakeProcInstance((LPSTR)OpenSave, hInst);
   rc = DialogBox(hInst, "OPEN", hWnd, lpOpenDlg);
   break;
   case IDC_OPENBUT:
   lpOpenDlg = MakeProcInstance((LPSTR)OpenSave, hInst);
   rc = DialogBox(hInst, "OPEN", hWnd, lpOpenDlg);
   break;
return (DefWindowProc(hWnd, message, wParam, lParam));

FUNCTION: GraphProc()

PURPOSE: Processes messages

COMMENTS:

long FAR PASCAL GraphProc(hWnd, message, wParam, lParam)
    hWnd; // window handle
    word; // type of message
    long; // additional information

switch (message)
    case WM_CREATE:
        break;
    case WM_MOUSEMOVE:
        if (wParam == WM_LBUTTONDOWN)
            break;
    case WM_LBUTTONDOWN:
        // Handle vertical status line info for current position.
        DoStatusLine(hWnd, lParam, RM_NOSTATUS);
        break;
    
    return (DefWindowProc(hWnd, message, wParam, lParam));

FUNCTION: InputParams(hWnd, message, wParam, lParam)

PURPOSE: Gets input parameters for the dialog box.

COMMENTS:

BOOL FAR PASCAL InputParams(hWnd, message, wParam, lParam)
    hWnd; // window handle of the dialog box
    word; // type of message
    long; // message-specific information

switch (message)
    case WM_CREATE:
        SendDlgItemMessage(hWnd, IDC_TYPE, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_RATE, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_THICKNESS, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_WELDDEPTH, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_TOLERANCE, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_MAIN, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_WELLMASK, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_FAIL, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_POROUSITY, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_RS, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_RESTORE, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_SHUTDOWN, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_SKIN, EM_LIMITTEXT, 40, NULL);
        SendDlgItemMessage(hWnd, IDC_SPECGRAV, EM_LIMITTEXT, 40, NULL);
    
    return (FALSE); // Didn't process a message.

FUNCTION: About(HWND, lParam, wParam, lParam)

PURPOSE: Processes messages for "About" dialog box.

COMMENTS:

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

Wait for user to click on "OK" button, then close the dialog box.
fran.c

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

(PATHS: 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:
    if (ValidateEntries(hDlg)) {
      GetDlgItemText(hDlg, IDC_MAIN, (LPSTR)&buf[80]);
      strcat(buf, "error!");
      hdi = OpenFile((LPSTR)FileName, "create", ofSTRUCT, of_READWRITE);
      if (hdi == -1)
        strcpy(buf2, "Unable to create the file ");
      strcat(buf2, FileName);
      MessageBox(hDlg, (LPSTR)buf2, NULL, MB_OK | MB_ICONHAND);
      EndDialog(hDlg, FALSE);
      ReadInputFile(hdi, hWnd);
    } break;
  case IDC_OK:
    if (ValidateEntries(hDlg)) {
      GetDlgItemText(hDlg, IDC_MAIN, (LPSTR)&buf[80]);
      strcat(buf, "error!");
      hdi = OpenFile((LPSTR)FileName, "create", ofSTRUCT, of_READWRITE);
      if (hdi == -1)
        strcpy(buf2, "Unable to create the file ");
      strcat(buf2, FileName);
      MessageBox(hDlg, (LPSTR)buf2, NULL, MB_OK | MB_ICONHAND);
      EndDialog(hDlg, FALSE);
      WriteInputFile(hdi, hWnd);
      _close(hdi);
      hd = OpenFile(LPSTR)"input.txt", ofSTRUCT, of_READWRITE);
      len = strlen(buf);
      _write(hdi, (LPSTR)buf, len);
      _close(hdi);
      EndDialog(hDlg, TRUE); /* Exits the dialog box */
      return (TRUE);
    } break;
  case IDC_CANCEL:
    EndDialog(hDlg, TRUE); /* Exits the dialog box */
    break;
}
fran.c

return(FALSE);

if (CheckText(hDlg, IDC_WELDEPTH, "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_H2S, "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_RETEMP, "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_WELLCORE, "No field Data file entered") == FALSE)
    return(FALSE);

if (CheckText(hDlg, IDC_AUTO, "No automate curve indicator entered") == FALSE)
    return(FALSE);
/* FillRect(hdc, rect, GetStockObject(white_brush)); */
CreatePen(INVALID_HANDLE_VALUE);
max = GetMaxHr;
SetMapMode(hdc, MM_ANISOTROPIC);
xfact = 1000.0 / XScale;
if (xfact < 0)
  xfact = -30000.0;
ystart = (((float)maxy - Yinit) / YScale);
if (ystart < 0)
  ystart = -30000.0;
SetWindowExt(hdc, (int)xfact, (int)ystart);
SetViewportExt(hdc, Xinit, Yinit);
SetViewportOrg(hdc, 0, 0);
HPen = CreatePen(dp_SOLID, 1, RGB(0, 0, 255));
holdPen = SelectObject(hdc, HPen);
Polyline(hdc, (LPPOINT)points, NUMPTS);
SelectObject(hdc, holdPen);
DeleteObject(hPen);
DrawScales(NWMAIN, Xinit, Yinit, xfact, ystart);
ReleaseDC(NWmd, hdc);
*/

/**
 *  Function Name PrintVar()
 *  Parameters    hdc    hdc ; Handle of device context to the printer.
 */
void PrintVar(hdc)

int far * tempptr;
int i;
int gptr;
int j;
int kp;
int yptr;
BOOL found;
char buf[255];
HPEN hPen;
HPEN hPenOrg;
int delx;
int dy;
int winy;
int winz;
int delay;
int dp;
BOOL first;

/* Create pen handle */
/* graphs. */
/* Start by selecting the pen */

return(TRUE);

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

// CheckText()
// PURPOSE:
// COMMENTS:

BOOL CheckText(HWND hwnd, int id, char * errstr)

if (GetDlgItemText(hwnd, id, (LPSTR)buf, 80) == 0)

MessageBox(hwnd, (LPSTR)errstr, (LPSTR)"Parameters Entry",
  MB_OK | MB_ICONHAND);
SetFocus(GetDlgItem(hwnd, id));
return(FALSE);

return(TRUE);

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

// DrawWindow()
// PURPOSE:
// COMMENTS:

void DrawWindow(HWND hwnd)

HDC hdc;

HDC hdc = GetDC(hwnd);
GetClientRect(hwnd, &rect);
delx = rect.right - rect.left;
dely = rect.bottom - rect.top;
fran.c

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:

FUNCTION: GetDefPrinterDC()
PURPOSE:
DESCRIPTION: Retrieves printer device from Win.ini and opens a context to it.
PARAMETERS: None.

HDC GetDefPrinterDC()
{
    hdc;
    int cch;
    PSTR pDriver;

    hdc = GetDefPrinterDC();
}
void DrawPoints(HWND hWnd)
{
    HDC hDC; RECT rect;
    int delx; int dely;
    int maxx; int maxy;
    float xfact; float yfact;
    HPEN hPen;
    hPen = SelectObject(hDC, hPen);
    int i; int j;
    GetClientRect(hWnd, &rect);
}

fran.c

delx = rect.right - rect.left;
dely = rect.bottom - rect.top;
SetPoints();
if (PrintIt == TRUE)
{
    PrintPoints(hWnd);
    return;
}
maxw = 10000;
maxh = 10000;
hDC = GetDC(hWnd);
SetMapMode(hDC, MM_ANISOTROPIC);
xfact = (float)(maxX / XScale);
yfact = (float)(maxY / YScale);
SetWindowExt(hDC, (int)xfact, (int)yfact);
SetViewportOrg(hDC, 0, 0);
SetViewportOrg(hDC, 0, dely);
SetViewportOrg(hDC, 0, 0);

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

// Polyline(hDC, (LPOINTER)Points, NumPts); /*0-10-02-91*/
if (Filemode == FIELD)
    Polyline(hDC, (LPOINTER)Points, NumPts-1);
else
    for (i = 0; i < NumPts - 1; ++i)
    {
        MoveTo(hDC, Point[i].x, Point[i].y);
        SetPixel(hDC, Point[i].x, Point[i].y, RGB(255, 0, 0));
        for (j = 0; j < 40; ++j)
        {
            SetPixel(hDC, Point[i].x + j, Point[i].y, RGB(255, 0, 0));
            for (j = 0; j < 40; ++j)
            {
                SetPixel(hDC, Point[i].x, Point[i].y + j, RGB(255, 0, 0));
                for (j = 0; j < 40; ++j)
                {
                    SetPixel(hDC, Point[i].x, Point[i].y - j, RGB(255, 0, 0));
                }
            }
        }
    }
    SelectObject(hDC, hOldPen);
    ReleaseDC(hWnd, hDC);
}
franc.c

void FrameWindow(HWND hWnd)
    hWnd hWnd;
{
    HDC hdc;
    RECT rect;
    int delay;
    int delay;
    int max;
    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 hWnd;
{
    int delay;
    int delay;
    HDC hdc;

    delay = GetDeviceCaps(hdc, HORZRES);
    delay = GetDeviceCaps(hdc, VERTRES);
    hdc = GetDC(hWnd);
    // Rectangle(hdc, (int) 0.1 * delay), (int) (0.15 * delay),
    // (int) (0.3 * delay), (int) (0.9 * delay);
    Rectangle(hdc, 50, 50, 520, 300);
    ReleaseDC(hWnd, hdc);
}

FUNCTION: FormatFloat()
PURPOSE:
COMMENTS:

void FormatFloat(recest, fitval)
char * recstr;
double fitval;
{
    char utilbuf[80];
    char * fpstr;
    char * utilptr;
    int dec;
    int sign;
    int len;

    /* Utility to format floating pt number into */
    /* a string. */
    fpstr = fprint((float) fitval, 2, 4dec, 4sign);

```c
if (fptr != 0) {
    if (align)
        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, "0.0");
    strcat (recstr, fptr);
}
strcat (recstr, " ");
}

/* Function: DrawScales() */

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);
```
fran.c

switch (GraphType)
{
    case IDC_LOGLOGERRV:
        xDetails = Col2Details;
        yDetails = Col1Details;
        hdrDetails = LogDerivDetails;
        break;
    case IDC_LOGLOG:
        xDetails = Col2Details;
        yDetails = Col1Details;
        hdrDetails = LogDetails;
        break;
    case IDC_CARTESIAN:
        xDetails = Col1Details;
        yDetails = Col7Details;
        hdrDetails = CartesianDetails;
        break;
    case IDC_NONLINE:
        xDetails = Col4Details;
        yDetails = Col7Details;
        hdrDetails = NonLinearDetails;
        break;
    case IDC_DIMENSIONLESS:
        xDetails = Col3Details;
        yDetails = Col6Details;
        hdrDetails = DimensionlessDetails;
        break;
    case IDC_PRESSURE:
        xDetails = Col1Details;
        yDetails = Col6Details;
        hdrDetails = PressureDetails;
        break;
    case IDC_LINE:
        xDetails = Col5Details;
        yDetails = Col7Details;
        hdrDetails = LineDetails;
        break;
    case IDC_RATEERROR:
        xDetails = Col1Details;
        yDetails = Col3Details;
        hdrDetails = RateErrors;
        break;
    case IDC_MIN:
        xDetails = Col2Details;
        yDetails = Col7Details;
        hdrDetails = MedianDetails;
        break;
}

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

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

/* Draw out title, xmin, xmax, ymin, and ymax values */
xCur = 80;
yCur = 78;
TextOut(NDC, xCur, yCur, hdrDetails, strlen(hdrDetails));
xCur = 45;
yCur = 420;
TextOut(NDC, xCur, yCur, XMinLabel, strlen(XMinLabel));

xBc = 680;
yCur = 470;
TextOut(NDC, xCur, yCur, XMaxLabel, strlen(XMaxLabel));
xCur = 55;
yCur = 390;
TextOut(NDC, xCur, yCur, YMinLabel, strlen(YMinLabel));
xCur = 55;
yCur = 40;
TextOut(NDC, xCur, yCur, YMaxLabel, strlen(YMaxLabel));

delta = 555 / (XTickPct + 1);
xCur = 455;
yCur = 470;
delx = 10000;
dely = 10000;
xCur = 0;

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

/* Draws Y tick values, and horz line representing the tick */
xCur = 55;
yCur = 390;
yCur2 = 0;
delta = -(delx / (XTickPct + 1));
delta2 = -(dely / (XTickPct + 1));
for (i = 0; i < YTickPct; i++)
{
    yCur = yCur + delta;
    yCur2 = yCur2 + delta2;
    TextOut(NDC, xCur, yCur, YTickValue[i], strlen(YTickValue[i]));
    MoveTo(NDC, 500, yCur2);
    LineTo(NDC, 9500, yCur2);
}

void DrawAxes()
{
    /* FRANCOIS */
    FOR Purpose:
    COMMENTS:
   ܨ

FUNCTION: DrawAxes()
franc

void xDetails = Col1Details;
hdDetails = RateDetails;
break;

case IDC_MDH:
    xDetails = Col2Details;
yDetails = Col2Details;
hdDetails = MDHDetails;
break;

SetupForms(TAXIS, xDetails);
SetupForms(TAXIS, yDetails);

xCur = 80;
yCur = 70;
TextOut(hdc, xCur, yCur, hdrDetails, strlen(hdrDetails));
xCur = 45;
yCur = 420;
TextOut(hdc, xCur, yCur, XMLabel, strlen(XMLabel));
xCur = 69;
yCur = 420;
TextOut(hdc, xCur, yCur, XMLabel, strlen(XMLabel));
xCur = 57;
yCur = 390;
TextOut(hdc, xCur, yCur, TMLabel, strlen(TMLabel));
xCur = 57;
yCur = 40;
TextOut(hdc, xCur, yCur, XMLabel, strlen(XMLabel));

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

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

delta2 = del / (XMntickPts + 1);
for (i = 0; i < XMntickPts; i++)
    
    xCur = xCur + del;
xCur2 = xCur2 + del2;
TextOut(hdc, xCur, yCur, XTickValue[i], strlen(XTickValue[i]));
MoveTo(hdc, xCur2, xCur2, rect.bottom);
LineTo(hdc, xCur2, xCur2, rect.top);
MoveTo(HDC2, rect.left, yCur);
LineTo(HDC2, rect.right, yCur);

*/
xPt = x1;
for (i = 0; i <= 10; i++)
    NextTick(HDC, xCur, xPt, XAXIS);
    xCur += 55;
    xPt += (int)(yFact * 0.1);

yCur = y2;
for (i = 0; i <= 10; i++)
    NextTick(HDC, yCur, yPt, YAXIS);
yCur -= 35;
yPt += (int)(yFact * 0.1);

*/
ReleaseDC(hWnd, HDC);
ReleaseDC(hGraphMod, HDC);

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

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, rembuf)
char * curptr;
char * rembuf;

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

ptr = curptr;

cnt = 0;

while(TRUE)

    if (*ptr == ' ')
    {
        *ptr = ' ';
    }
    if (*ptr == '
    {
        *ptr = '\n';
    }
    if (*ptr == '\n')
    {
        *ptr = ' ';
    }
    if (*ptr == ' ')
    {  
        *ptr = ' ';
    }
    if (*ptr == ' 
    {  
        *ptr = ' ';
    }
    if (*ptr == ' ')
```c
("ptr = '3'") ++
("ptr = '3'") ++
("ptr = '4'") ++
("ptr = '4'") ++
("ptr = '5'") ++
("ptr = '5'") ++
("ptr = '6'") ++
("ptr = '6'") ++
("ptr = '7'") ++
("ptr = '7'") ++
();
break;
resbuf = *ptr;
resbuf++;  
ptr++;
}
while (TRUE)
{
    if ("ptr = '7'") ++
("ptr = '8'") ++
("ptr = '9'") ++
("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: NextTick()

PURPOSE:

COMMENTS:

void NextTick(HDC, pos, val, axis)
HDC hdc;
int pos;
int val;
int axis;

char buf[80];

for (val = pos, buf[10];

if (axis == XAXIS)
    TextOut(hdc, 39, pos, buf, strlen(buf));
if (axis == YAXIS)
    TextOut(hdc, pos, 420, buf, strlen(buf));

FUNCTION: WriteinputFile()

PURPOSE:

COMMENTS:

void WriteInputFile(hdl, hdig)
int hdl;
int hdig;
char buf[80];

//
WriteInputElement(hdl, hdig, IDC_TITLE);
WriteInputElement(hdl, hdig, IDC_MAIN);
WriteInputElement(hdl, hdig, IDC_MAIN);
WriteInputElement(hdl, hdig, IDC_TYPE);
WriteInputElement(hdl, hdig, IDC_WINDOW);
WriteInputElement(hdl, hdig, IDC_RADIUS);
WriteInputElement(hdl, hdig, IDC_DAMAGED);
WriteInputElement(hdl, hdig, IDC_WELL);
WriteInputElement(hdl, hdig, IDC_WELLDEPTH);
WriteInputElement(hdl, hdig, IDC_THICKNESS);
WriteInputElement(hdl, hdig, IDC_FOSSITY);
WriteInputElement(hdl, hdig, IDC,:),
WriteInputElement(hdl, hdig, IDC_B32);
WriteInputElement(hdl, hdig, IDC_CO2);
WriteInputElement(hdl, hdig, IDC_CALIB);
WriteInputElement(hdl, hdig, IDC_RESTART);
WriteInputElement(hdl, hdig, IDC_SPECTRUM);
WriteInputElement(hdl, hdig, IDC_RADIUSS);
WriteInputElement(hdl, hdig, IDC_WELL);
WriteInputElement(hdl, hdig, IDC_WELLDEPTH);
WriteInputElement(hdl, hdig, IDC_THICKNESS);
WriteInputElement(hdl, hdig, IDC_FOSSITY);
WriteInputElement(hdl, hdig, IDC_SPECTRUM);
WriteInputElement(hdl, hdig, IDC_B32);
WriteInputElement(hdl, hdig, IDC_CO2);
WriteInputElement(hdl, hdig, IDC_CALIB);
WriteInputElement(hdl, hdig, IDC_RESTART);
WriteInputElement(hdl, hdig, IDC_SPECTRUM);
WriteInputElement(hdl, hdig, IDC_RADIUSS);
WriteInputElement(hdl, hdig, IDC_WELL);
WriteInputElement(hdl, hdig, IDC_WELLDEPTH);
WriteInputElement(hdl, hdig, IDC_THICKNESS);
WriteInputElement(hdl, hdig, IDC_FOSSITY);
WriteInputElement(hdl, hdig, IDC_SPECTRUM);
WriteInputElement(hdl, hdig, IDC_B32);
WriteInputElement(hdl, hdig, IDC_CO2);
WriteInputElement(hdl, hdig, IDC_CALIB);
WriteInputElement(hdl, hdig, IDC_RESTART);
WriteInputElement(hdl, hdig, IDC_SPECTR...
/*
 * ReadInputFile(hdl, hdlg)
 * 
 * ReadInputFile returns a read file handle for reading an input file.
 * 
 * Return Values:
 * 0 = Success
 * E = Error
 * 
 */

void ReadInputFile(hdl, hdlg)
    int hdl;
    int hdlg;
{
    /*
     * ReadInputFile returns a read file handle for reading an input file.
     * 
     * Return Values:
     * 0 = Success
     * E = Error
     * 
     */

    _close(hdl);
}

FUNCTION: ReadInputFile
PURPOSE:
COMMENTS:

void ReadInputFile(hdl, hdlg)
    int hdl;
    int hdlg;
{
    /*
     * ReadInputFile returns a read file handle for reading an input file.
     * 
     * Return Values:
     * 0 = Success
     * E = Error
     * 
     */

    _close(hdl);
}
FUNCTION: ReadInputElement()

PURPOSE:

COMMENTS:

*******************************************************************************
void ReadInputElement(hdl, hdlg, id)
int    hdl;
int    hdlg;
int    id;

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

buf[0];
len = 0;
while (TRUE)
{
    len = _read(hdl, buf, 1);
    if (buf[0] == 'r' || buf[0] == 'R' || len == 0)
        break;
    if (buf[0] == 'r')
        { 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    id;
int    j;
BOOL   set;
char   buf[20];
int    len;
DWORD  alloc;
DWORD  rbytes;
*******************************************************************************

set = FALSE;

hdl = OpenFile((LPSTR)OpenName, (LPSTRSTRUCT)OFStruct, OF_READ);
if (hdl == -1)
    MessageBox (hwndMain, (LPSTR)"hdl == NULL",
            (LPSTR)"Alloc",
            MB_OK | MB_ICONHAND);
 Racing
    if ((strrchr(OpenName, '.') == NULL) &&
            (strrchr(OpenName, '.'.filo) == NULL)
            FileMode = REGULAR;
            else
                FileMode = FIELD;
                NumPts = NumPoints(hdl);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                GetDetailLine(hdl, CollDetails);
                if (pMemAscLits)
                    GlobalUnlock(pMemAscLits);
                    GlobalFree(pMemAscLits);
                    pMemAscLits = GlobalAlloc(GMEM_MOVEABLE, (DWORD)NumPts * (DWORD)24 + (DWORD)100);
                    if (pMemAscLits == NULL)
                        MessageBox (hwndMain, (LPSTR)"pMemAscLits NULL",
                                (LPSTR)"Alloc",
                                MB_OK | MB_ICONHAND);
                        pMemAscLits = GlobalLock(pMemAscLits);
                        if (pMemAscLits == NULL)
                            MessageBox (hwndMain, (LPSTR)"pMemAscLits NULL",
                                    (LPSTR)"Alloc",
                                    MB_OK | MB_ICONHAND);
                            rembytes = rbytes;
                            alloc = (DWORD)rembytes + (DWORD)1500;
                            rbytes = GlobalAlloc(GMEM_MOVEABLE |GMEM_NOT_BANKED, alloc);
                            if (rbytes == NULL)
                                MessageBox (hwndMain, (LPSTR)"rbytes NULL",
                                                        (LPSTR)"Alloc",
                                                        MB_OK | MB_ICONHAND);
                                pLoadPts = GlobalLock(pMemLoadPts);
                                hdl = OpenFile((LPSTR)LoadName, (LPSTR)OFStruct, OF_READ);
                                if (hdl == -1)
                                    MessageBox (hwndMain, (LPSTR)"hdl == NULL",
                                            (LPSTR)"Alloc",
                                            MB_OK | MB_ICONHAND);
                                            pLoadPts = GlobalLock(pMemLoadPts);
fran.c

if (pLoadPts == NULL)
    MessageBox
        hwndMain,
        (LPCTSTR)"pLoadPts NULL",
        (LPCTSTR)"Abort",
        MB_OK | MB_ICONHAND;

pIntPps = (int *)pAscPlPts;
pCurPts = pLoadPts;

i = i_read(hdl, (LPSTR)pLoadPts, (int)pCurPts + 12 * 6) + 1000;

for (i = 0; i < NumPts; ++i)
{
    for (j = 0; j < 10; j++)
    {
        dataPt = ReadNextPt(hdl);

        *pIntPps = dataPt;
        pIntPps++;
    }
}

GlobalUnlock(hwndLoadPts);
GlobalFree(hwndLoadPts);
_i_close(hdl);

MessageBox
    hwndMain,
    (LPCTSTR)"Completed reading of output file",
    (LPCTSTR)"OK",
    MB_OK;

FUNCTION: ReadNextPt()
PURPOSE:
COMMENTS:

FUNCTION: ReadNextInt()
PURPOSE:
COMMENTS:

FUNCTION: GetDetailLine()
PUSH hive

for (i = 0; i < NumPmts; i++)
    xpt = (int)(((float)(pintPts + xoffset - 1)) * TScale);
    ypt = (int)(((float)(pintPts + yoffset - 1)) * TScale);
    Points[i].x = xpt + Xinit;
    Points[i].y = ypt + Yinit;

pintPts = pintPts + 12;

FUNCTION: NumPoints()
LineTo(hdc2, XLogMouse, YLogMouse);
  // SelectObject(hdc2, hOldPen);
  // DeleteObject(hPen);
  ReleaseDC(hWd, hdc2);
  /* Write out digit values reflecting graph inter- */
  /* sections with new vertical line. */
  UpdateStatusBox(XLogMouse);
}

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

Function Name - UpdateStatusBox()
Description - Draw vertical line indicating current position as reflected
by the mouse.
Parameters - hWnd : Window handle to draw into.
              iParam : Value containing x- and y- mouse pos.
              rop : Indicates logical drawing mode.

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

void UpdateStatusBox(xpt)
int  xpt;
  int  dp;
  float xconv;
  float yconv;
  int  l;
  HDC hdc;
  char buf[40];
  char buf2[40];
  int  x;
  int  y;
  int  xmode;
  int  ymode;

  dp = ((int)((float)xpt * (float)10000) / (float)3350);
  for (l = 0; l < NumTps; l++)
    if ((Points[l].x > dp) && (GraphType != IDC_HORNER))
      if ((Points[l].x < dp) && (GraphType == IDC_HORNER))
        switch (GraphType)
          case IDC_LOGLOG:
            xmode = LOG;
            ymode = LOG;
            break;
          case IDC_LogLogInv:
            xmode = LOG;
            ymode = LOG;
            break;
          case IDC_CARTEsIAN:
            xmode = CARTESIAN;
            ymode = CARTEsIAN;
            break;
          case IDC_HORNER:
            xmode = LOG;
            ymode = CARTESIAN;
            break;
          case IDC_DIMENSIONLESS:
```c
int pt;
int mode;
{
    float conva;
    int intval;
    float delmaxmin;
    float delmaxminlog;
    double a;
    float logval;

    if (mode == CARTESIAN)
    {
        if (axis == AXIS)
        {
            delmaxmin = MAXFloat - XMInFloat;
            conva = (((float)pt + delmaxmin) / (float)550 + XMInFloat);
        }
        else
        {
            delmaxmin = MAXFloat - YMInFloat;
            conva = (((float)pt + delmaxmin) / (float)10000 + YMInFloat);
        }
        return (conval);
    }
    if (mode == LOG)
    {
        if (axis == AXIS)
        {
            delmaxminlog = log10(MAXFloat) - log10(YMInFloat);
            a = (((float)pt + delmaxminlog) / (float)550 +
                 log10(YMInFloat));
        }
        else
        {
            delmaxminlog = log10(MAXFloat) - 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
int x;
int y;
{
    DWORD dwColor;
    int is;
```
**fran.c**

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);
}
//GenericMenu MENU
//BEGIN
// POPUP   "Input..."
// BEGIN
//   MENUITEM "Input Parameters...", IDM_PARAM
//   END
// POPUP   "$Help"
// BEGIN
//   MENUITEM "$About...", IDM_ABOUT
//   END
//END
; 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
```c
#include <windows.h>
#include <io.h>
#include <string.h>
#include "filemgr.h"

char FileName[128];
char PathName[128];
char OpenName[128];
char SaveName[128];
char DefPath[128];
char DefSpec[3] = ".*";
char DefExt[] = ".txt";
char str[255];

BOOL DeletingFiles;

HANDLE FAR PASCAL OpenSave(HWND, message, wParam, lParam);

CASE LAN_SELCHANGE:
    // If item is a directory name, append ".*" /
    if (lParamSelect(hWndg, IDC_LISTBOX))
        strcat(str, DefSpec);
    SetDlgItemText(hWndg, IDC_EDIT, str);
    SendMessage(hWndg, IDC_EDIT, EM_SETSEL, NULL, MAKELONG(0, 0xFFFF));
    break;

case LAN_DBLCLK:
    goto openfile;
    return (TRUE);

openfile:
    GetDlgItemText(hWndg, IDC_EDIT, OpenName, 128);
    if (OpenName[0] || OpenName[2])
        { separateFile(hWndg, (LPSTR) str, (LPSTR) DefSpec, (LPSTR) OpenName);
            if (OpenName[1])
                strcpy(DefPath, str);
            ChangeDefExt(DefExt, DefSpec);
            UpdateListbox(hWndg);
            return (TRUE);
        }
    if (OpenName[0] == 'No filename specified.
        MessageBox(hWndg, NULL, MB_OK | MB_ICONHAND);
        return (TRUE);
    }
    AddExt(OpenName, DefExt);
    EndDialog(hWndg, hfile);
    return (TRUE);

case IDCANCEL:
    OpenName[0] = 0;
    EndDialog(hWndg, NULL);
    return (FALSE);
    break;

case WM_INIT鲢:
    /* message: initialize */
    OpenName[0] = 0;
    UpdateListbox(hWndg);
    SetDlgItemText(hWndg, IDC_EDIT, DefSpec);
    SendMessage(hWndg, IDC_EDIT, EM_SETSEL, NULL, MAKELONG(0, 0xFFFF));
    SetFocus(GetDlgItem(hWndg, IDC_EDIT);
    ShowWindow(GetDlgItem(hWndg, IDC_DISPLAY), SW_HIDE);
```
return (FALSE); /* Indicates the focus is set to a control */

/*
 | return FALSE;
 |*/

/*
 | Function Name - UpdateListBox()
 | Description -
 | Parameters -
 */

void UpdateListBox(HWND hWnd)

strcpy(Ext, DefPath);
strcat(Ext, DefSpec);
DisplayList(hWnd, str, IDC_LISTBOX, IDC_PATH, 0x4010);

/* To ensure that the listing is made for a subdir. of
 * current drive dir... */
if ((strchr (DefPath, '.'))
DisplayList(hWnd, DefSpec, IDC_LISTBOX, IDC_PATH, 0x4010);

/* Remove the '.' character from path if it exists, since this
 * will make DisplayList move us up an additional level in the tree
 * when UpdateListBox() Is called again. */
if (strchr (DefPath, '..'))
DefPath[0] = '\0';
SetDlgItemText(hWnd, IDC_EDIT, DefSpec);

/*
 | Function Name - ChangeDefEx
 | Description -
 | Parameters -
 */

void ChangeDefEx(HWND hWnd, PSTR Ext, Name, p

strcpy(DefPath, p);
while (*p && *p != '\0')

if (*p
if (strchr(p, '.') && strchr(p, '?'))
strcpy(p, p + 1);

Function Name - AddExt()
Description -
Parameters -

void AddExt(HWND hWnd, PSTR Name, Ext, p

/*
 | Function Name - SeparateFile()
 | Description -
 | Parameters -
 */

void SeparateFile(HWND hWnd, lpDestPath, lpSrcPath, lpSrcFileName)

LPSTR lpTmp;
char ctmp;
if (strncpy(lpDestPath, lpSrcPath, lpSrcFileName) + 1 - 0;

if (strncpy(lpDestPath, lpSrcPath, lpSrcFileName) + 1 - 0;

}
/** 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 AddFile(LPSTR, LPSTR); 
void ChangeDefExt(LPSTR, LPSTR); 

132