DEVELOPMENTS IN WATER SCIENCE

37

D. CLARKE

GROUNDWATER DISCHARGE TESTS: SIMULATION AND ANALYSIS

ELSEVIER

GROUNDWATER DISCHARGE TESTS: SIMULATION AND ANALYSIS

OTHER TITLES IN THIS SERIES

1 G. BUGLIARELLO AND F. GUNTER
COMPUTER SYSTEMS AND WATER RESOURCES
2 H.L. GOLTERMAN
PHYSIOLOGICAL LIMNOLOGY
3 Y.Y. HAUMES, W.A. HALL AND H.T. FREEDMAN
MULTIOBJECTIVE OPTIMIZATION IN WATER RESOURCES SYSTEMS:
THE SURROGATE WORTH TRADE-OFF-METHOD
4 J.J. FRIED
GROUNDWATER POLLUTION
5 N. RAJARATNAM
TURBULENT JETS
6 D. STEPHENSON
PIPELINE DESIGN FOR WATER ENGINEERS
7 V. HÁLEK AND J. ŠVEC
GROUNDWATER HYDRAULICS
8 J. BALEK
HYDROLOGY AND WATER RESOURCES IN TROPICAL AFRICA
9 T.A. McMAHON AND R.G. MEIN
RESERVOIR CAPACITY AND YIELD
10 G. KOVÁCS
SEEPAGE HYDRAULICS
11 W.H. GRAF AND C.H. MORTIMER (EDITORS)
HYDRODYNAMICS OF LAKES: PROCEEDINGS OF A SYMPOSIUM
12–13 OCTOBER 1978, LAUSANNE, SWITZERLAND
12 W. BACK AND D.A. STEPHENSON (EDITORS)
CONTEMPORARY HYDROGEOLOGY: THE GEORGE BURKE MAXEY MEMORIAL VOLUME
13 M.A. MARIÑO AND J.N. LUTHIN
SEEPAGE AND GROUNDWATER
14 D. STEPHENSON
STORMWATER HYDROLOGY AND DRAINAGE
15 D. STEPHENSON
PIPELINE DESIGN FOR WATER ENGINEERS
(completely revised edition of Vol. 6 in the series)
16 W. BACK AND R. LÉTOLLE (EDITORS)
SYMPOSIUM ON GEOCHEMISTRY OF GROUNDWATER
17 A.H. EL-SHAARAWI (EDITOR) IN COLLABORATION WITH S.R. ESTERBY
TIME SERIES METHODS IN HYDROSCIENCES
18 J. BALEK
HYDROLOGY AND WATER RESOURCES IN TROPICAL REGIONS
19 D. STEPHENSON
PIPEFLOW ANALYSIS
20 I. ZAVOIANU
MORPHOMETRY OF DRAINAGE BASINS
21 M.M.A. SHAHIN
HYDROLOGY OF THE NILE BASIN
22 H.C. RIGGS
STREAMFLOW CHARACTERISTICS
23 M. NEGULESCU
MUNICIPAL WASTEWATER TREATMENT
24 L.G. EVERETT
GROUNDWATER MONITORING HANDBOOK FOR COAL AND OIL SHALE DEVELOPMENT
25 W. KINZELBACH
GROUNDWATER MODELLING: AN INTRODUCTION WITH SAMPLE PROGRAMS IN BASIC
26 D. STEPHENSON AND M.E. MEADOWS
KINEMATIC HYDROLOGY AND MODELLING
27 A.M. EL-SHAARAWI AND R.E. KWIATKOWSKI (EDITORS)
STATISTICAL ASPECTS OF WATER QUALITY MONITORING – PROCEEDINGS OF THE WORKSHOP HELD AT
THE CANADIAN CENTRE FOR INLAND WATERS, OCTOBER 1985
28 M. JERMAR
WATER RESOURCES AND WATER MANAGEMENT
29 G.W. ANNANDALE
RESERVOIR SEDIMENTATION
30 D. CLARKE
MICROCOMPUTER PROGRAMS IN GROUNDWATER
31 R.H. FRENCH
HYDRAULIC PROCESSES IN ALLUVIAL FANS
32 L. VOTRUBA, Z. KOS, K. NACHÁZEL, A. PATERA AND V. ZEMAN
ANALYSIS OF WATER RESOURCE SYSTEMS
33 L. VOTRUBA AND V. BROŽA
WATER MANAGEMENT IN RESERVOIRS
34 D. STEPHENSON
WATER AND WASTEWATER SYSTEMS ANALYSIS
35 M.A. CELIA ET AL.
COMPUTATIONAL METHODS IN WATER RESOURCES, VOLUME 1 MODELING SURFACE AND SUB-SURFACE FLOWS. PROCEEDINGS OF THE VII INTERNATIONAL CONFERENCE, MIT, USA, JUNE 1988
36 M.A. CELIA ET AL.
COMPUTATIONAL METHODS IN WATER RESOURCES, VOLUME 2 NUMERICAL METHODS FOR TRANSPORT AND HYDROLOGICAL PROCESSES. PROCEEDINGS OF THE VII INTERNATIONAL CONFERENCE, MIT, USA, JUNE 1988

GROUNDWATER DISCHARGE TESTS: SIMULATION AND ANALYSIS

D. CLARKE

20 Musgrave St. Crystal brook, S.A. 5523 Australia

ELSEVIER

Amsterdam — Oxford — New York — Tokyo 1988

ELSEVIER SCIENCE PUBLISHERS B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

Distributors for the United States and Canada:

ELSEVIER SCIENCE PUBLISHING COMPANY INC.
655, Avenue of the Americas
New York, NY 10010, U.S.A.

Library of Congress Cataloging-in-Publication Data

Clarke, Dennis.
Groundwater discharge test simulation and analysis : microcomputer programmes in turbo Pascal / D. Clarke.
p. cm. -- (Developments in water science ; 37)
Includes index.
ISBN 0-444-43037-7 : fl 180.00
1. Groundwater flow--Measurement--Computer programs. 2. Turbo Pascal (Computer program) I. Title. II. Series.
GB1197.7.C58 1988
551.4'9'0724--dc19 88-24574
CIP

ISBN 0-444-43037-7(Vol.37)
ISBN 0-444-41669-2(Series)

Printed in The Netherlands

Appreciation

I wish to thank Zack Sibenaler, Bob Read, and Don Armstrong for all their help over the years with hydrogeological questions, and Michael Cobb for his encouragement.
I should also thank the multitude of workers, investigators, and writers who's work formed the background for the programs in this book. This book is a very imperfect application of selected parts of an enormous amount of work that has gone before it.
To my young son, Ken, and daughter, Julia, I must apologise for chasing them away from their computer games so many times.
Especially, I thank my wife, Denece, for her patient help and encouragement in the writing of this book.

Introduction

1. AIMS OF THE BOOK

This book is written with the intention of providing tools for the practising hydrogeologist, in a similar vein to it's predecessor, Microcomputer Programs for Groundwater Studies (Clarke 1987). Emphasis is placed on utility rather than on theoretical rigor.

All of the programs given in this work were developed with the dual intention that they should be both as useful as possible, be clear and sufficiently self explanatory for others to use with a minimum of learning time.

A decision has been made to provide computer output that shows, as far as practicable, the steps taken in arriving at a solution, so that the user may be in a position to follow the 'reasoning' and judge the validity of a particular case. An alternative would have been to concentrate on producing pretty or impressive graphical and printed output. Perhaps some users may wish to modify the programs to do this, but I did not see that as being the most desirable course in a book designed to show how computers can be used to provide answers to field questions. The output from these programs tries to be tidy, but providing information is given much more weight than aesthetic values.

Why another book on very much the same theme as one written only a few years previously? The short answer is that the use and availability of computers and computer software is changing, as also is my exposure to new hardware and software. Whatever it's qualifications, the IBM PC has set a much needed standard for the providers of software; this standard had not yet had such a great effect when my first book was being written, so that work was aimed more broadly. Memory comes more cheaply with each year, so it can be used more liberally by programming languages that are more memory hungry than Basic is. The advances in computing hardware and software, and the advances in the authors experience, allow programs that are more powerful, easier to use, faster, and which cover more ground than those of the older work.

The main differences between this and the older book are:

1/ The programs of this book are written in Turbo Pascal rather than in Basic.

2/ This book covers more ground than the last (eg. simulation of drawdown following Neuman's Unconfined Well Function, a more versatile plotting program, and much more discharge test analysis.

3/ This book is less elementary than the last. It does not start with simple solutions to basic functions and work up from there, rather it explains fully functioning programs right from the first chapter. This does not mean that this book is for more experienced programmers than the last; while greater experience may be necessary if one is to fully understand the operation of the programs, less should be needed to use them.

4/ Concentrating on one group of computers (IBM PC and clones) has allowed the use of screen graphics.

2. COMPUTER METHODS USED FOR GROUNDWATER PROBLEMS

Geology in general, and hydrogeology especially, often involves applying numerical values to naturally occurring systems; eg. an age to a rock formation, or a transmissivity to an aquifer. While in physics quantities may be known to high degrees of accuracy, hydrogeology uses approximations and generalizations. It is often not possible, or even desirable, to be totally accurate. eg. The age of which part of the formation? - the transmissivity of which part of the aquifer, and in which direction? Answers will be required for these questions at times, but often an approximate answer for the whole unit is all that is needed.

The moment one applies a mathematical equation (ie. a model) to a groundwater problem, even if that equation is as simple a Darcy's law (Bouwer, 1978), some simplifying assumptions must be made. To list a few common assumptions;

1/ The porous medium is homogeneous.

2/ The piezometers used to monitor the system give values that are representative of a significant cross-section of the system rather than of just one point.

3/ Vertical flow within the aquifer is negligible.

4/ The aquifer is fully confined.

5/ The aquifer is of infinite extent.

In reality these assumptions are very often not justified.

If unjustified simplifying assumptions are applied to a complex real world situation in order to obtain information on that system, then it follows that the information so obtained will at best be approximate. Only in an ideal (and therefore nonexistent) groundwater system will our methods of mathematical analysis give completely accurate answers.

Hydrogeology is not, cannot be, an exact science. The skill of the competent and experienced hydrogeologist rests largely in his/her ability to make meaningful generalizations and approximations, and in knowing how far

these can be pushed before errors become so great as to invalidate any conclusions that he/she may make.

These arguments must be borne in mind in using the programs in this book. There seems to be a tendency among some people to take any numbers produced by a computer to be absolutely correct. There is a saying in computer science; "rubbish in - rubbish out". You cannot expect the output of your computer to be better than it's input. In many cases errors in data will be magnified, and results will be less accurate than input.

If all this sounds pessimistic and defeatist, then it is time for a note of optimism. Very often the bulk properties of an aquifer can be approximated by average figures in such a way as to produce a reasonable simulation of the behaviour of that aquifer under given conditions of recharge or discharge. I've seen discharge test results from many wells that indicated an aquifer that behaved very similarly to an ideal infinite, confined, homogeneous, isotropic aquifer, at least for the duration of the test.

In summary, I would like to make two suggestions.

1/ Don't expect five figure accuracy when using these programs to evaluate some aquifer parameter, often one figure, or even order of magnitude values may be both useful and the best that can be expected.

2/ Perhaps there are times when mathematical rigour should take second place to seat of the pants empiricism?

3. WHY TURBO PASCAL?

This relatively new language is a super-set of standard Pascal and has quickly become extremely popular. It is quite true that Basic is a much better known language, but I cannot imagine anyone who has taken the trouble to learn Turbo Pascal, ever by choice using Basic again. Fortran seems popular among those who were trained in the use of Fortran as the programming language for science applications, but does anyone ever choose to learn Fortran after becoming proficient in Pascal?

The ease of use of Turbo Pascal, it's exceptionally good editor, it's speed, and it's (Pascal's) structure, are probably it's main advantages. A program who's source code occupies around 30 kilobytes can be compiled directly into memory in about 20 seconds on even a relatively slow PC, or it can be stored on disk as a stand alone machine language program.

Turbo Pascal can be expected to be around three to four times as fast in run time as interpreted Basic in most groundwater applications, but it may be twenty or more time faster than Basic in such tasks as sorting.

The relative difference if your computer has a maths co-processor will be greater than this.

The programs were written using Turbo Pascal version 3.0.

4. WHAT YOU NEED

These programs were written on an IBM PC XT compatible microcomputer with a colour graphics adaptor (CGA) and a maths co-processor. They should run on any computer of the IBM PC type, so long as there is at least 250k available to the programs. Some of the programs use monochrome graphics, and some use colour graphics. Program PLOTWTD uses a plotter with either the DXY or HPGL set of commands. So if you want to use all of the programs to their full potential you will need a colour monitor, a colour graphics adaptor, and a plotter such as the Roland DXY 880, or one of the Hewlett-Packard range.

Useful but not essential are a hard (fixed) disk, a maths co-processor, and 640k of RAM. Failing a hard disk, at least two floppy disk drives are probably essential.

5. GETTING STARTED

The programs may be typed in from the book if you do not have them on disk (see below). If there are only one or two programs that you are interested in, then this may be the best course. If you decide to type the programs in from the book, or if you want to be able to modify them in any way, you will require the Turbo Pascal editor/compiler, which at the time of writing was available for around $60 US in Australia. If you have the programs on disk, and you do not wish to alter them, you do not need Turbo Pascal.

If you do have the disks then you should copy the programs onto another working disk, and then put the distribution disks away in a safe place. If your computer has a hard disk, then I recommend making a new directory (see your Dos manual), and copying all the programs from the distribution disks into that directory. Perhaps two new directories would be better, with the programs in one and your discharge test data files in the other (see the notes on GW.BAT in the next section of this book, Preliminaries).

If you have a two floppy system, or if you want to get straight into using the programs, then simply place your working copy of the disk containing the executable code in the default disk drive, type GW2, and press enter. This will get you to a preliminary message, and on pressing any key you will be shown a menu. You may choose from the menu by pressing the

indicated key. You will notice that in all cases in these programs, when a choice may be indicated by one key, it is not necessary to press Enter after pressing that key.

From this point onward, you will have to consult the instructions for the individual programs.

6. THE PROGRAMS ON DISK

To avoid the tedious job of typing the programs in from the listings, you may purchase them on two thirteen centimetre (5 1/4 inch) double sided floppy disks for Australian$70 (approx. US$52) from the author at Clarke Computer Services, 20 Musgrave St., Crystal Brook, S.A., Australia, 5523. (20% sales tax is payable by buyers inside Australia.) The disks will be dispatched by air mail when the buyer is outside of Australia. There is no extra charge for postage and packing.

The disks are in a format compatible with an IBM PC. One disk contains the source code, and the other contains the executable object code. If you ask for object code suitable for an 8087 maths co-processor when you order, then that will be supplied, otherwise it will be assumed that your computer does not have this chip. (Code compiled to not use an 8087 chip will run when one is present, but the opposite is not true.)

The programs as printed in this book will not be entirely free of 'bugs'. I anticipate improving, upgrading, and adding to the programs in the future. If you find problems in your use of the programs, or you see room for improvement or expansion, please contact me. If you are reporting a 'bug' please take care to give very full and specific information about how the problem arose in writing, or better, on disk with the data involved.

7. LAYOUT OF THE CHAPTERS

Each chapter describes one or more programs. Each program description has three parts;

1/ The program from the users point of view. Non programmers should be able to follow this description.

2/ A description by procedure and function. This section describes the way the programs work in words. It is this section that gives the equations and algorithms on which the programs are based.

3/ A list of the key lines of the program. This, at a glance, shows the locations of the beginnings of all procedures and functions, and

any other major feature, within the program.

4/ The program listing itself. The programs are listed with each line numbered. The line numbers must not be typed in to your computer, they are for reference only. You will notice that some program lines in the listings are too long to go on one printed line. When typed in, these program lines should be placed entirely on one screen line.

8. UNITS

It would be very nice to be able to assume that every user would be happy to use units based on the metre and the day, but this cannot reasonably be assumed. This is especially so as discharge test times are usually dealt with in minutes.

Units are suggested by the programs, and are assumed in some places. Whether users wish to be bound by the suggestions is entirely up to them, but if they do not then they must consider the consequences of their decision.

The programs assume metres as the unit of length in all cases. The time unit assumed depends on the application, and will normally be specified. Entry of discharge test data assumes that times will be in minutes and drawdowns in metres, but if the user wants to enter times in days and drawdowns in feet there is no problem (at least until analysis is attempted). The programs always assume that the times stored on disk file are in minutes, but for any analytical operation times are converted to days so that a consistent unit set is achieved. (This causes no problems in DTDHA when data is picked up from disk file, an analytical operation done, and the data re-saved. It will go back to the file in minutes if it came off in minutes; unless a specific instruction was given to change the form of the data.)

I suggest that users stay with the assumed units as far as possible when analysis is involved. The choice of unit for data entry is of little consequence, data units are very easy to change at will using program DTDHA. Cosmetic modification to the graphing program, PLOTWTD, will be required if you want your graphs to say that drawdowns are in feet.

9. REFERENCES

Bouwer, H., (1978). Groundwater Hydrology, 480pp. McGraw-Hill Kogakusha.

Clarke, D.K., 1987. Microcomputer Programs for Groundwater Studies. Developments in Water Science, 30. Elsevier, Amsterdam/Oxford/New York/Tokyo, 340pp.

Preliminary

This section describes some small programs which serve to call up the main programs described in the body of this book. It also details some files which are used by the main programs in reading and writing discharge test data files. Readers who are not interested in the operational aspects of the programming might wish to pass over this section and go directly to Chapter One, although some notice should be taken of the notes on the batch file GW.BAT.

1. MOVING AROUND THE PROGRAMS

Several of the programs in this book are quite large, and all deal with basically the same subject, so it is convenient to have some controlled and easy method of moving around from one program to another, or from one part of a program to another part.

When a Turbo Pascal program is compiled to produce a compiled (COM) file, it always includes a library of functions, whether or not these may be required in the particular program; this can lead to waste of disk or memory space when several programs are involved. However, it is possible to compile only one program into the COM form, and all related programs into chain (CHN) form (see Appendix B); the chain files do not require their own library, but use that of the COM file. If this method is used, then the programs must be linked together in use, the program in the COM file being called first, and the chain files being accessed from there.

Turbo Pascal does not give a simple way of producing a directory of disk files, but this can be provided from a short batch file which may be called from Turbo.

The preliminary programs described in this section can provide the structure from which the main programs are accessed. Note that the preliminary programs are not essential, any one of the groundwater programs in this book can be compiled as a COM file and then used on it's own; but going from one program to another is easier, and the integration of the group is improved, if these are used.

2. FILE GW.BAT

This is a batch file which gives the program group the ability of obtaining directories of data files. To use the file you have two alternative courses, you may use the file as it is and tailor your

directories to suit, or you can tailor the file to suit your directories.

1/ If the file is to be used as it is then your discharge test data files must be on drive C in a subdirectory named PT (for pump test) which is itself in a subdirectory named DAT (for data).

2/ If you wish to modify the file, then you have full control over the placement of your data files.

If you wish to use the batch file, GW.BAT, as it is, then you can create the needed subdirectories by typing the following commands from the root directory. Type in the commands between the quotes. Type "md dat", press Enter, type "cd dat", press Enter, type "md pt", press Enter.

If you choose to modify the batch file then the simplest modification is to change only those lines which refer to the path to your data files (lines 3, 11, and 15). For example, if you want to have your discharge test data in a subdirectory having the path \dtest (a first level subdirectory), then you would change all occurrences of "\dat\pt" to "\dtest". Another example, perhaps you are using only two floppy drives, and you want your discharge test data files to be on drive B; then you could change "c:\dat\pt" in both lines 11 and 15 to "b:" (and line 3 could be removed as it now becomes redundant).

Beginners to the use of DOS commands will have to refer to their DOS manuals to learn how to use the editor 'Edlin' to alter a Batch file.

2.1. Function of GW.BAT

The Append statement used in line 3 tells your computer to always check in the stated directory when any data file is called for. This can save you having to type in the full path and file name whenever you want a data file read. Note that the append statement will not cause a data file to be written to the stated directory; it will go to the default directory unless you specifically state the path name. For example, when any one of the programs asks for a file name under which to store data, if you give the name "c:\dat\pt\test" then the file will go into a file named "test" (with the extension wtd if you have asked for an ASCII file, or ftd if you have asked for a 'fast' file), on drive C:, in subdirectory pt of subdirectory dat (supposing those subdirectories exist).

Program GWMENU is not able to call GW.BAT unless GW.BAT originally called program GWSTART. ie. This batch file is not useable unless you start the programs by writing GW (or GW2) and pressing Enter. If you ask for a directory when at the primary menu by pressing d, then you will be asked to specify Wtd or Ftd (Wtd is the extension name of an ASCII Well Test Data

file, while Ftd is that of a Fast well Test Data file). Depending on your decision, the Turbo Pascal 'Halt' command will be given the parameter 63 or 64. The Halt command will cause program execution to be terminated, and control will pass back to this batch file because it called the Turbo Pascal program group. The parameter then becomes a batch file error level, and controls the way in which the directory is called for.

Note that the command line parameter 1 will be passed to GWSTART.COM when it is first called, but 2 will be passed on any call after a directory. It is the value of this parameter that lets GWSTART know:

1/ if it was not called from GW.BAT (parameter neither 1 nor 2),

2/ whether it is being called before any directory has been provided (parameter equals 1),

3/ or whether it is being called after a directory (parameter equals 2).

I suggest that the user should freely change this file to suit his own needs, and to suit the configuration of his PC.

A second, simpler, version of GW.BAT, named GW2.BAT, has been included on the program disk. This does not contain an Append statement, and will read files from the default directory.

2.2. File listing, GW.BAT

```
 1 echo off
 2 cls
 3 append \dat\pt
 4 gwstart 1
 5 :begin
 6 if errorlevel 65 goto end
 7 if errorlevel 64 goto ftd
 8 if errorlevel 63 goto wtd
 9 goto end
10 :wtd
11 dir c:\dat\pt\*.wtd/w
12 pause
13 goto ret
14 :ftd
15 dir c:\dat\pt\*.ftd/w
16 pause
17 :ret
18 gwstart 2
19 goto begin
20 :end
```

3. PROGRAM GWSTART

This program, when compiled, serves as the essential COM file to which all the chain (CHN) files can refer for access to the functions of the Turbo Pascal run time library. As a minor function, it causes an introductory

message to be displayed when the G/W programs are first called up. On finishing, this program passes control to the primary menu program for the G/W group, GWMENU.CHN.

If you compile this program, then use the Turbo Pascal COM option, and be sure to set the Code Segment and Data Segment values to those given in line 2 of the program. Please refer to your Turbo Pascal manual for an explanation.

As with all the programs of this book, GWSTART contains the Include file FIRST.SEG. This file is a collection of those variables, procedures and functions that are common to most of the programs. It is described and listed later in this section.

There is little else that need be said of GWSTART, except a few words on the use of the ChainTo procedure. This is described in the notes on the Include File FIRST.SEG which contains it. At this point it is only necessary to say that on ending, this program attempts to call up the chain file GWMENU.CHN, and if it cannot be found then the present program, GWSTART.COM, will abort with an appropriate message.

3.1. Program listing, GWSTART.PAS

```
 1 Program GWSTART_PAS; {A COM file for well test data handlers etc.
 2 Code segment minimum is 0A40, data segment minimum is 0680 paragraphs.
 3 Turbo Pascal apparently cannot take more code in a chain file.}
 4
 5 {$I First.seg}
 6
 7 var
 8   Ch: char;
 9
10 Procedure Introduction;
11 var Ch: Char;
12 begin
13   writeln('             The GW set of programs');
14   writeln;
15   writeln('   These programs are subject to copyright by Clarke Computer
   Services,');
16   writeln('20 Musgrave St., Crystal Brook, 5523, South Australia.');
17   writeln('   If you have difficulties in using these programs and
   cannot solve them');
18   writeln('by reference to the instructions please contact the above
   address.');
19   writeln; writeln('        Disclaimer');
20   writeln('   While the programs are supplied in the belief that they
   work as described,');
21   writeln('Clarke Computer Services make no guarantee at all, and take
   no');
22   writeln('responsibility for any damages whatever, which may arise out
   of the use');
23   writeln('or misuse of these programs.'); writeln;
24   writeln;
25   writeln('Press any key to continue.');
```

```
26   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
27 end; {Procedure Introduction}
28
29 begin {Main part of Program GWSTART}
30   ClrScr; TextColor(Green); IOCode:=1;
31   if ParamStr(1)<>'2' then Introduction;
32   if (ParamStr(1)<>'1') and (ParamStr(1)<>'2') then
33   begin
34     writeln('ERROR: This program has been called direct.  Directory ',
35       'will not be available.');
36     delay(2000);
37   end;
38   ChainTo('GWMENU.CHN',IOCode);
39   if IOCode<>0
40   then begin
41     writeln('Unable to execute program GWMENU.CHN!');
42   end;
43 end.
```

4. PROGRAM GWMENU

As is the case with GWSTART, this is a small and simple program. Neither of these is essential to the running of any of the main programs of this book, but as explained above they do make the set of programs more integrated.

The program consists of two main parts, the first of which (procedure DisplayMenu) displays a menu giving the names of all the major programs, and the name of the key which should be pressed to get to each. The second part calls the chain file appropriate to the key pressed.

Two alternatives other than an exit to one of the G/W program are available, these are to press E to end the program, or to press D to have a directory of discharge test data files displayed. The first of these simply results in termination of this program, while the second is a little more involved. On pressing D, you will be asked: "Which type of data file, Wtd, or Ftd?" (line 63). Here you indicate your choice by pressing either W (to indicate a file with the extension WTD, an ASCII file), or F (to indicate a file with the extension FTD, a machine language, or 'fast' file). (See Appendix A for more information of data file formats.) Function CapOptions (explained under the notes on file FIRST.SEG), then returns a 1 if W has been pressed, or a 2 if it was F that was pressed. The value returned is added to 62 in line 65 and the result is used as the parameter passed to DOS (the disk operating system) by the Halt command. It is this number that batch file GW.BAT then uses to decide the class of discharge test data files that are to have their names displayed.

4.1. Program listing, GWMENU.PAS

```
 1 Program GWMENU_PAS;
 2
 3 {$I First.seg}
 4
 5 var
 6   Finished: boolean;
 7   Ch: char;
 8   TempInt: integer;
 9   ProgFileName: ShortString;
10 const
11   ValidResponse:
12     set of char=['1','2','A','J','N','P','S','D','E'];
13
14 Procedure DisplayMenu;
15 begin
16   writeln('         GW primary menu');
17   writeln;
18   writeln('  Which program do you want to run?');
19   writeln('Press the indicated number or letter key.');
20   writeln; writeln;
21   writeln('1: DTDHA     Discharge Test Data Handling and Analysis;');
22   writeln('2: DRAWDOWN  Calculate drawdowns in various aquifers;');
23   writeln('A: ANALYZE   Analyze aquifer test data;');
24   writeln('J: JOINWTD   Join a leaky aq. simulation to an unconfined
   aq.'
25     ,' simulation;');
26   writeln('N: NEUMAN    Simulation by Neuman''s unconfined well
   function;');
27   writeln('P: PLOTWTD   Plot well test data on the VDU screen;');
28   writeln('S: SIM7      Simulation of drawdown from a pumped well;');
29   writeln;
30   writeln('D:           To view the directory of data files;');
31   writeln('E:           To end GW and return to Dos.');
32 end; {Procedure DisplayMenu}
33
34 begin
35   ClrScr; Finished:=false;
36   repeat
37     DisplayMenu;
38     repeat
39       Ch:='x'; repeat read(kbd,Ch) until Ch<>'x'; Ch:=UpCase(Ch);
40     until Ch in ValidResponse;
41     if (Ch<>'E') and (Ch<>'D')
42     then begin
43       case Ch of
44         '1': ProgFileName:='DTDHA.CHN';
45         '2': ProgFileName:='DRAWDOWN.CHN';
46         'A': ProgFileName:='ANALYZE.CHN';
47         'J': ProgFileName:='JOINWTD.CHN';
48         'N': ProgFileName:='NEUMAN.CHN';
49         'P': ProgFIleName:='PLOTWTD.CHN';
50         'S': ProgFileName:='SIM7.CHN';
51       end; {of cases}
52       ChainTo(ProgFileName,IOCode);
53       Ch:='x';
54       if IOCode<>0
55       then begin
```

```
56          writeln('Sorry, this option is not available.');
57          writeln('Please select another.');
58          Delay(2000); ClrScr;
59        end;
60      end; {if Ch<>'E' etc.}
61      if Ch='D'
62      then begin
63        writeln('Which type of data file, ');
64        TempInt:=CapOptions('Wtd, or Ftd? ');
65        halt(TempInt+62);
66      end; {if Ch='D'}
67      if Ch='E' then Finished:=true;
68    until Finished
69  end.
```

5. INCLUDE FILE FIRST.SEG

This file contains the global variables that are common to all the Pascal programs of this book, and the functions and procedures that are of particularly general use.

An important variable type is MainVec, which is the vector (single dimensional array) type that is used for the very basic purpose of holding time, drawdown, or discharge rate data. Pointer variables of this type are used in those programs that have need to hold more than one set of discharge test data at one time.

5.1. Procedures and functions of file FIRST.SEG

CapOptions function Line 28

Purpose: to take a string containing several options which are indicated by words beginning with capital letters, to allow the user to indicate which option is required, and to return the ordinal number of the chosen option.

The string passed to the function is first displayed, with all capitals white, and all other characters green. At the same time, a copy is made of all the capital letters, and stored in the string, Valid. The keyboard is then monitored until a key is pressed who's capital form is contained in Valid. Finally, the ordinal position of the indicated capital is given to CapOptions, to be returned to the calling routine.

Response function Line 60

Purpose: to take a list of valid responses, allow the user to choose one, and return the choice.

The list of choices must be a short string of capital letters, and is placed in the variable, Targ. The keyboard is monitored until a key corresponding to one member of the list is pressed, then the capitalised form of the selected letter is displayed and returned.

ReadReal function Line 74

Purpose: to allow the user to input a real number, and to reject any invalid entries.

Unlike basic, Turbo Pascal by itself does not object, and ask for a repeat, if a user enters something invalid when a numerical entry is expected. This, and functions ReadInt and ReadIntInput, have been written to make up for this lack.

First a record is taken of the position of the cursor before the user enters anything. The user's entry is picked up in the global string, Short, and an attempt is made to convert this to a real in the same line of code. If an error is found in the conversion, then Result is given a non zero value, and the message 'Invalid' is displayed for one second at the place on the screen where the 'number' was typed. After the seconds delay, the message is removed, and the cursor is returned to it's original position.

If no error occurs in the conversion, then the result is returned to the calling routine. Note that a nul entry (the pressing of the Enter key before any character is typed) is also considered to be invalid, although it is acceptable to the built in Turbo Pascal Val routine. Finally the number of lines that were passed to the routine in variable, Num, are printed on the screen - in preparation for the next display.

ReadInt function Line 95

This function is almost identical to ReadReal except that it accepts an integer, where that function accepts a real; please refer to the notes on that function for an explanation.

ReadIntInput function Line 116

Again, very similar to the function ReadReal, please refer to those notes. Here an integer is read in, and a nul input is acceptable. If a nul input is entered, then zero will be returned to the calling routine.

ChainTo procedure Line 138

Purpose: to chain to a file who's name is given.

All of the Pascal programs given in this book can terminate and pass execution to another program file having the extension name of 'CHN'. This procedure brings about that operation. If the chain file is not found, then integer variable, IOCode, (input-output code) will receive a non zero value and control will pass back to the calling routine.

5.2. Include file FIRST.SEG, key lines

```
  4 {#----- Include file FIRST.SEG -----#}
 28 Function CapOptions {Choice of a highlighted option.
 60 Function Response {Exit only on receiving an acceptable key entry.
 74 Function ReadReal {Reads a real number from the user, rejects errors.
 95 Function ReadInt {Reads an integer from the user, rejects errors.
138 Procedure ChainTo {Chain to another file, return code if not found}
116 Function ReadIntInput {Reads an integer or nul from the user.
149 {#----- End of include file FIRST.SEG -----#}
```

5.3. Include file FIRST.SEG, listing

```
  1
  2 (*{$I FIRST.SEG}*)
  3
  4 {#----- Include file FIRST.SEG -----#}
  5 type
  6   MainVec=array[1..500] of real;
  7   LongString=string[80];
  8   MedString=String[40];
  9   ShortString=string[20];
 10   Test=(Discharge, Recovery, Simulation, TOverT1);
 11   Well=(Pumped, Observation);
 12   TypeOfData=(FTD, WTD);
 13   Rec=record
 14     OneTime, OneDd, OneRate: real;
 15   end;
 16
 17 var
 18   Error: boolean;
 19   I, J, First, IOCode: integer;
 20   TempVal, Num: real;
 21   Exten, Short, Answer: ShortString;
 22   FileName: MedString;
 23   Long: LongString;
 24   ThisRec: Rec;
 25   DataType: TypeOfData;
 26   FtdFile: file of Rec; WtdFile: text; MainFile: file of byte;
 27
 28 Function CapOptions {Choice of a highlighted option.
 29   Prints a string, beginning at the current cursor position, highlights
 30 all capital letters, and returns the ordinal number of the chosen option}
 31 (LongStr: LongString):integer;
 32 type CapSet= set of char;
 33 const AllCaps: CapSet=['A'..'Z'];
 34 var
 35   I: byte; Valid: String[10]; Ch, Cap: char;
 36 begin
 37   Valid:=''; Cap:=' ';
 38   For I:=1 to Length(LongStr) do
 39   begin
 40     Ch:=copy(LongStr,I,1);
 41     if Ch in AllCaps
 42     then
 43       begin
 44         TextColor(White); write(Copy(LongStr,I,1));
 45         Valid:=Valid+Copy(LongStr,I,1);
 46       end
 47     else
 48       begin
```

```
49          TextColor(Green); write(Copy(LongStr,I,1));
50        end {else}
51    end; {for}
52    while pos(Cap,Valid)=0 do
53    begin
54      read(kbd,Cap); Cap:=UpCase(Cap);
55    end;
56    write(' ',Cap); writeln;
57    CapOptions:=Pos(Cap,Valid);
58  end; {Function CapOptions}
59
60  Function Response {Exit only on receiving an acceptable key entry.
61    The ordinal number of the key's position in the list is returned}
62  (Targ: ShortString): ShortString;
63  var
64    Temp: byte;
65    Ch: Char;
66  begin
67    repeat
68      read(Kbd,Ch);
69      Temp:=Pos(UpCase(Ch), Targ)
70    until Temp>0;
71    writeln(Ch); Response:=UpCase(Ch);
72  end; {Function Response}
73
74  Function ReadReal {Reads a real number from the user, rejects errors.
75    The number of line feeds to be output is passed to this function in
     Num}
76  (Num: real):real;
77  var
78    I, J, X, Y: byte;
79    Result: integer;
80  begin
81    X:=WhereX; Y:=WhereY; Result:=1; I:=round(Num);
82    while (Result<>0) or (length(Short)=0) do
83    begin
84      read(Short); Val(Short,Num,Result);
85      if (Result<>0) or (length(Short)=0) then
86      begin
87        GotoXY(X,Y); write('Invalid'); Delay(1000);
88        GotoXY(X,Y); write('       '); GotoXY(X,Y);
89      end;
90    end;
91    ReadReal:=Num;
92    for J:=1 to I do writeln;
93  end; {Function ReadReal}
94
95  Function ReadInt {Reads an integer from the user, rejects errors.
96    The number of line feeds produced is passed to this function in I}
97  (I: Integer): Integer;
98  var
99    J, X, Y: byte;
100   Num, Result: integer;
101 begin
102   X:=WhereX; Y:=WhereY; Result:=1; Short:='';
103   while (Result<>0) or (length(Short)=0) do
104   begin
105     read(Short); Val(Short,Num,Result);
106     if (Result<>0) or (length(Short)=0) then
```

```
107     begin
108       GotoXY(X,Y); write('Invalid'); Delay(1000);
109       GotoXY(X,Y); write('       '); GotoXY(X,Y);
110     end;
111   end;
112   ReadInt:=Num;
113   for J:=1 to I do writeln; 114 end; {Function ReadInt}
115
116 Function ReadIntInput {Reads an integer or nul from the user.
117   The number of line feeds produced is passed to this function in I
118   This funct. accepts a nul entry}
119 (I: Integer): Integer;
120 var
121   J, X, Y: byte;
122   Num, Result: integer;
123 begin
124   X:=WhereX; Y:=WhereY; Result:=1; Short:=' ';
125   while (Result<>0) and (Short<>'') do
126   begin
127     read(Short); Val(Short,Num,Result);
128     if (Result<>0) and (Short<>'') then
129     begin
130       GotoXY(X,Y); write('Invalid'); Delay(1000);
131       GotoXY(X,Y); write('       '); GotoXY(X,Y);
132     end;
133   end;
134   if Short='' then ReadIntInput:=0 else ReadIntInput:=Num;
135   for J:=1 to I do writeln;
136 end; {Function ReadIntInput}
137
138 Procedure ChainTo {Chain to another file, return code if not found}
139 (ProgFileName: ShortString; var IOCode: integer);
140 var
141   CFile: file of byte;
142 begin
143   Assign(CFile,ProgFileName);
144   {$I-}
145   Chain(CFile);
146   {$I+}
147   IOCode:=IOResult;
148 end; {Procedure ChainTo}
149 {#----- End of include file FIRST.SEG -----#}
```

6. INCLUDE FILE SAVE.PRC

This file contains all the procedures and functions necessary to save a set of discharge test data to a disk file.

The user interacts with the procedures of this file in several places. He is informed of the last file name used (if a file name has been used previously). He is asked to specify the type of file that he wants the data stored in, either 'fast', or 'human readable'. (A 'fast' file is stored in machine language and is unreadable to anything other than another Turbo Pascal program designed to read that file. A 'human readable' file is written in ASCII and is readable by any ASCII editor, and it is printable, and displayable.)

Next the user is asked to enter the path name (if any), and the file name by which he wishes to save the current data. An example of a path name and file name is "\dat\pt\mytest". The first part, the "\dat\pt", tells the disk operating system (DOS) that the file is to be stored in a subdirectory called "pt" (short for pump test) which is itself in a subdirectory called

"dat" (short for data). The last part, "mytest", is the name that will be given to the file. The user must not specify a file name extension, that will be given by the program. It will depend upon and indicate the type of the file.

It is possible that a file may exist in the directory specified by the path name (or in the default directory if no path name is given). It is also possible that the path name may be invalid; it may name subdirectories that do not exist. Both of these possibilities are checked, and any errors or conflicts are reported to the user.

6.1. Procedures and functions of file SAVE.PRC

Exist function Line 5

Purpose: to test for the existence of a file with a given name.

Perhaps the best way to show how this function operates is with and example. Suppose that FileName = "TestFile". When function Exist is called, DataType will be set to either FTD or WTD, suppose that in this case it is set to WTD. Line 10 will Assign WtdFile to "TestFile.WTD", and then line 15 will attempt to reset a file by this name. If no such file exists, an Input/Output error will occur, and IOResult in line 18 will have a non zero value.

Called by subprocedure RewriteCheck of procedure SaveData; and as the same function is also used in files READ.PRC and READSAVE.PRC, it is again called by subprocedures ReadWTD and ReadFTD of procedure ReadTestDataFile.

SaveData procedure Line 21

Purpose: to record on disk file all data for a particular discharge test. SaveData also warns the user when an attempt is made to store data over that in an existing file, or when an invalid path name or disk identifier is used.

The procedure is called with all the variables carrying data to be saved as parameters. This is because some of the programs in this book use pointer variables, and others do not. By having parameters passed to this procedure, data held in either pointer variables, or ordinary variables can be saved to disk.

In line 100, if a file name has already been used then the user is reminded of what it was. This may be useful in a program such as DTDHA, when a file has been read, edited, and is now to be re-saved.

Lines 101 to 106 arrange the type of file, and the extension file name, before calling procedure RewriteCheck. This procedure obtains a file name from the user, tests and reports on the consequences of trying to save the data under that name. When RewriteCheck has passed the intended file name and opened the file, either SaveFast or HrSave is called to save the data from lines 108 or 109.

Called by: any program that has need to store discharge test data to disk file.

HrSave sub-procedure Line 26

Contained within procedure SaveData

Purpose: to write the discharge test data currently stored in RAM memory to disk as a sequential ASCII file.

RewriteCheck has already obtained a file name, and checked the validity of that name; all that remains is the writing of the data. Lines 28 to 31 store the test type code number, the well type code number, the distance (from pumped well to piezometer), and the number of time/drawdown/discharge rate records on the first line of the file. Lines 33 to 39 store the records themselves, and line 40 closes the file.

Called by: the parent procedure, SaveData

SaveFast sub-procedure Line 43
Contained within procedure SaveData
Purpose: to write the discharge test data currently stored in RAM memory to a random access disk file.

This procedure is very similar to HrSave in general. As a random access file can store only one type of data, this stores records which consist of three reals each. (Records of the type Rec, as defined in file FIRST.SEG.) The necessity of storing the test type code number, the well type code number, the distance, and the number of records is the reason for the peculiar programming of lines 46 to 56.
Called by: the parent procedure, SaveData

RewriteCheck sub-procedure Line 69
Contained within procedure SaveData
Purposes: 1/ to avoid the unintentional overwriting of an existing file.
2/ to check that any drive specification or path name given by the user is valid.

Line 76 calls function Exist to achieve the first objective. There are two segments of code to cover the second objective, one for each file type. If the drive specification or path name is illegal, then an attempt to 'rewrite' a file will cause an Input/Output error. This, in turn, will cause the value of Error to be set to true. Normal exit from the procedure is not possible until Error is false.
Called by: the parent procedure, SaveData
Calls: function Exist

6.2. Include file SAVE.PRC, key lines

```
 2 {# Include file SAVE.PRC; contains all the procedures and functions
 3  # required to save 'fast' and 'human readable' discharge test data
    files}
 5 Function Exist {Test for existence of a given file}
21 Procedure SaveData {Save calculated series of values to disk file}
26   procedure HrSave; {Human Readable Save}
43   procedure SaveFast; {File fast to save and read, but not human
    readable}
69   procedure RewriteCheck; {Check validity of path, and 'rewrite' file}
99 begin {# main part of procedure SaveData}
112 {# End of include file SAVE.PRC}
```

6.3. Include file SAVE.PRC, listing

```
 1
 2 {# Include file SAVE.PRC; contains all the procedures and functions
 3  # required to save 'fast' and 'human readable' discharge test data
    files}
 4
 5 Function Exist {Test for existence of a given file}
 6 (FileName: ShortString): boolean;
 7 begin
 8   case DataType of
 9     FTD: Assign(FtdFile,FileName+'.FTD');
10     WTD: Assign(WtdFile,FileName+'.WTD');
11   end; {of cases}
12   {$I-}
13   case DataType of
14     FTD: Reset(FtdFile);
15     WTD: Reset(WtdFile);
16   end; {of cases}
17   {$I+}
```

```
 18   Exist:=(IOResult=0);
 19 end; {Function exist}
 20
 21 Procedure SaveData {Save calculated series of values to disk file}
 22 (Time, Drawdown, Rate: MainVec; TestType: Test; WellType:
 23 Well; Distance: real; first, last: integer);
 24 var Error: boolean;
 25
 26   procedure HrSave; {Human Readable Save}
 27   begin
 28     Str(Ord(TestType)+1:14,Long);
 29     Str(Ord(WellType)+1:14,Short); Long:=Long+Short;
 30     Str(Distance:14:3,Short); Long:= Long+Short;
 31     Str(Last-First+1:14,Short); Long:= Long+Short;
 32     writeln(WtdFile,Long);
 33     for I:=first to last do
 34     begin
 35       Str(Time[I]:14:3,Short); Long:= Short;
 36       Str(Drawdown[I]:14:3,Short); Long:= Long+Short;
 37       Str(Rate[I]:14:3,Short); Long:= Long+Short;
 38       writeln(WtdFile,Long);
 39     end;
 40     Close(WtdFile);
 41   end; {sub-procedure HrSave}
 42
 43   procedure SaveFast; {File fast to save and read, but not human
    readable}
 44   var Error: boolean;
 45   begin
 46     with ThisRec do
 47     begin
 48       OneTime:=ord(TestType);
 49       OneDd:=ord(WellType);
 50       OneRate:=Distance;
 51     end;
 52     Write(FtdFile,ThisRec);
 53     with ThisRec do
 54     begin
 55       OneTime:=Last-First+1; OneDd:=0; OneRate:=0;
 56     end;
 57     Write(FtdFile,ThisRec);
 58     for I:=first to last do
 59     begin
 60       with ThisRec do
 61       begin
 62         OneTime:=Time[I]; OneDd:=Drawdown[I]; OneRate:=Rate[I];
 63       end; {ThisRec}
 64       write(FtdFile,ThisRec);
 65     end;
 66     Close(FtdFile);
 67   end; {sub-procedure SaveFast}
 68
 69   procedure RewriteCheck; {Check validity of path, and 'rewrite' file}
 70   begin
 71     {$I-}
 72     repeat
 73       repeat
 74         write('What name to save the file by? '); readln(FileName);
 75         I:=1;
```

```
 76         if Exist(FileName)=true
 77         then begin
 78           write('A file by this name exists, ');
 79           Long:='Replace, or Change name?';
 80           I:= CapOptions(Long);
 81         end; {if Exist}
 82       until I=1;
 83       case DataType of
 84         FTD: begin
 85           rewrite(FtdFile);
 86           Error:=(IOResult<>0);
 87           if Error then writeln('Error: check path or file name
    validity');
 88         end; {case FTD}
 89         WTD: begin
 90           rewrite(WtdFile);
 91           Error:=(IOResult<>0);
 92           if Error then writeln('Error: check path or file name
    validity');
 93         end; {case WTD}
 94       end; {of cases}
 95       {$I+}
 96     until not Error;
 97   end; {sub-procedure RewriteCheck}
 98
 99 begin {# main part of procedure SaveData}
100   if FileName<>'' then writeln('Last used file name was ',FileName);
101   write('Do you want a '); Long:='Fast save, or a Human readable save?';
102   I:=CapOptions(Long); DataType:=TypeOfData(I-1);
103   write('Chosen type is ');
104   if DataType=FTD then writeln('Fast') else writeln('Human readable');
105   case DataType of FTD: Exten:='.FTD'; WTD: Exten:=',WTD'; end;
106   RewriteCheck;      {Check validity of path, and 'rewrite' file}
107   case DataType of
108     FTD: SaveFast;
109     WTD: HrSave; 110   end; {of cases} 111 end; {Procedure SaveData}
112 {# End of include file SAVE.PRC}
```

7. INCLUDE FILE READ.PRC

This file includes all those procedures and functions that are required to read a discharge test data file of either 'human readable' (ASCII, with a file name extension of WTD) or 'fast' (machine language, with a file name extension of FTD) form. The user will be asked to specify the name of the file to be read. If the user does not specify the type of file, then first a 'fast' file will sought, if not found then a 'human readable' file having the same name will be looked for. If neither searches are successful, then he will be requested to enter another name. If the user specifically does not want a 'fast' file read, then he must add a '+' sign to the file name that he enters; this will result in the first and only search being for a 'human readable' file. If it is desired to exit the procedure, and not load a file at all, then a lower case 'x' must be entered instead of a file name.

On successful loading of a data file, the opportunity is given to view the file and carry out minor editing of the data. Note that editing the data in memory will not change the file on disk without a specific instruction to save the modified data back to disk.

7.1. Procedures and functions of file READ.PRC

AlterEntry procedure Line 140

Purpose: to allow the user to alter one record.

This procedure is called after time/drawdown/discharge rate data has been displayed on the screen. If the user notices one displayed value that is wrong he can use AlterEntry to change it as he wishes. Line 158 uses function CapOptions to allow the user to choose between altering time, drawdown, or discharge rate. Lines 159 to 177 display the current value before allowing the user to enter a new value.

Called by: procedure ViewReadings

Exist (This function has been described under the notes on file SAVE.PRC, so it will not be covered again here.)

NoSpaces function Line 8

Purpose: to remove space characters from the string Short, passed to it.

Called by: sub-procedure ReadHuman of procedure ReadTestDataFile

ReadFast sub-procedure Line 76

Purpose: to read in a set of discharge test data from a random access disk file.

The record used to write the file consisted of three real numbers. The record itself has the name ThisRec, and it is made up of the three reals, OneTime, OneDd, and OneRate. The first two records are used to store the test type code, the well type code, the distance from discharge well to piezometer, and the number of time/drawdown/discharge rate readings in the file. This part of the file is read in lines 80 to 88. The data of the readings themselves make up the remainder of the file, and are read in from line 89 to 102.

Called by: sub-procedure ReadFTD of procedure ReadTestDataFile.

ReadFTD sub-procedure Line 114

Purpose: to check for the existence of a random access ('fast') file having a given name, and if found, call ReadFast to read it.

If the file is not found, then Boolean variable FileThere is set to false. This value will be acted upon by procedure ReadTestDataFile.

Called by: the parent procedure ReadTestDataFile.

ReadHuman procedure Line 47

Purpose: to read in a set of discharge test data from a sequential access disk file.

This procedure has been written so as to be compatible with files produced by the Basic language programs of my previous book (Clarke 1987), as well as to give a clear, alterable, ASCII file. The first line of the file contains the test type code, the well type code, the distance from discharge well to piezometer, and the number of time/drawdown/discharge rate readings in the file. This line is read by the section of code from line 54 to 61. The data of the readings themselves are read in lines 63 to 73.

Called by: sub-procedure ReadWTD of procedure ReadTestDataFile.

ReadTestDataFile procedure Line 37

Purpose: to control the reading of a file of discharge test data, and to make the operation as easy as possible for the user.

The procedure is called with all the variables carrying data to be saved as parameters. This is because some of the programs in this book use pointer variables, and others do not. By having parameters passed to this procedure, data held in either pointer variables, or ordinary variables can be saved to disk.

Line 131 checks for the occurrence of a '+' in the file name entered by the user; if one is present, it indicates the users desire to read a sequential access file rather than a random access file. If the '+' is found, then Turbo Pascal gives Result a value indicating it's position in the string. If it is not found, then Result is given a zero value.

If the user has not included a '+' in his file name, then ReadFTD is called in an attempt to read a random access data file. If this is successful the procedure will be concluded. If either the '+' is found, or the random access file is not found, then ReadWTD will be called in an attempt to read a sequential access file. If this fails, then the user is called upon to enter another file name.

If the user cannot think of a file name that the program can find, he may escape by entering 'x', instead of a file name.

Called by: any program having a need to store a set of discharge test data, or discharge test simulation data, to disk file.

ReadWTD sub-procedure Line 105

Purpose: to check for the existence of a sequential access (ASCII, or 'human readable') file having a given name, and if found, call ReadHuman to read it.

If the file is not found, then Boolean variable FileThere is set to false, and control passes back to ReadTestDataFile.

Called by: the parent procedure ReadTestDataFile.

ViewAlterData procedure Line 210

Purpose: first, to allow the user to view the data of the current file, and second, to allow alteration of one reading.

This procedure by itself causes only the test description data to be displayed. If the user decides to view the time/drawdown/discharge rate readings then procedure ViewReadings is called. If alteration of a reading is required, then ViewReadings calls procedure AlterEntry.

Called by: any program that has need of a display of the current discharge test data.

Calls: ViewReadings

ViewReadings procedure Line 180

Purpose: to display the time/drawdown/discharge rate data of the current file, and to call procedure AlterEntry if the user indicates a wish to change any reading.

The user may enter the number of the reading at which he/she wants the data display to begin. If the value entered is invalid, being either less than one or greater than the number of readings on record, it is rejected (lines 192 and 193). If the user does not enter anything, only pressing the Enter key, then it is assumed that he wants to begin viewing the data from the first record.

Once the display of data has commenced it proceeds 20 lines at a time, and the user must respond at the end of each 'screen page'. At each response it is possible to continue the display, alter an entry, or discontinue the display and exit the procedure. (Lines 195 to 207.)

7.2. Include file READ.PRC, key lines

```
 4 {#----- Include file READ.PRC -----#
 5 #   This file contains all the procedures and functs. required to read
 6 # 'fast' and 'human readable' discharge test data files}
 8 Function NoSpaces {Removes spaces from a given 'Short' string}
21 Function Exist {Tests for the existence of a given file}
37 Procedure ReadTestDataFile {Read a file of discharge test data}
47   procedure ReadHuman; {Read a human readable file}
76   procedure ReadFast; {Read fast file, not human readable}
```

```
105   procedure readWTD; {Controls the reading of a WTD file}
114   procedure readFTD; {Controls the reading of a FTD file}
123 begin {# main part of ReadTestDataFile}
140 Procedure AlterEntry {Allow the alteration of one entry in the file}
180 Procedure ViewReadings {Display all readings in the data file}
210 Procedure ViewAlterData {Control the display and alteration of data}
244 {#----- End of Include file READ.PRC -----#}
```

7.3. Include file READ.PRC, listing

```
  1
  2 (*{$I READ.PRC}*)
  3
  4 {#----- Include file READ.PRC -----#
  5 #   This file contains all the procedures and functs. required to read
  6 # 'fast' and 'human readable' discharge test data files}
  7
  8 Function NoSpaces {Removes spaces from a given 'Short' string}
  9 (Short: ShortString): ShortString;
 10 var  I: byte;
 11 const  Space = ' ';
 12 begin
 13   I:=pos(Space,Short);
 14   while I>0 do
 15   begin
 16     delete(Short,I,1); I:=pos(Space,Short);
 17   end;
 18   NoSpaces:=Short;
 19 end; {Function NoSpaces}
 20
 21 Function Exist {Tests for the existence of a given file}
 22 (FileName: ShortString): boolean;
 23 begin
 24   case DataType of
 25     FTD: Assign(FtdFile,FileName+'.FTD');
 26     WTD: Assign(WtdFile,FileName+'.WTD');
 27   end; {of cases}
 28   {$I-}
 29   case DataType of
 30     FTD: Reset(FtdFile);
 31     WTD: Reset(WtdFile);
 32   end; {of cases}
 33   {$I+}
 34   Exist:=(IOResult=0);
 35 end; {Function exist}
 36
 37 Procedure ReadTestDataFile {Read a file of discharge test data}
 38 (var Time, Drawdown, Rate: MainVec; var TestType: Test; var WellType:
    Well;
 39 var Distance: real; var NumData: integer);
 40 var
 41   FileThere: boolean;
 42   TempInt: integer;
 43   Line: LongString;
 44   TempVal: real;
 45   Result: integer;
 46
 47   procedure ReadHuman; {Read a human readable file}
 48   const
 49     TimeStart=1; TimeLen=14;
```

```
  DdStart=15; DdLen=14;
  RateStart=29; RateLen=18;
begin
  writeln('Reading data file, ',FileName+Exten);
  Readln(WtdFile,Line);
  Short:=copy(Line,1,14); Val(NoSpaces(Short),TempVal,result);
  TestType:=Test(round(TempVal)-1);
  Short:=copy(Line,15,14); Val(NoSpaces(Short),TempVal,result);
  WellType:=Well(round(TempVal)-1);
  Short:=copy(Line,29,14); Val(NoSpaces(Short),Distance,result);
  Short:=copy(Line,43,14);
  Val(NoSpaces(Short),NumData,result);
  I:=0;
  for I:=1 to NumData do
  begin
    Readln(WtdFile,Line);
    Short:=copy(Line,TimeStart,TimeLen);
    Val(NoSpaces(Short),Time[I],result);
    Short:=copy(Line,DdStart,DdLen);
    Val(NoSpaces(Short),Drawdown[I],result);
    Short:=copy(Line,RateStart,RateLen);
    Val(NoSpaces(Short),Rate[I],result);
  end;
  Close(WtdFile);
end; {sub-procedure ReadHuman}

procedure ReadFast; {Read fast file, not human readable}
begin
  writeln('Reading file ',FileName+Exten);
  seek(FtdFile,0);
  read(FtdFile,ThisRec);
  with ThisRec do
  begin
    TestType:=Test(round(OneTime));
    WellType:=Well(round(OneDd));
    Distance:=OneRate;
  end;
  seek(FtdFile,1);
  read(FtdFile,ThisRec);
  with ThisRec do
    NumData:=Ord(round(OneTime));
  for I:=2 to NumData+1 do
  begin
    seek(FtdFile,I);
    read(FtdFile,ThisRec);
    with ThisRec do
    begin
      Time[I-1]:=OneTime;
      Drawdown[I-1]:=OneDd;
      Rate[I-1]:=OneRate;
    end; {ThisRec}
  end;
  Close(FtdFile);
end; {sub-procedure ReadFast}

procedure readWTD; {Controls the reading of a WTD file}
begin
  DataType:=WTD; Exten:='.WTD';
  writeln('Attempting to open data file, ',FileName+Exten);
```

```
109     FileThere:=Exist(FileName);
110     if FileThere then ReadHuman
111     else writeln('  This file does not exist.');
112   end; {sub-procedure readWTD}
113
114   procedure readFTD; {Controls the reading of a FTD file}
115   begin
116     DataType:=FTD; Exten:='.FTD';
117     writeln('Attempting to open data file, ',FileName+Exten);
118     FileThere:=Exist(FileName);
119     if FileThere then ReadFast
120     else writeln('  This file does not exist.');
121   end; {sub-procedure readFTD}
122
123 begin {# main part of ReadTestDataFile}
124   repeat
125       writeln('Enter the name of the data file (without an extension).');
126     writeln('Enter a file name of "x" to exit.');
127     write('Suffix with a + if you want a .WTD file. (eg. test+) ');
128     readln(FileName);
129     if FileName<>'x' then
130     begin
131       Error:=false; Result:=Pos('+',FileName);
132       if Result<>0 then Delete(FileName,Result,1);
133       if Result=0 then ReadFTD; {User wants FTD file}
134       if (Result<>0) or (not FileThere) then ReadWTD;
135     end;
136   until FileThere or (FileName='x');
137   if FileName='x' then NumData:=0;
138 end; {Procedure ReadTestDataFile}
139
140 Procedure AlterEntry {Allow the alteration of one entry in the file}
141 (var Time, Drawdown, Rate: MainVec; NumData: integer);
142 var
143   RecNum, Answer: integer;
144 begin
145   writeln('          Alter one reading (record)'); writeln;
146   writeln('   Please enter the number of the reading you wish to alter,
    ',
    'press Enter');
147   write('without typing a number to exit: ');
148   RecNum:=ReadIntInput(1);
149   if RecNum<>0 then
150   begin
151     if (RecNum<0) or (RecNum>NumData)
152     then begin
153       writeln('Invalid reading number!'); delay(2000)
154     end {then}
155     else begin
156       write('Do you want to alter ');
157       Answer:=CapOptions('Time, Drawdown, or discharge Rate?');
158       Case Answer of
159         1: begin
160            writeln('Current time for this reading is '
161              ,Time[RecNum]:8:1,'min.');
162            write('Enter new time (min.) '); Time[RecNum]:=ReadReal(2);
163         end;
164         2: begin
165            writeln('Current drawdown for this reading is '
166
```

```
167              ,Drawdown[RecNum]:8:3);
168            write('Enter new drawdown '); Drawdown[RecNum]:=ReadReal(2);
169          end;
170          3: begin
171            writeln('Current discharge rate for this reading is '
172              ,Rate[RecNum]:8:2);
173            write('Enter new rate '); Rate[RecNum]:=ReadReal(2);
174          end; {case 3}
175        end; {all cases}
176      end; {else}
177    end; {if RecNum}
178  end; {Procedure AlterEntry}
179
180  Procedure ViewReadings {Display all readings in the data file}
181  (var Time, Drawdown, Rate: MainVec; NumData: integer);
182  var
183    Result: integer;
184  const
185    Heading='  No.       min.        days      Drawdown      Rate';
186  begin
187    writeln('Which reading number to start listing (press Enter alone if
     you');
188    write('wish to start with the first reading) ? ');
189    repeat
190      readln(Short);
191      if length(Short)=0 then First:=1 else Val(Short,First,Result);
192      if (First<1) or (First>NumData) then
193        writeln('Invalid!  Please re-enter.');
194    until (First>=1) and (First<=NumData);
195    writeln('All times below are in minutes');
196    I:=First; J:=0;
197    repeat
198      writeln(Heading);
199      repeat
200        write(I:4,' ',Time[I]:11:3,'  ',Time[I]/1440:9:4,'
     ',Drawdown[I]:9:4);
201        writeln('  ',Rate[I]:9:2);
202        I:=I+1; J:=J+1;
203      until (J=20) or (I>NumData);
204      Result:=CapOptions('Continue, Alter, or Exit? ');
205      if Result=2 then AlterEntry(Time, Drawdown, Rate, NumData);
206      J:=0;
207    until (I>NumData) or (Result=3);
208  end; {Procedure ViewReadings}
209
210  Procedure ViewAlterData {Control the display and alteration of data}
211  (var Time, Drawdown, Rate: MainVec; TestType: Test; WellType: Well;
212  Distance: real; NumData: integer);
213  var
214    Ch: Char;
215    I, J, First: integer;
216  begin
217    ClrScr;
218    write('Test type is ');
219    Case TestType of
220      Discharge:
221        writeln('Discharge');
222      Recovery:
223        writeln('Recovery');
```

```
224     Simulation:
225       writeln('Simulation');
226     TOverT1:
227       writeln('t/t1 recovery');
228   end; {of cases}
229   write('Well type is ');
230   case WellType of
231     Pumped:
232       Writeln('Pumped');
233     Observation:
234       writeln('Observation');
235   end; {of cases}
236   writeln('Distance = ',Distance:10:2,'m');
237   writeln('The number of data is ',NumData);
238   writeln;
239      write('Do  you want to see the time/drawdown/discharge rate data? ');
240   Short:='YN';
241   if Response(Short)='Y' then
242   ViewReadings(Time, Drawdown, Rate, NumData);
243 end; {Procedure ViewAlterData}
244 {#----- End of Include file READ.PRC -----#}
```

8. INCLUDE FILE READSAVE.PRC

This file will not be listed because it consists simply of file SAVE.PRC appended onto file READ.PRC. (As both files contain the function FileExist, one copy of this function must be removed from the resulting file.)

9. REFERENCE

Clarke, D.K., 1987. Microcomputer Programs for Groundwater Studies. Developments in Water Science, 30. Elsevier, Amsterdam/Oxford/New York/Tokyo, 340pp.

This Page Intentionally Left Blank

Chapter 1

The program described in this chapter has been named DTDHA for the initial letters of the words Discharge Test Data Handling and Analysis. It is similar to, but more advanced than, a previous program by the same name published in Clarke, 1987.

If you have the programs on disk, then pleas note that DTDHA is compiled as a chain file, and it cannot be run directly while in this form. To run program DTDHA first follow the instructions in the introduction to commence the execution of the controlling program, GW.COM. From the primary menu which will be displayed, press the key indicated to activate program DTDHA.

1. THE AIMS OF PROGRAM DTDHA

Computers have great and largely untapped potential in handling and analysing data from water well discharge tests; whether such tests are for the purpose of assessing a well or an aquifer. Discharge test data can be stored on floppy disk and accessed as required for analysis, printing, or display. Careful naming of the disk files, and of the disks themselves will enable easy access of a very large number of discharge tests.

(Those readers who have need of a library of several thousand discharge tests will find a data base program allowing efficient indexing, to be well worth while. Commercial programs are available for this purpose.)

At this time the only practical way of getting hand written data into a computer is by typing it in via a keyboard (although optical character scanners are becoming cheaper and better), therefore the first purpose of DTDHA is to allow this job to be done as efficiently as possible.

Having once loaded the data into a computer there is no limit to the number of ways it can be manipulated for various purposes. For example it might be desirable to convert all recorded water level recovery times into the t/t_1 form, or convert water levels referred to ground level so that they refer to standing water level. The computer can be used to produce graphs of the data, of a type to suit the aquifer type, and the users preference; DTDHA will record the data on a disk file so that program PLOTWTD can graph it. Of course we would be ridiculously under utilising a computer if we stopped at entering and manipulating the data; in the computer we have a tool that can be used to great advantage for analysis of discharge test data. The analytical jobs done by DTDHA comprise three different methods of solving the coefficients of the well equation. Analysis to do with aquifers

is handled by program ANALYZE (see Chapter 7) using data from a disk file produced by program DTDHA.

If discharge test analysis is possible by microcomputer, then so to is simulation. Program DRAWDOWN produces files of simulated discharge test data that may be displayed, printed, edited, or manipulated in other ways using DTDHA.

Rather than deal with all the applications of DTDHA in summary here, they will be dealt with one at a time, in detail below.

2. FUNCTIONS OF THE PROGRAM: (MENU ONE)

2.1. Entry of discharge test data via the keyboard

Assuming that you have field data that you wish to analyze or file, the first thing that must be done is to type the manually collected data into the computer. This task is the most trying and uninteresting of all. The more fortunate readers may have office staff who can do this job for them, others will have to take the time to do it themselves. The data entry routines of DTDHA are designed primarily to minimise the key punching required to achieve the desired result. Secondary aims are to allow quick and easy correction of errors, and to help users keep track of exactly where they are in the process.

There are two methods of data entry available, both of which are called up via the first option of the main menu of DTDHA. Most frequently used will be the option of entry of times as elapsed minutes measured from the beginning of the discharge test. In long tests, or in long recoveries, it is sometimes more convenient to record the date and time of day, rather than the number of minutes since the beginning of the test. Manually converting this to elapsed minutes can become an irksome and repetitive task, so provision is made for entry of times in this form.

On selection of the data entry option at the main menu you will have to answer some questions about the source of the data before entry of times, drawdowns, and discharge rates begins. The first question will be whether the test is a discharge test, recovery test, or a simulation. You reply to this question by pressing one key, the initial letter of the chosen word. Your answer will be recorded when the data is saved to disk file, and will be retrieved whenever the file is read, but it will not have any effect on any operations which may later be done on the data. The second question will be whether the well in which the measurements were taken was a pumped well, or an observation well. Again, the answer will be recorded, but it will not

take any important part in later operations. The third and last question of this type will be either:

1/ the distance between the pumped well and the observation well, if you indicated that the readings concern an observation well, or

2/ the effective radius of the well, if a pumped well is involved. Please note that the answer to this question is very important in case one above, as the value will be used if you later ask for the storage coefficient to be calculated from the entered data (or have some curve fitting process done by program ANALYZE). As storage coefficient cannot be reliably calculated for the data recorded from the pumped well, the figure you enter as the effective radius is less important.

After answering the above questions, you will be required to indicate whether you want to enter times as minutes (ie. the time for each drawdown reading as the interval, in minutes, from commencement of discharge, to the moment of the drawdown determination), or as a calendar date and time of day.

If there are already data in memory, newly entered data will go onto the end of existing data. If there are no data in memory, newly entered data will begin with record number one.

RECORD The data are stored and handled in terms of records, each of which is sequentually numbered. A record consists of a group of data including one time, one drawdown, and one discharge rate, where the drawdown and discharge rate were true at the specified time. This concept is central to understanding the operations of many of the programs of this book. (In this book the term Reading will be used almost synonymously with Record when the latter is applied to discharge test data.)

2.1.1. The form of data required

In the simplest case of a discharge test consisting of a single pumping rate, data plainly will be in the form of a time, measured from the beginning of discharge; a drawdown, measured at that time; and a discharge rate, also measured, or assumed, at that time.

In the case of a multistage discharge test, the form that the data must take is not so obvious. For most purposes, times are to be measured from the beginning of the first discharge phase. The exception is when there are a series of discharge phases with full recovery (not measured, or at least, not recorded) between the phases. In this latter case, the times to be entered into the computer should relate to the commencement of the particular stages. If the data is in this form of separate stages, then the user must

realize that it cannot be treated as one integrated whole; it must rather be treated as a consecutive series of separate discharge tests.

CHANGES IN DISCHARGE RATE An example. You pumped for thirty minutes at 100 cubic metres per day, and then increased the discharge rate to 200 cubic metres per day. You took a drawdown reading a moment before the rate was increased, and then another a minute and 12 seconds after the increase. The relevant part of your data should have a form something like... Time 30, Drawdown 3.23, Rate 100; Time 31.2, Drawdown 4.85, Rate 200; etc. Looking at this data it could be supposed that the discharge rate increased at any time between 30 and 31.2 minutes. The programs of this book assume that the rate increase was immediately after the last reading at the old rate; ie. the increase occurred around 30.000001 minutes.

2.1.2. Entry of data with times in minutes

If you are unsure of how to get to this point, first follow the preliminary steps given in the introduction (Setting up the Disks and Files), and at the beginning of this chapter (Executing DTDHA).

The program will ask for you to specify the time of the current record (or reading). Note that the times -1, and -2 minutes are used to indicate something other than time to the program (see below), so they cannot be used as times. If you should happen to want to enter a time of say -1 minutes, (perhaps for an observation well reading one minute before the starting of the pump) you will have to use a number close to, but not equal to -1, instead. For example you could enter -1.0000001. Real numbers in Turbo Pascal have approximately eleven decimal digits of precision, so there is little likelihood of the program mistaking your entry for -1. (The 8087 version of Turbo has around 16 decimal digits of precision.)

Enter the time of the reading in minutes, and press Enter to indicate that you are ready. You will next be called upon to enter the drawdown at the given time, and then the discharge rate.

As the discharge rate is normally constant for a number of readings, it would involve unnecessary typing to enter it for every time/drawdown pair. Therefore, after the first record, if you do not indicate that you want to enter a new discharge rate by typing a '+' sign at the end of the drawdown, it will be assumed that the discharge rate remains unchanged.

DATA ENTRY ERROR If you enter a wrong value by mistake, and you want to correct it, you can enter a time of -1 to indicate that you want to 'back up' to the last record, and re-enter it. If you enter a time wrongly, and realize that fact when you are about to enter the drawdown; go ahead and

enter some figure for the drawdown, and then enter -1 for the following time. Any value that is in error and is not corrected here may be corrected later from the data editing menu, menu 2.

ENDING DATA ENTRY To tell the program that you do not want to enter any more readings, enter a time of -2.

2.1.3. Entry of data with times as date and day

Sometimes it may be more convenient to enter the time of a reading by reference to the time and calendar date of that reading. If a strip aquifer, or an aquifer bounded on all sides is suspected, then a recovery period of a week or so may be desirable. A few special purpose discharge tests may continue for up to a year. In cases such as these, the computer can be given the job of converting the date and time of day to the time in elapsed minutes from the beginning of discharge.

If you use this option it will be necessary for you to enter the date and time of the beginning of the test before entering any reading times.

Entry of a particular point in time will begin with specification of the minute, followed by the hour, day, month, and year.

ENTRY OF MINUTE As in the above option, enter -1 to back up, or -2 to end the data entry session. The only other valid entries here are the integers from 0 to 59 inclusive. If you want to enter fractions of a minute, you will have use the other data entry method.

ENTRY OF HOUR Here the only valid entries are the integers from 0 to 23 inclusive, am and pm are not used.

ENTRY OF DAY Valid entries are the date of any day which occurres in the current month. The validity of the day will be checked after entry of the month and year. eg. 29th of February, 1989 is not valid as that is not a leap year. Valid entries, then are the integers from 1 to 28, 29, 30 or 31 depending on the month, and in the case of February, also depending upon the year.

After entry of the date of commencement of the test, the day must be suffixed (or prefixed) with a + to go on to entry of the month, this saves some key punching because in most cases the next reading will be in the same month, making it unnecessary to re-enter the same month.

ENTRY OF MONTH Valid entries are integers from 1 to 12 inclusive. After entry of the date of commencement of the test, the month must be suffixed (or prefixed) with a + to go on to entry of the year. If this is not done, then the year of the present reading will be assumed to be the same as that of the previous reading.

ENTRY OF YEAR Here any integer from 0 to 3000 is valid. In almost all cases the short form of the year may be used; eg. 89 instead of 1989, as under the Gregorian calendar the year 0089 would not be a leap year just as the year 1989 will not be a leap year, etc. There will be a minor exception if 100 is used instead of 2000, as the year 100 was not a leap year (or would not have been, had the Gregorian calendar been in use then), but the year 2000 will be.

After you have entered data, or altered data and returned to the main menu, a message will be displayed above the menu reminding you to save the new data to disk before leaving the program. If you were to forget to save the data, and attempt to return to the primary menu (the GW menu), you would be given a reminder that your data have not been stored on disk. Please note that leaving any program causes the data in memory to be lost, unless you specifically ask that it be saved to a disk file.

2.2. Edit the data in memory

This option takes you to menu number two; see below. If you attempt to take this course before there are any data in memory, the program will refuse and give you an error message. After having gone to menu two, the program will assume that you have altered the data, and will display a message above the main menu reminding you to save the altered data before exiting the program.

2.3. Solve the well equation

This option takes you to menu three; see below. As above, you may not use this until there are data in memory.

2.4. Read a file from disk

You may use this to load a discharge test data file previously produced by this or some other program. When asked by any of these programs for a file name do not suffix it with an extension name. This program expects data files to have an extension of either '.FTD' or '.WTD', and it will search for them in that order. Therefore if files having the same name but different extensions are present, and you want the program to read the one with the '.WTD' extension and not the one with the '.FTD' extension, you will have to specifically indicate your desire by typing a '+' sign on the end of the file name. (The program will inform you of this at the time.)

Drive names and directory names (path names) are acceptable as a part of a file name, but if no path name is given, the program will search the

default drive and directory. If a file with the given name and an extension of either '.FTD' or '.WTD' cannot be found in the drive and directory that you have specified, then you will be informed and asked to enter another file name.

If your data files are all on a special directory, consider using the DOS command 'Assign' to tell the operating system to always check that directory, as an alternative to typing in the full path and file name every time you want to load a file. (See also the notes on batch file GW.BAT in the Preliminary section of this book.) Bear in mind, however, that if DOS is not given the full path and file name when told to save the file, it will go onto the default directory. These programs have little control over directory usage, except when specific names are given by the person using the programs.

After the data file has been successfully found and read, you will have the opportunity of viewing it. If you do choose to view it, then you can alter some of the data at that time by specifying the number of the record that you wish to alter.

DATA FILE NAME EXTENSIONS Disk files containing discharge test data produced by these programs will have file name extensions of either 'FTD' or 'WTD'. The former is a contraction of 'Fast Test Data', and the latter, of 'Well Test Data'. For more details see Appendix A.

2.5. Save the data to a disk file

Having entered or edited your data, this option can be taken to save it to disk. The program will not save the data to disk automatically at any time, it will only do so if you specifically tell it to using this option.

You will be asked to specify whether you want the data saved in a 'Fast' file, or in a 'Human readable' file. See the notes in Appendix A on file formats.

VALID DATA FILE NAMES You will be asked for a file name without an extension, you may also use a valid path name. Examples of some valid file names for use with any of these programs are: 'test', 'A:test2', '3rdtest', and assuming there is a first level subdirectory called 'data' on your disk then '\data\pumptest' would be a valid file name. Names like 'test.dat', and 'pumptest.' are not valid.

If you choose to write a 'Human readable' file, then your data file will be given the extension file name 'WTD', and if you use the 'Fast' file format, then the extension name will be 'FTD'. The former is compatible with the programs of the earlier book (Clarke 1987), the latter is not.

2.6. View the data in memory

The data of the current file will be displayed on the video display unit twenty records at a time. While times are normally stored in the computers memory in this program in minutes, they will be displayed in both minutes and days. You will have the choice of viewing all, or a selected part of the data; and if you notice a value which is wrong, you will be able to change it.

At the end of each screen 'page', you will be asked whether you want to continue, exit, or alter something. If you choose to continue then the next sequence of records will be displayed, if you choose to exit, you will be returned to the main menu. The third option, alteration of data, will allow you to change one reading. You will be asked to enter the number of the reading that you want to alter. If you press Enter without first typing a number at this stage you will be able to get back to viewing the rest of the data without changing anything. If you type a valid reading number you will be asked whether you want to enter a new time, drawdown, or discharge rate. Having pressed T, D, or R, you will then be given the current value for that item, and be asked to enter a new value.

2.7. Print the data in memory

Similar to the above option, but the output is sent to the printer rather than to the screen. If your printer writes 66 lines to the page, and it is started at the top of a page the output should be correctly paginated. Unlike viewing the data however, here the whole of the data will be printed, there is no option for printing a part of the data.

3. FUNCTIONS OF THE PROGRAM: (MENU TWO)

Menu two is concerned with alterations to the data. Details of the options available from menu two are given below. You will not be able to reach menu two until there are data in memory.

3.1. Alter an individual entry

This uses the same procedure that can be used to alter an individual entry from the data viewing option (see above). You will simply be asked which reading you want to alter, and then allowed to alter that reading.

3.2. Add a constant to entries

This can be used for jobs such as converting water levels measured from

some reference point, to true drawdowns referred to the SWL (standing water level).

CHANGING REFERENCE POINTS For example, the original SWL was 10.23m below the TOC. (top of casing), and during your discharge test, all water levels were measured from TOC. These measurements were subsequently entered into DTDHA as 'drawdowns'. You can use this option to change all water level readings to true drawdowns by adding -10.23 to all the 'drawdowns' in the file.

CHANGING REFERENCE TIMES Suppose that you entered recovery times as times measured from stopping the pump and you want to change them to times measured from commencement of discharge. If duration of discharge was 1440 min., then just add 1440 to all readings relating to recovery.

This option, and the next, allow you to do the field work the simplest way possible, and then make any necessary adjustments to the data after you have typed it into your computer.

3.3. Multiply entries by a constant

This can be used for changing units, correcting for readings taken in angle holes, etc.

CONVERSION OF FEET TO METRES If, for example, you measured your drawdowns in decimal feet and you want to convert them to metres, multiply them all by 0.3048.

ANGLE HOLE READINGS If your drawdowns were measured in a 60 degree angle hole, and you want to convert them to metres head equivalent, multiply them all by 0.86603 (Sine 60 degrees). (You may also have to alter the reference point using the option above.)

Several conversion factors which you may require are displayed at the time you use this function.

3.4. Delete a reading

The reading number is required for this operation, ie. you must tell the computer that you want to delete reading number x, rather than the reading at time x, or the reading having a drawdown of x.

If a reading is deleted there is obviously going to be a gap left in the data. In practice, a reading is not really deleted, rather it is overwritten with a copy of the data of the succeeding reading, which is itself then overwritten with a copy of the next, and so on. Finally, the variable recording the total number of readings is decremented by one. This process, then, involves the renumbering of all readings after the one that is deleted.

For example, suppose that the latter part of your recorded times is as in the first pair of columns below, and you want to delete the reading for 134 minutes because it is a typographical error. The second pair of columns show your data after deleting reading number 42. Of course the drawdown and discharge rate recorded against the deleted time would also be deleted.

record #	time
40	1220
41	1280
42	134
43	1340
44	1400
45	1440

record #	time
40	1220
41	1280
42	1340
43	1400
44	1440

DELETING SEVERAL READINGS To delete more than one reading, (other than consecutive readings) start by deleting the one with the highest number and work down to the one with the lowest number. If you start with low numbers and work up, the job will be more complicated because of the renumbering mentioned above.

You may find it simpler to use an ASCII editor to alter a WTD data file. If you chose this method, then take care to change the last number on the top line, which records the number of readings in the file (ie. the total number of lines in the file, minus one).

3.5. Delete a number of readings

Use this if you want to delete a number of consecutive readings. eg. from a file containing both discharge and recovery data, you might want to delete all the recovery data, save the discharge data under a new file name, reload the original (combined) data, delete the discharge data, and save the recovery data under a new name. At the end of this operation you would have three files, the first containing all the data, the second containing the discharge data only, and the last containing the recovery data alone.

Take care to keep track of what you are doing in deleting data. It is easy to make mistakes and delete some part that you did not intend to. I strongly recommend having a backup set of your data files on a separate disk in case of emergencies.

3.6. Simulate full recovery between discharge stages

To reduce time, and hence costs, in running a multistage discharge

test, it is a common practice to change from one discharge rate to the next with no recovery period between. On the other hand, methods of evaluating the coefficients of the well equation usually call for full recovery between steps. It is possible to graphically convert the data to the form it may be expected to take had there been full recovery before each step. (See the procedures and functions section for a description of this algorithm.) The program segment called here automates this imperfect graphical procedure.

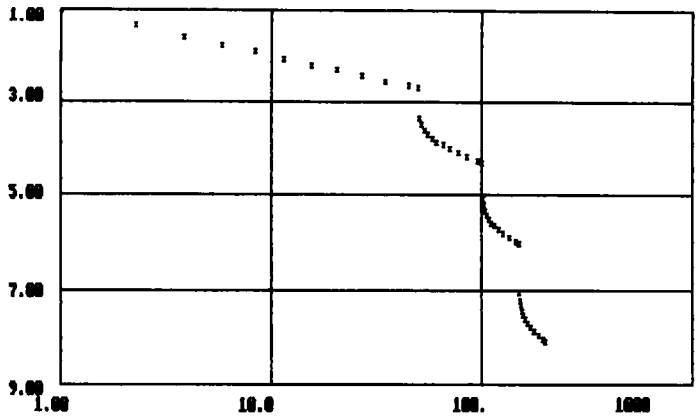
Data converted by this method will have times converted, as well as the drawdowns of all stages after the first. While the times associated with the input data must all relate to the beginning of the first discharge stage, the output times will all relate to the beginning of the simulated fully recovered single stages.

As an example, the graphs of Figures 1.1 and 1.2 are from simulated discharge test data. (The data were produced by program SIM7, the graphs were printed copies of screen graphs produced by program PLOTWTD, the conversion to simulated full recovery between readings was done by program DTDHA.) This discharge test consisted of four fifty minute stages at 200, 280, 360, and 450 cubic metres per day. The data that produced the first graph had reading times, 1, 2.3, 3.8, ... 34.9, 45, 50, 51, 52.3, ... 84.9, 95, 100, 101, 102.3, ... 134.9, 145, 150, 151, 152.3, ... and so on. These were converted to 1, 2.3, 3.8 ... 34.9, 45, 50, for the first stage, followed by 1, 2.3, ... 34.9, 45, 50, for the second, and similarly for the third and last stages. This latter is the form of data required for both the Rorabaugh analysis, or the s/Q vs. Q analysis for evaluation of the coefficients of the well equation.

It is important to realize that this feature gives only approximate results. Even when theoretically correct data is used as a starting point, as that used in Figures 1.1 and 1.2, this option will not bring it back to exactly what it would have been if there was full recovery between steps. The necessity of using this imperfect operation before calculating the coefficients of the well equation by Rorabaugh's method, or by the s/Q vs. Q method is perhaps the greatest disadvantage of those methods. It is also one of the strongest points in favour of using the Modified Sternberg Analysis.

The amount of error in conversion increases greatly with the later steps, and the latter parts of the latter steps. The second step will be converted with great accuracy, the third step with fair accuracy, but thereafter accuracy will diminish substantially. As an example of the degree of accuracy, examine table 1.1.

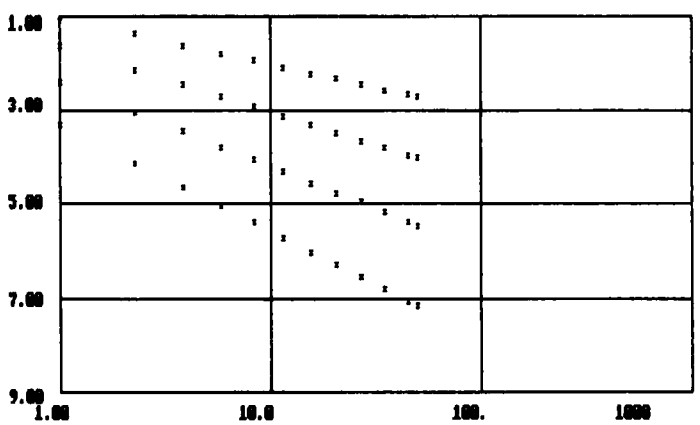
Figure 1.1



Graphed discharge well data from a simulated four stage discharge test. Drawdown increases linearly down the y scale, while time increases logarithmically along the x scale.

The simulation used the well equation with A=0.003 (time unit minutes), B=0.005, C=0.00001, n=2. There were four discharge stages, each of 50 minutes; the discharge rates were 200, 280, 360, and 450 cubic metres per day.

Figure 1.2



The simulated discharge test data of Figure 1.1 converted to simulate full recovery between stages. The scale of the graph is identical to that of Figure 1.1 to allow ease of comparison.

Table 1.1

Time	Step No.	Actual drawdown	Converted drawdown
1 min.	2	8.000m	8.000m
28.8	2	11.503	11.503
1	3	16.500	16.511
28.8	3	21.754	21.757
1	4	28.00	27.80
28.8	4	35.01	34.86
1	5	42.50	41.95
28.8	5	51.26	50.86

The 'actual drawdowns' of table 1.1 are calculated for single step simulations by program SIM7, while the 'converted drawdowns' are produced by converting the data from a five step discharge simulation to imitate full recovery between steps. This simulation was based on a well equation with A=0.01 (time unit minutes), B=0.012, C=0.00015, and n=2. Each discharge step was 0.02 days (28.8 minutes) long, and the discharge rates were 100, 200, 300, 400 and 500 cubic metres per day, in that order. (This is not the simulation used to produce figures 1.1 and 1.2.)

3.7. Convert time to t/t'

The later part of recovery data from a single rate discharge test in a simple, infinite, homogeneous and isotropic confined aquifer will fall on a straight line when plotted on a semilogarithmic graph as t/t' where t is the time of a reading measured from the beginning of discharge, and t' is the time of the same reading measured from the end of discharge.

The equation of Theis (1935) for residual drawdown reduces to

$$s' = \frac{2.3Q}{4\,Pi\,T} \log \frac{t}{t'} \qquad (1.1)$$

where s' is the residual drawdown,
Q is discharge rate,
T is transmissivity,
and t and t' are defined above.

To solve this equation for T, s' is plotted against t/t' on semilogarithmic paper. A straight line is fitted through the data points. The slope of this line is equal to 2.3 Q / 4 Pi T and also to the change in s' per unit log (base 10) cycle. (Termed the Delta s' slope.) Thus T can be calculated as

$$T = \frac{2.3\ Q}{4\ Pi\ Delta\ s'} \qquad (1.2)$$

or more simply,

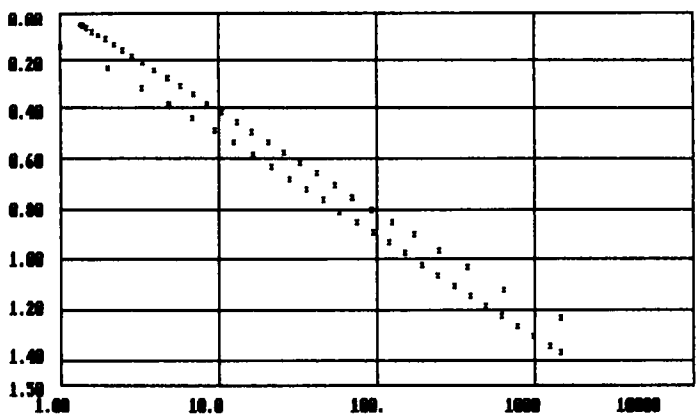
$$T = \frac{0.183\ Q}{Delta\ s'} \qquad (1.3)$$

Bouwer (1978) describes the use of these equations on pages 99 and 100.

This procedure automates the conversion of recorded times to t/t'. You do not need to tell the program when the recovery began, it will search for the first period of zero discharge, and convert the times for the whole of this period.

Use recovery data with time converted to t/t' to calculate transmissivity for your recovery period (using program ANALYZE), or to graphically confirm that the water level in your well is returning to the original level (using program PLOTWTD); but do not attempt solutions of the well equation from data in this form.

Figure 1.3



Simulated piezometer readings from a single stage one day discharge test in a simple confined aquifer, followed by a further three days of residual drawdown readings. The times of the residual drawdown readings have been converted to t/t' before plotting. The lower line is the drawdown phase, and the upper line is the recovery phase. Note that the recovery is in line with the origin.

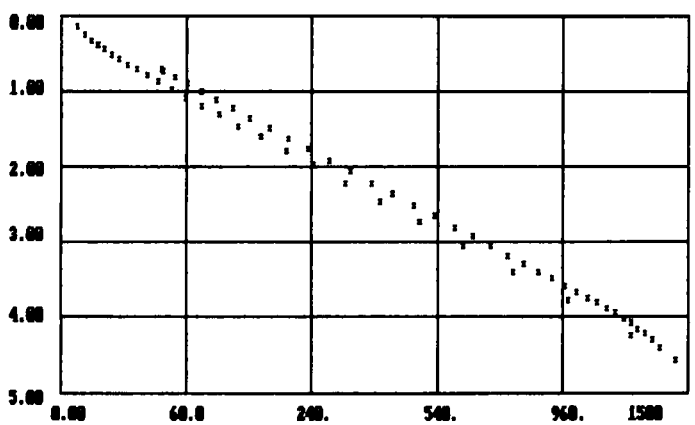
This simulation was produced by program DRAWDOWN for a piezometer at 35m from a discharging well, T=900, S=0.0007, and a discharge rate of 2000 cubic metres per day.

3.8. Convert time to (root t minus root t') squared

As recovery data with times converted to t/t' from a simple isotropic aquifer will fall on a straight line on a semilogarithmic graph, so recovery data from the later part of recovery in a simple strip aquifer will fall in a straight line when plotted on double linear paper with drawdown plotted against square root of t minus the square root of t'. (The later part of drawdown data from a single rate test in such an aquifer will fall on a straight line when plotted against the square root of time on double linear paper.)

A slightly different way of producing virtually the same graph is to construct a graph with linear drawdown on the y axis, and to mark the x axis off in proportion to the square root of time. Figure 1.4 uses this method. Note that the time reference lines of figure 1.4 are labelled 0, 60, 240, 540, 960, and 1500. The square roots of these numbers are approximately 0, 7.75, 15.49, 23.24, 30.98, and 38.72. The increment in each case is one fifth of the square root of 1500. The advantage of using a graph of this type, rather than using one marked with the square root of time along the x scale, is that times can be read directly from the graph. The plotting of root t minus root t' on such a graph requires that the values be first squared.

Figure 1.4



Simulated piezometer readings from a single stage one day discharge test in a confined strip aquifer, followed by a further nine days of residual drawdown readings. The times of the residual drawdown readings have been converted to (root t - root t')^2 before plotting. The lower line is the drawdown phase, and the upper line is the recovery phase. Note that the recovery is in line with the origin, as is to be expected from a strip of infinite length with no leakage from the sides.

The simulation of Figure 1.4 was produced by program DRAWDOWN for a piezometer at 35m from a discharging well, T=900, S=0.0007, and a discharge rate of 2000 cubic metres per day. The discharging well was 100m from the boundary, the piezometer 117m from the same boundary, and the strip was 325m wide.

The present option will convert times from the first period of recovery (zero discharge rate) to the above (root t minus root t') squared values in very much the same way as the previous option produced t/t' values.

3.9. Correct data for background 'noise'

BACKGROUND NOISE During an aquifer test of duration no more than around six hours, background noise in the form of rises or falls in the water level in observation wells can probably be safely ignored. In aquifer tests lasting much longer than this however, noise due to variations in atmospheric pressure etc. must be considered, especially if drawdowns in the observation wells are small.

The simplest way of allowing for background is to monitor a well in the same aquifer sufficiently far away from the discharging well to be unaffected by the pumping. Then the readings from this well are subtracted from the readings from the aquifer test observation well to remove the noise from the data. (Of course it is necessary that the background well is in the same aquifer as the discharge test observation well, and that it is unaffected by noise to which the test observation well is not subject; ie. irrigation pumping etc.)

If your background well has a different barometric susceptibility to the aquifer test observation well, you may use a conversion factor of less than, or greater than, 1. The background readings will all be multiplied by this factor before they are subtracted from the aquifer test readings.

USING A BAROMETER FOR BACKGROUND CORRECTION It is possible to use a barometer to measure background, but using barometric pressures to correct observation well readings is a little more complicated than using water levels from a background well. First, the barometric pressure readings must be entered into a data file as if they were drawdowns. Next they should all be multiplied by 0.0102 to convert them to metres head equivalent, then a constant will need to be subtracted to bring the pressures back to around zero. (Exactly what that constant is will depend on the circumstances at the time, I have used 10.25.) Now the two files, barometric pressure, and observation well readings should be graphed on the VDU using PLOTWTD, so that the amplitudes of the two curves can be compared. The factor that you use

for the correction operation could be decided from the results of this comparison.

For the description that follows it will be assumed that a disk file of readings from a background noise monitoring observation well is available.

1/ Load the data to be corrected into memory in DTDHA.

2/ Select 'Correct for background' from menu two.

3/ You will now be asked for the name of the file holding your background data; give it.

4/ Enter the correction factor when requested.

5/ The data in both files will be checked to make sure that all times are in increasing order. An error message will be displayed if one or other is found to be out of order. (If order is the only problem with your file, it can be corrected using the sort option, see below).

6/ All being well, the correction of the observation well data will now commence. The amount of correction to be applied will be calculated by linear interpolation of the background data for each recorded time in the observation well file.

7/ If the first reading in the observation well was taken at a time before the first reading in the background well, then this correction will be calculated by extrapolation from the first two readings in the background well. This situation is to be avoided. Start your readings in your background well before you start your discharge test, and then give these early readings negative times. (eg. ten minutes before the beginning of the test would give time = -10.)

8/ If the duration of the readings in the background well was less than those in the observation well then plainly it will not be possible to correct all the data. In this case those data that can be corrected will be, and those that cannot be will be deleted from the end of the file.

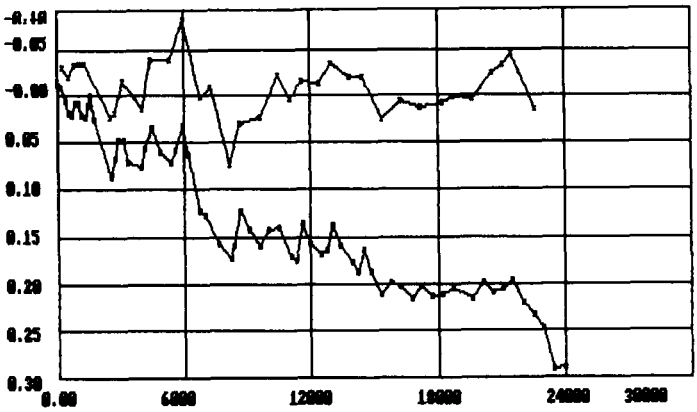
Figures 1.5 and 1.6 show some real field data before and after correction for background.

3.10. Change the test description data

This very simple facility allows you to change that part of the data file which records a little about the origin of the data.

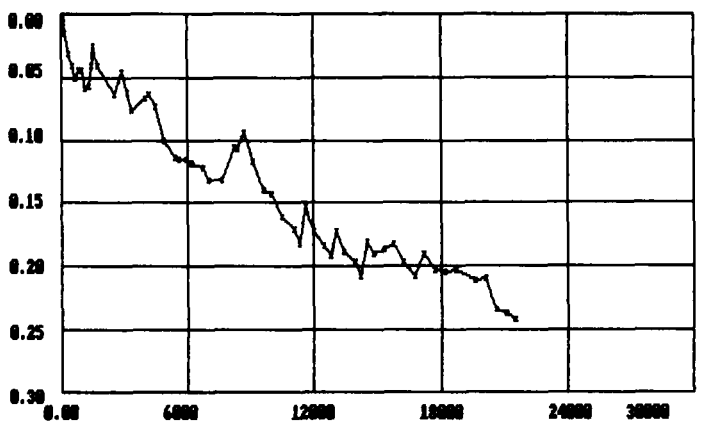
CHANGE R, THE DISTANCE You may use this to change the number which records the distance from the discharging well to the observation well or piezometer. This number is of great importance when analysing discharge test data for storage coefficient.

Figure 1.5



The upper curve is the drawdown in a background well. The lower curve is the data from an observation well presumed to be subject to the same 'noise'. Some similarity in the curves can be seen; and the need for sufficiently frequent, or near simultaneous, readings in both wells is obvious.

Figure 1.6



This is the above observation well data after correction for background noise. In this case it is plain that the observation well was subject to some noise in addition to that recorded in the background well, but there is some smoothing of the curve. (Note that the y scale on this graph is slightly different to that of Figure 1.5, and also that the data has been truncated because the background readings did not continue as long as those of the observation well.)

(In the case of drawdown readings taken in a pumped well, R is the effective radius, and has little importance, largely because it can usually be very poorly known.)

CHANGE THE WELL TYPE A code in the data file records whether the data came from a pumped well or a piezometer (observation well). This may be changed using this option.

CHANGE THE TEST TYPE A code in the file records the type of 'test' from which the data came, discharge test, recovery test, or simulation. This code can also be changed here.

3.11. Merge current data with another file

Two discharge test data files can be joined together to make one with this option. The data of the file that is currently in memory retains it's existing record numbers; and a new file, who's name must be specified, will be appended to the current data. It makes no difference if the current data were in a file with the 'FTD' extension, and the data to be appended are in a file with the 'WTD' extension because the actual data in a file are quite independent of the form in which they are stored.

The following is a hypothetical example of the use of this option. The data from the drawdown part of an aquifer test were entered into a file named Test1d, and the recovery data from the same test were stored in a file named Test1r. To combine both into one file, first load Test1d into memory, then merge Test1r. If the times in the recovery file were recorded as elapsed minutes since the end of discharge, you may now wish to change them to elapsed minutes since the beginning of discharge using the 'Add a constant' facility described above.

3.12. Sort into order of increasing time

If your data are out of order for some reason you may wish to use this option to return them into the correct order. You may, for example, have added several previously overlooked readings to the end of the file and now want to have them placed in their proper position.

Note that there are times when it is desirable to store data in an order other than that of increasing times. eg. If the times associated with recovery have been converted to t/t', or if your data have been converted to simulate full recovery between readings.

4. FUNCTIONS OF THE PROGRAM: (MENU THREE)

This last major menu is concerned with three methods for the solution of the well equation. Most readers will be familiar with the well equation, but a short description might be useful for those who are not.

THE WELL EQUATION The drawdown at any time after commencement of steady rate discharge from a fully penetrating well in a homogeneous, isotropic, infinite, confined aquifer can be approximated by use of the equation;

$$s=A\ Q + B\ Q\ \mathrm{Log}\ t + C\ Q^n \qquad (1.4)$$

where;

s is the drawdown in the discharging well,

t is time from commencement of discharge,

Q is the discharge rate,

A and B are coefficients dependent upon characteristics of the aquifer and well,

C is the coefficient quantifying the 'well loss',

and n has a value of between 1.5 and 3. (See Jacob, 1947; Rorabaugh, 1953; Lennox, 1966; and Bouwer, 1978.)

The exponent, n, above is often assumed to be equal to 2. Two of the procedures below, the s/Q vs. Q method, and the Sternberg method, make this assumption, while Rorabaugh's procedure evaluates the exponent. It is because Rorabaugh's procedure evaluates the exponent that it requires a minimum of three discharge stages, while the other methods can make do with two.

The so called well loss is apparently due to the relatively high velocity flow near the well being turbulent. (While flow within a porous aquifer is very rarely sufficiently fast to be turbulent, flow through the slotted casing, gravel pack, or well screen may well be.)

The coefficient B is directly proportional to the delta s slope of the time-drawdown data from a constant rate discharge test plotted with the drawdown on a linear scale, and the time on a logarithmic (base 10) scale. Consequently, B is inversely proportional to transmissivity. The remaining coefficient, A, is apparently associated with non turbulent aquifer losses.

4.1. Units in the well equation

The units used for time and discharge rate in the calculation of the well equation do not need to be consistent with each other. All that matters is that any time a particular well equation is used, it is used with the same units that were employed at the time of it's evaluation. As discharge test times are normally stored as minutes, and discharge rates (to be consistent with metres and days at the time of calculation of transmissivities and

storage coefficients etc.) are stored in cubic metres per day, these units are assumed here (and are assumed in program SIM7, which produces data from a given well equation).

The value of A will differ according to the time unit used for the drawdown readings, in the analyses presented in this program A will be given for both days and minutes.

This work will not provide a detailed explanation of the use and limits of the well equation.

4.2. Data requirements

There are a number of constraints on the form in which the data must be if they are to be acceptable for the various methods of solution of the well equation. The following notes will attempt to cover this, for more general information of the form of data used or produced by these programs, refer to Appendix A.

NUMBER OF STEPS A minimum of two discharge steps at different rates is required for solution by methods not evaluating the exponent, three are required if the exponent is to be evaluated (Rorabaugh's method). I would recommend a minimum of three discharge phases for Sternberg and for the s/Q vs. Q method, and four for Rorabaugh, to allow a best fit solution to be found. Field data will rarely produce a very good fit, and having an additional step or two will allow you to see how good the fit is between the field data and the given well equation. Use of the minimum allowable number of steps should lead to a rather misleading perfect fit.

Note that the Rorabaugh and s/Q vs. Q method will only count the steps up to the first recovery step (not including the recovery step). Any discharge steps after a period of zero discharge (except for initial conditions) will be ignored.

ORGANIZATION OF STEP DATA The Sternberg analysis requires the times in the data all to relate to the beginning of discharge. The other two methods expect to find the data as a series of consecutive steps with times in each step being measured from the beginning of that step. In other words, Sternberg uses data from a single discharge test having a number of different discharge rates; while the other methods expect data from a number of independent discharge tests each having a constant discharge rate, and being consecutively placed in one data file.

Sternberg also expects the initial conditions to be present as the first datum group, the other methods will work with initial conditions present or

absent. (The initial conditions are the drawdown and discharge rate at time zero.) The initial discharge rate must be zero, and the initial drawdown will normally be zero, but as Sternberg is capable of producing a well equation from pressure heads (in metres) instead of drawdowns, the value recorded as the initial 'drawdown' can actually be the initial head.

As an example, we have a test consisting of three thirty minute steps. If the data was to be used for Sternberg, the times would start with 0 minutes, and then probably 1, 2, ... 28, 30, for the first step, 31, 32, ... 58, 60, for the second step, and 61, 62, ... 86, 88, 90, for the last step. The discharge rate would have been increased at the end of each step, ie. at 30, and 60 minutes from the beginning of the first step. Sternberg does not expect full recovery between steps.

The same data in a form suitable for the other two methods would start with 0 minutes (optionally), and then 1, 2, ... 28, 30, for the first step, followed by 1, 2, ... 28, 30, for the second step, and 1, 2, ... 26, 28, 30, for the last step. The most important difference here is that these steps must be data from quite independent discharge steps, there having been full recovery between each test. (If your data came from a test with consecutive steps, with no recovery, then full recovery must be simulated using something like the facility described above, under menu two.)

Any of the three methods can work with data having a long last step, eg. data from a test which started with three short steps, and then went on to a long higher rate step would be quite acceptable.

Sternberg works best if a recovery step is included. In fact several recovery steps can be included anywhere in the discharge test, so a temporary pump failure causes no problems, so long as the person running the test goes on taking readings.

Sternberg can handle periods of continually changing discharge rate, while the other two methods expect distinct steps having a constant discharge rate within each. Especially when using the Sternberg procedure, care must be taken that the discharge rate is defined correctly at all times.

The times in the data used for the Sternberg analysis must be in increasing order, so if you add the drawdown and discharge rate at time zero (initial conditions) to the end of the data file, be sure to sort them into order (using the sort option of Menu 2) before running Sternberg. Obviously, since the data for the other tests must be in the form of several 'independent' discharge tests, then they must not be sorted. If one or other of the procedures rejects your data, have a look at it's form using the view or printout options, and consider the points raised in this section.

4.3. The modified Sternberg analysis

The modified Sternberg analysis, when applied to the data of the five step simulation of table 1.1 (Simulation of Full Recovery between Discharge Stages, above) and given a time of 25min., should produce coefficients for the well equation of A=0.0100, B=0.0120, C=0.000150; identical to those used to produce the simulation.

The algorithm can best be described in a series of steps.

1/ At the beginning, three tests for invalid data are carried out. First the data are checked for ordering of times, an error message is displayed and the procedure is aborted if it is found that each time is not greater than that which preceded it. Next, the program checks for initial conditions; the drawdown and discharge rate for time zero must be present, and the discharge rate at time zero must be zero.

2/ If the data passed the first checks for validity, the program moves on to the calculation of the vector StrnVec(x). The values in this vector are given by the equation;

$$StrnVec_i = \sum_{j=2}^{j=i} Delta\ Q_j\ Log_{10}(t_1 - t_{j-1}) \qquad (1.5)$$

This in turn involves several separate steps.

(1) The first element of the vector SternVec[x] is set to zero.

(2) Delta Q(i) is calculated for each time after the first, and stored in vector DeltaQ(x).

(3) Then SternVec(i) is the sum of all the terms; DeltaQ(J)*Log(Time(i)-Time(j-1)) for j from 2 to i. Of course, terms in which DeltaQ(j) equals zero have zero value; they are not calculated. The values of the elements in SternVec are displayed on the screen as they are calculated.

To make the operation of the program easier to understand, try running it on a file produced by program SIM7 as below.

Execute GWSTART.COM or GW.BAT (refer to the Introduction if you don't know how to do this). Call up program SIM7 from the GW menu. Specify a time unit of minutes. Enter a value of 0.004 for the well equation coefficient, A, 0.003 for B, 0.00005 for C, and 2 for the exponent n. When asked for discharge rates and the ending times of each step enter the following.

Step	Discharge rate	Time at end of step
1	100	50 (First step completed OK.)
2	0	60 (The pump stopped for ten min.)
3	200	100 (Forty minutes of the 'second' step.)
4	0	110 (That pump is no good!)
5	200	140 (Enough at 200m^3/day.)
6	300	190 ('Third' step went OK.)
7	0	1440 (Normal recovery.)

The simulation above produces a file which would be difficult to analyse using methods other than the Sternberg analysis because of the 'pump' stopping twice during the test. The second part of what we will suppose was intended to be the second step (200m^3/day) does not help the Sternberg analysis at all, but was included to illustrate a point below. This simulation should yield the results (from step 2/ above) given in table 1.2.

It is probably worth using this data to illustrate the way in which changes in discharge rate are handled (mentioned above). Note from the table that reading number 36 gives the discharge rate (Q) as zero at time 110 minutes, and reading number 37 gives the discharge rate as 200 at time 111 minutes. This means that the discharge rate increased from 0 to 200 immediately after time 110 minutes.

In the example data, if readings number 37 to 44 inclusive are deleted, correct results will still be obtained from the analysis. On the other hand, if readings number 39 to 46 are deleted, then it will appear that the discharge rate of 200 finished immediately after reading number 38, at 112.3 minutes, rather than the correct value of 140 minutes, and erroneous results will be calculated.

3/ After the calculation of the vector StrnVec, a small menu will be displayed. The menu gives the options of;

(1) Listing of the data in their present state to the printer. For checking, manual analysis, etc. (This was used to produce Table 1.2.)

(2) Display of the data on the screen; again, this would probably be required for checking.

(3) The third option is the most important; this will cause the remainder of the analysis to be performed.

Table 1.2

Sternberg output data

No.	Q	min.	days	Drawdown	Delta Q	Stern. vec.
1	0.0	0.0	0.0000	0.000	0.00	0.00
2	100.0	1.0	0.0007	0.900	100.00	0.00
3	100.0	2.3	0.0016	1.006	0.00	35.41
4	100.0	3.8	0.0027	1.076	0.00	58.51
5	100.0	5.8	0.0041	1.130	0.00	76.69
6	100.0	8.4	0.0058	1.177	0.00	92.26
7	100.0	11.5	0.0080	1.219	0.00	106.23
8	100.0	15.5	0.0108	1.257	0.00	119.15
9	100.0	20.6	0.0143	1.294	0.00	131.35
10	100.0	26.9	0.0187	1.329	0.00	143.03
11	100.0	34.9	0.0243	1.363	0.00	154.32
12	100.0	45.0	0.0313	1.396	0.00	165.33
13	100.0	50.0	0.0347	1.410	0.00	169.90
14	0.0	51.0	0.0354	0.512	-100.00	170.76
15	0.0	52.3	0.0363	0.409	0.00	136.41
16	0.0	53.8	0.0374	0.344	0.00	114.60
17	0.0	55.8	0.0388	0.294	0.00	98.01
18	0.0	58.4	0.0405	0.253	0.00	84.36
19	0.0	60.0	0.0417	0.233	0.00	77.82
20	200.0	61.0	0.0424	3.023	200.00	74.39
21	200.0	62.3	0.0432	3.224	0.00	141.39
22	200.0	63.8	0.0443	3.350	0.00	183.40
23	200.0	65.8	0.0457	3.446	0.00	215.25
24	200.0	68.4	0.0475	3.525	0.00	241.59
25	200.0	71.5	0.0497	3.594	0.00	264.58
26	200.0	75.5	0.0525	3.656	0.00	285.39
27	200.0	80.6	0.0560	3.714	0.00	304.77
28	200.0	86.9	0.0604	3.770	0.00	323.23
29	200.0	94.9	0.0659	3.823	0.00	341.13
30	200.0	100.0	0.0694	3.852	0.00	350.51
31	0.0	101.0	0.0701	1.057	-200.00	352.23
32	0.0	102.3	0.0710	0.851	0.00	283.52
33	0.0	103.8	0.0721	0.720	0.00	239.89
34	0.0	105.8	0.0735	0.620	0.00	206.64
35	0.0	108.4	0.0753	0.538	0.00	179.27
36	0.0	110.0	0.0764	0.498	0.00	166.12
37	200.0	111.0	0.0771	3.278	200.00	159.23
38	200.0	112.3	0.0780	3.467	0.00	222.36
39	200.0	113.8	0.0791	3.580	0.00	260.10
40	200.0	115.8	0.0804	3.662	0.00	287.33
41	200.0	118.4	0.0822	3.726	0.00	308.78
42	200.0	121.5	0.0844	3.780	0.00	326.65
43	200.0	125.5	0.0872	3.827	0.00	342.21
44	200.0	130.6	0.0907	3.869	0.00	356.31

Table 1.2 continued

No.	Q	min.	days	Drawdown	Delta Q	Stern. vec.
45	200.0	136.9	0.0951	3.909	0.00	369.53
46	200.0	140.0	0.0972	3.924	0.00	374.82
47	300.0	141.0	0.0979	6.829	100.00	376.43
48	300.0	142.3	0.0988	6.941	0.00	413.80
49	300.0	143.8	0.0999	7.018	0.00	439.27
50	300.0	145.8	0.1013	7.081	0.00	460.30
51	300.0	148.4	0.1030	7.138	0.00	479.25
52	300.0	151.5	0.1052	7.192	0.00	497.21
53	300.0	155.5	0.1080	7.244	0.00	514.79
54	300.0	160.6	0.1115	7.297	0.00	532.38
55	300.0	166.9	0.1159	7.351	0.00	550.25
56	300.0	174.9	0.1215	7.406	0.00	568.59
57	300.0	185.0	0.1285	7.463	0.00	587.53
58	300.0	190.0	0.1319	7.487	0.00	595.72
59	0.0	191.0	0.1326	1.792	-300.00	597.28
60	0.0	192.3	0.1335	1.479	0.00	492.99
61	0.0	193.8	0.1346	1.278	0.00	426.07
62	0.0	195.8	0.1360	1.123	0.00	374.46
63	0.0	198.4	0.1378	0.994	0.00	331.35
64	0.0	201.5	0.1400	0.881	0.00	293.79
65	0.0	205.5	0.1427	0.781	0.00	260.26
66	0.0	210.6	0.1462	0.690	0.00	229.92
67	0.0	216.9	0.1506	0.607	0.00	202.29
68	0.0	224.9	0.1562	0.531	0.00	177.08
69	0.0	235.0	0.1632	0.462	0.00	154.10
70	0.0	247.7	0.1720	0.400	0.00	133.25
71	0.0	263.7	0.1831	0.343	0.00	114.44
72	0.0	283.9	0.1971	0.293	0.00	97.60
73	0.0	309.3	0.2148	0.248	0.00	82.64
74	0.0	341.3	0.2370	0.208	0.00	69.48
75	0.0	381.6	0.2650	0.174	0.00	58.02
76	0.0	432.4	0.3003	0.144	0.00	48.13
77	0.0	496.4	0.3447	0.119	0.00	39.68
78	0.0	577.0	0.4007	0.098	0.00	32.53
79	0.0	678.6	0.4713	0.080	0.00	26.53
80	0.0	806.6	0.5601	0.065	0.00	21.54
81	0.0	967.9	0.6721	0.052	0.00	17.43
82	0.0	1171.1	0.8132	0.042	0.00	14.05
83	0.0	1427.1	0.9910	0.034	0.00	11.29
84	0.0	1440.0	1.0000	0.034	0.00	11.18

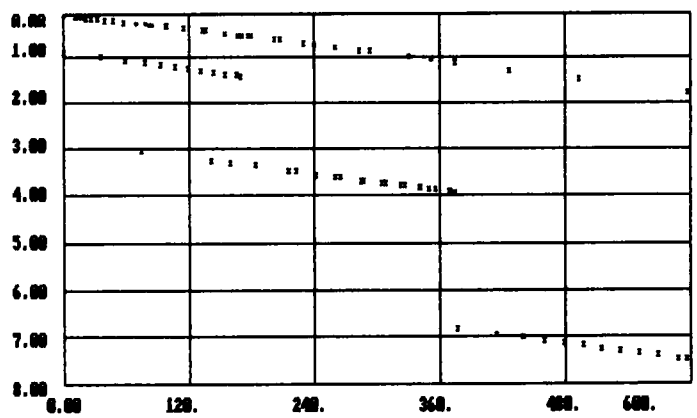
4/ Printout or display of the StrnVec data needs no further explanation, so these notes will go on to explain the calculation of the well equation. The next step is to mark the beginnings and endings of each discharge step. For this purpose, a step has been defined as consisting of a minimum of three consecutive records each having identical discharge rate.

The step numbers, and the first and last records within each step, are displayed as they are recorded.

5/ The discharge rates for all the steps defined above are recorded in the vector StepRate(x). As the data in this recovery step are of great importance in a subsequent part of the analysis, tests should be arranged to finish with a longish recovery period; or to include a reasonably long period of recovery. A recovery step is not essential, but is likely to lead to a more accurate evaluation of the coefficients of the well equation.

6/ In a manual Sternberg analysis, it would be necessary to produce a graph with the values of the StrnVec (along the x axis) plotted against drawdowns (along the y axis). For the sake of this explanation this graph is given in Figure 1.7.

Figure 1.7



A graph of the type which would be used for manual analysis of Sternberg data. Note that on the graph the data from several steps falls on one line. All the data from the zero discharge rate steps, 2, 4, and 7 fall on the top line of the graph, and the data from both the 200 kilolitre per day steps, 3 and 5, fall on the line between 3 and 4 metres drawdown. Refer also to table 1.3 on the following page.

The next section of the program calculates and displays the slopes and y intercepts for all of the steps as if they were plotted on a graph. Assuming a confined, homogeneous, isotropic, unbounded aquifer, all the slopes should be identical. In this imperfect world they are unlikely to be so; therefore the average slope is used to calculate the coefficient, B, of the well

equation. This is displayed immediately after it is calculated (see Table 1.3).

Table 1.3

Step	Y intercept	Slope
1	0.900	0.0030
2	0.000	0.0030
3	2.800	0.0030
4	0.000	0.0030
5	2.800	0.0030
6	5.700	0.0030
7	0.000	0.0030

The user is asked to "Press any key to continue" before progressing further with the analysis, so that he/she has a chance to examine the information given in the displays that accompanied the above operations. If there are more than five steps then the earlier data will scroll off the screen by the time the later data is displayed; to prevent this the program can be temporarily stopped by holding down <Ctrl> and pressing <Num Lock>. To continue after stopping the program in this way, press any key.

This section also calculates the y intercept divided by the discharge rate for each (non zero discharge rate) step, for possible later use. This is stored in the vector YQ(x).

7/ A decision is now made on the method which is to be used to calculate the coefficients A and C, based on the presence or absence of recovery step data.

8/ If no recovery data is present, then the slope and y intercept of StepRate(x) vs. YQ(x), with YQ(x) on the y axis, is calculated by linear regression. Well equation coefficient A is equal to the y intercept produced by this operation, and coefficient C is equal to the slope. As this method relies on projecting the step trends back to the y axis, any slight errors in the step data will be magnified. I have therefore called this the inferior method of calculating the coefficients A and C.

9/ If the data does contain a recovery step, then a rather more involved, but probably also more accurate method may be used to evaluate A and C.

Referring to Figure 1.7, the distances between the Q=0 line (the long, top line) and each of the other lines corresponds to the drawdown at one minute for each respective discharge rate. In the case of the example data

it is not necessary for us to check the graph, we can see what the one minute drawdowns are from the slope and Y intercept information displayed at this point in the analysis (see Table 1.3).

All the slopes are identical for this artificial data, therefore one point on a given line will be the same distance from a second line as is any other point on the first line (because the lines are parallel). The Y intercepts of the steps having zero drawdown are all equal to zero, so the one minute drawdown for Q=100 is 0.9m, for Q=200 is 2.8m, and for Q=300 is 5.7m as would be expected given A=0.004, and C=0.00005, and remembering that the B term is zero because Log 1 (minute) equals zero.

Real data would not be so consistent, so instead of measuring the distance between the lines at their Y intercepts, the distance between them is measured from a point about half way along the data of each non zero step. The one minute drawdowns for each non zero discharge rate step is displayed as it is calculated.

10/ The last step in the solution involves;

(1) Take each step from the first to the second to last in turn. In the program and in these notes, the current step will be called I.

(2) For each step I, take each of the other steps in turn. (Each of these will be called J.)

(3) Subtract the drawdown (or head) of step J from that of step I. Call this D2.

(4) Subtract the discharge rate of step I from that of step J. Call this Q1.

(5) Sum the discharge rate for each of these steps. Call this Q2.

Table 1.4

I	J	D2	Q1	Q2	D2/Q1
1	3	1.9	100	300	0.019
1	5	1.9	100	300	0.019
1	6	4.8	200	400	0.024
1	7	-0.9	-100	100	0.009
5	6	2.9	100	500	0.029
5	7	-2.8	-200	200	0.014
6	7	-5.7	-300	300	0.019

You will notice from the source code that the decision to do these calculations is dependent upon certain restrictions. The calculations will not be done if either of the two steps involved is a recovery step other than the main recovery step, or if the relative difference in discharge rate

between the two steps involved is less than 8%, or if the first of the two steps involved is the main recovery step. The variable TempInt1 is given the total number of the selected datum pairs.

(6) Calculate the slope and y intercept of the best fit line through the selected Q2 (on the x scale) vs. the corresponding D2/Q1. This operation can easily be checked manually by use of the displayed data, if so desired. You will notice from the code that the variables D2, Q1, and Q2 hold data very temporarily, and that the linear regression analysis is given these values as they are calculated. Also, as these values are calculated they are displayed.

(7) Coefficient A is equal to this Y intercept, and coefficient C is equal to this slope. These coefficients, as well as the previously calculated value for B, are displayed at this time.

The coefficients given by the program should be very close to those used in SIM7 to produce the simulated data, but their exact value will depend upon the truncation errors in the particular computer system (usually insignificant in Turbo Pascal), and on the precision lost if data is stored in an ASCII file between SIM7 and this program. In the case of field data any loss of precision will be trivial in comparison with errors of measurement and errors due to disparities between the theoretical model and reality, but the data handling errors may be significant when simulated data are analysed. Using an ASCII file (WTD) for the data transfer, my computer gave A=0.003997, B=0.003001, and C=0.00005001, a maximum error of less than 0.1%. With data transferred in a machine language (FTD) file, and using an 8087 co-processor, their was no error (to the accuracy of the displayed results).

After the display of the coefficients of the well equation, execution passes to the section of code which allows the user to enter trial values for discharge and duration of pumping, and then calculates the drawdown to be expected from each pair of values according to the well equation.

4.4. Rorabaugh's procedure

This has been adapted from Bouwer (1978) and was originally published by Rorabaugh (1953). Note that for this and the next analysis method, the data must be in the form of full recovery between stages; see the notes on 'Conversion to simulate full recovery between discharge stages' above.

At some time, t, after the commencement of pumping, when the water level is no longer declining rapidly, for any given discharge rate the drawdown, s, can be approximated by the equation;

$$s = C_f Q + C_w Q^n \qquad (1.6)$$

where s is the measured drawdown at the given time,

C_f is a constant (at a particular time) relating to the formation,

Q is the discharge rate,

C_W is a constant relating to the well ('C' of the well equation),

and n is the exponent of the well equation.

Bouwer (1978, pp 83, 84) described the procedure. A value is first guessed for C_f and then s/Q-C_f is plotted against Q on double logarithmic paper for each discharge stage. The value of C_f is discovered by making progressive adjustments until the line is as straight as can be obtained. When the value of C_f has been obtained, n can be calculated as the slope plus one. C_W can now be calculated from the equation above. While this procedure is slow and tedious when done manually, it lends itself very well to computerisation.

Given the transmissivity, drawdown and the discharge rate at a particular time, the Y intercept of the delta s slope is calculated, and then the coefficients of the well equation may be evaluated using;

$$A = \{Yi - C_W * \exp[\ln(Q_1) * n]\} / Q_1 \qquad (1.7)$$

$$B = 0.183 / T \qquad (1.8)$$

$$C = C_W \qquad (1.9)$$

where Yi is the y intercept of the log (base 10) of time (on the x scale) against the drawdown (on the y scale),

exp[x] represents the natural exponent of x, ie. e^x,

ln(x) represents the natural logarithm of x,

Q_1 is the discharge rate of the first stage,

T is the transmissivity,

C_W and n are defined above.

If the user does not wish to enter the transmissivity, drawdown, and discharge rate, then the program will look up the drawdowns and discharge rate for the last five readings of the first step, and calculate a slope and y intercept. Transmissivity is calculated from the slope.

These coefficients may now be used in a well equation of the form given in equation 1.4.

As with the Sternberg analysis, execution now passes to the section of code which allows the user to enter trial values for discharge and duration of pumping, and then calculates the drawdown to be expected from each pair of values according to the well equation.

4.5. The simpler s/Q vs. Q method

This method is probably the simplest and most commonly applied (manually) of the solutions of the well equation. A minimum of two discharge steps at different rates are required for solution by this method, three or more are better. The data must be in the form of complete recovery between steps. Several checks on the form of the data are made at the early stages of this procedure.

The program will ask for a time to use in the analysis. All that is required here is a time after the first reading of each step, and before the end of each step. eg. If your test had three steps, each of 30 minutes, and the first reading was taken at one minute in each step, then you could enter any number between one and thirty. Unlike the Rorabaugh procedure, here it should make no difference to the calculated coefficients exactly what time is entered.

The drawdown for each step at one minute (where log time = 0) is calculated by linear regression. (There may be a one minute reading for each step, but as well storage effects can be very significant at such an early time, it was felt better to use all the data of the step to calculate one minute drawdown, than to place great reliance on the one minuted readings.)

The next step is to calculate s/Q for each step (where s is the one minute drawdown, and Q is discharge rate). Here the program departs from the manual method slightly. Manually, the s/Q values would be plotted on the y scale of a graph against Q on the x scale, with both scales being linear. The computer, of course, does not need to plot the data, but calculates the slope and y intercept of the best fit straight line, again by linear regression. This slope and y intercept are then used to calculate A and C of the well equation.

The coefficients are calculated as:

A = The y intercept above,
B = 0.183 / transmissivity,
and C = the slope above.

The transmissivity can be entered by the user, or the user may have it calculated from the data of the first step of the discharge test data.

After solution of the coefficients of the well equation, this procedure also allows the user to do trial projections.

4.6. Checking the results of an analysis

The user may make a note of the coefficients and exponent produced by whichever of these methods he uses to analyze his data, and then use program SIM7 to simulate the discharge test which produced the original data. A comparison of the simulated and real data will show how successful the operation was. I see this ability to use one method or program to check the results of another quickly and easily, as one of the great advantages of microcomputers in comparison to manual methods. The more checks one carries out on ones work, the less likelihood of errors (this applies just as much to the use of computers as at any other time).

Using each of the above methods to solve the well equation for the data used to produce Figure 1.2 the results obtained were:

Entered values	Sternberg	Rorabaugh	s/Q vs. Q
A = 0.003	0.0030	0.0018	0.003
B = 0.005	0.0050	0.0050	0.0050
C = 0.00001	0.00001	0.00008	0.000009
n = 2	2 assumed	1.7	2 assumed

Several of these values are very different to those entered into SIM7, but the important point is not really how well the original values can be retrieved, but rather how well the final well equation fits the time/drawdown/discharge rate data. The table below shows drawdown values calculated for each method for a range of discharge rates and times.

Q (m^3/day)	t (min.)	Sternberg	Rorabaugh	s/Q vs. Q
230	80	3.408	3.413	3.411
230	10	2.369	2.373	2.372
430	50	6.791	6.755	6.747
430	30	6.315	6.277	6.270
300	1200	6.419	6.428	6.410

The figures above show that although the Rorabaugh analysis gave coefficients that differed quite widely from the other methods, when one came to calculating drawdowns the answers were very similar (there is less than one percent variation between all the above drawdowns for a given Q and t). In a real discharge test analysis situation, the choice of the best method to use will be dependent upon the form of the data, (some tests can only be analysed by the Sternberg method) and perhaps the perceived fit between the

model and the real drawdowns. It seems to me to be impossible to say whether or not the ability of the Rorabaugh method to calculate the value of the exponent gives this method a decided advantage over the others, given the variability of field data.

5. THE PROGRAM ITSELF

Program DTDHA is one of the largest programs in this book. If it were significantly larger then it might be better broken into two or more independent, or semi-independent, parts that could be edited and compiled separately.

5.1. The make up of the source code

Rather than have all the source code in one file, it has been broken up into five blocks. There is the main file, DTDHA.PAS, and four 'Include' files, FIRST.SEG, READSAVE.PRC, DTDHMEN2.SEG, and DTDHMEN3.SEG. Turbo Pascal uses Include files as a way of limiting the amount of source code that must be in the computers memory at one time. When it is time to compile the program, the Include files are called up from disk as directed in the source code.

5.2. The Include files

File FIRST.SEG contains variables that are global to all the programs in the GWTS group. It also contains procedures and functions that are used very widely in the group.

File READSAVE.PRC, as the name implies, contains all those procedures and functions that are required to read or save a disk data file. This Include file is also used in many of the programs of the GWTS group.

File DTDHMEN2.SEG contains all those procedures and functions that have to do with the options of menu number two, and file DTDHMEN3.SEG contains those that have to do with menu 3.

5.3. The object code

If compiled as one integral program, the object code of DTDHA would be too large to be useable as a Turbo Pascal chain file. Two large parts of the object code are therefore compiled into separate overlay files; only one of which is called into the computers memory at any one time. The two parts are those that handle everything to do with menu two and menu three respectively.

6. PROGRAM DTDHA.PAS, TECHNICAL COMMENTS

One of the limitations of Turbo Pascal (version 3.0) is that the data space is limited to a maximum of 64k bytes under normal circumstances. In the case of the programs of this book this limitation does not present a problem until one wishes to store more than one set of discharge test data in memory at one time. The way around the problem is to use pointer variables which are dynamically allocated while the program is running. This program has a maximum of two discharge test data files in memory at one time, so two sets of pointers are used (see line 556 of file DTDHA.PAS). This major array variable is named VRP, short for Vector Record Pointer.

VECTOR RECORD POINTER Each VRP array consists of three vectors which are themselves made up of an array of 500 reals. Note that the memory used by these arrays will be greater when Turbo 87 is used than with normal Turbo, as the former uses eight bytes per real and the latter six. For the same reason, if the programs are compiled under Turbo 87, then they will not be able to read data files of the FTD type produced by a program compiled under normal Turbo, and vice-versa.

The source code of this program is contained mainly in three files. The procedures and functions of each file are explained below under the name of the file in which they occur. The order is the same as that in which compilation takes place, beginning with DTDHA.PAS, and followed by DTDHMEN2.SEG, and DTDHMEN3.SEG. Within each file the explanatory notes are arranged in alphabetical order of procedure/function name.

In the notes that follow, full explanations will be given only for those procedures and functions who's workings might not be self explanatory.

6.1. Procedures and functions of file DTDHA.PAS

Display sub-procedure Line 239

Purpose: to display a short string at the current cursor position, and after a delay of one second, to erase the message, and return the cursor to it's initial position.

DisplayRecentData procedure Line 175

Purpose: to display the data of the last three readings so that the user can check his/her progress in data entry.

The heading is displayed first, and then, in order, the data of the third to last, second to last, and last readings. eg. Lines 182-184 display the data of the third to last reading, so long as such a reading exists (line

180). The statements controlling the position of this display are in lines 177, 179, 187, and 195.

DispMenu procedure Line 517

Purpose: to display the main menu of program DTDHA.

Before the menu itself is displayed, some information about the data at present in memory is given. Firstly, line 516 tests whether there are any well test data in memory. If data are not present, then line 517 writes an appropriate message. If data are present, then the last named file is given. (Note that this may not be the name under which the data in memory are stored, it might rather be the name of a file merged with the original data, or a name under which some part of the data has been saved.) Lines 523 to 530 then write notes on the recorded test type, well type, and the value of R (which, you may recall, is the distance from the piezometer to the discharging well when the data refers to a piezometer, but is the effective radius of the well when the data refers to a discharging well).

Lines 532 and 533 are to give a warning that the data in memory are not the same as those of any disk file. The boolean variable, DataSaved, is set to false whenever a call is made to Menu 2, the data modification menu, and in some other cases when it may be necessary to change the data in memory (eg. conversion of recovery times to t/t1).

Finally, the menu itself is displayed by the code starting at line 535.

EnterDate subprocedure Line 249

This is a subprocedure of procedure EnterTimeDate.

Purpose: to allow the user to enter the time of day and the date of a single reading, from the keyboard.

The moment of the particular reading is entered as a minute, hour (on a 24 hour clock, not am/pm), day (as a calendar date), month (by number), and year (ideally in full eg. 1988, but for most purposes the short form, eg. 88, is normally acceptable, see main text). Entry of the minute takes place on line 257, and is validated by the code from there to line 264.

The scene has been set by procedure EnterWTD (line 456), which displayed instructions applying to entry of times: as minutes, and as date and time of day.

Note that the Boolean variable GoBack, which is defined in the parent procedure EnterTimeDate, is made true for a minute entry of -1. The value of GoBack will be checked by EnterTimeDate, and if found to be true, then the procedure rolls back to the previous entry.

Line 275 refers to Boolean variable SecondTime. If a date is being entered for the first time, then it is necessary to enter minute, hour, day, month and year, but if a date has already been entered then it might be reasonable to assume that the month and year remain the same. SecondTime and another Boolean, GoOn (set in function ReadIntF), control these conditional entries.

Called by: the parent procedure, EnterTimeDate.

Calls: functions ReadInt and ReadIntF.

EnterMinutes procedure Line 424

Purpose: to enter and validate a time in minutes, enter a drawdown, and to ask the user whether the discharge rate is to be changed.

Called by: procedure EnterWTD.

Calls: procedures DisplayRecentData, ReadDrawdown, and function ReadReal.

EnterTimeDate procedure Line 206

Purpose: to assist the user to enter discharge test times as date and time of day.

This major procedure contains functions ReadIntF, GetMinutes and TestDate; and procedures Display and EnterDate.

The first steps are to display a heading and to ask the user to enter the starting time. (If the total duration of discharge is to be calculated, then a starting time as well as a time of reading is required.) Procedure EnterDate is called to obtain the date from the user (line 372). Now that the starting time is known, it is displayed for the users reference (line 377). Line 381 uses function GetMinutes to calculate the time in minutes from the beginning of the year zero to the entered date and time. (This requires around nine decimal digits of precision, normal Turbo Pascal has roughly eleven, Turbo 87 has around 15.) Line 383 begins a loop which controls entry of all the times and dates of the discharge test readings.

Line 386 causes the next time and date to be entered, and if the user has indicated a desire to delete the last reading, then line 387 sets back the counter. (Procedure EnterDate will not allow the user to set GoBack on the first discharge test reading for obvious reasons.) Lines 391 to 396 set the month and year to the default if the user has not specifically entered them.

To avoid unnecessary key punching, the discharge rate for any reading other than the first is assumed to be the same as the immediately previous

one unless the user includes a '+' sign with the drawdown entry. (The '+' has been chosen because it is present on the standard numerical key pad and it indicates that something is to be added.) If procedure ReadDrawdown detected the '+' sign, then the new discharge rate is entered in line 410.

Line 414 displays the data of the previous entry to make keeping track of progress in data entry easier.

Called by: EnterWTD

Calls: DisplayRecentData

EnterWtd procedure Line 456

Purpose: to supervise the entry of discharge test data via the keyboard.

Called by: the main part of DTDHA

Calls: functions CapOptions and ReadReal, and procedures EnterTimeDate and EnterMinutes.

FileExist function Line 35

Purpose: to check for the existence of a file by a given name.

This function is redundant and may be deleted; unfortunately this fact was not realised until the work of writing this book was too far advanced to allow deletion.

FirstLast procedure Line 56

Purpose: Many operations relate to a part of the data which can be specified by a first and a last record number. This procedure allows the user to enter these.

The only part of the procedure that might require explanation is the use of the defaults, around line 69 to 83. Integer variables First and Last are set to the first record number (1), and the last record number (the number of data sets in vector set number 1) in line 69. ReadIntInput (of file First-.seg) is called to enter two temporary integers (lines 72 to 75). Function ReadIntInput is written to return a value of zero if the user presses the enter key without first pressing some other key. The pressing of <Enter> without first typing a number is taken as indicating the the user wants to use the default values, the first and/or last records. Lines 81 and 82 change the values of First and Last if the user entered valid numbers for these variables.

GetMinutes sub-function Line 341

Purpose: to calculate the number of minutes from the beginning of year zero to the current time and date (on the false assumption that the Gregorian calendar was in use for all that time).

Lines 350 and 351 calculate the required number for all dates except those in a leap year and before 29th. of February; line 357 corrects for such a date.

Note that by the Gregorian calendar there is a leap year on every year whose number is divisible by 4, except those whose numbers are divisible by 100 and are not divisible by 400. eg. The year 1896 was a leap year (divisible by 4), the year 1900 was not (divisible by 100), and the year 2000 will be a leap year (divisible by 400).

Called by: parent procedure EnterTimeDate

LinReg procedure Line 87

Purpose: to produce a best fit linear (straight line) function from a set of data by linear regression.

The linear function is of the form;

$$y = s\,x + Y \tag{1.10}$$

where x and y are respectively the input and output of the function,

s is the slope,

and Y is the y intercept.

LINEAR REGRESSION The equations used for linear regression are;

$$s = \frac{n\,S_1 - S_2\,S_3}{n\,S_4 - S_2^2} \tag{1.11}$$

$$Y = \frac{S_3\,S_4 - S_2\,S_1}{n\,S_4 - S_2^2} \tag{1.12}$$

where S_1 is the sum of all $x_i\,y_i$,

S_2 is the sum of all x_i,

S_3 is the sum of all y_i,

S_4 is the sum of all x_i^2,

and n is the number of x, y pairs in the data under-going regression analysis.

Called by: 1/ sub-procedure LinRegCall of procedure SimulateRec (file DTDHMEN2.SEG),

2/ procedure SimulateRec,

3/ procedure Rorabaugh (file DTDHMEN3.SEG),

4/ procedure SQvsQ (from three points) (file DTDHMEN3.SEG),

5/ subprocedure SolvStrn of procedure Sternberg (file DTDHMEN3.SEG).

Log function Line 50

Purpose: to calculate logarithms to the base ten.

Constant LnTen is given the value of the natural logarithm (ln) of ten in line 34.

PrintData procedure Line 109

Purpose: to produce a properly paginated and easily readable paper copy of the current set of discharge test data.

Line 122 causes the contents of the variable, FileName to be printed as part of the heading. Note that if data in memory has been merged with that from a second file then this variable will contain the last used file name, and may not be the file name that would be required by the user. Lines 138 to 143 place a string of dashes after every seventh line with the aim of making the printed data more easily readable.

Called by: the main part of program DTDHA.

ReadDrawdown procedure Line 150

Purpose: to read in a drawdown value from the keyboard, look for a '+' sign (which would indicate the users desire to change the discharge rate) and reject typographical errors.

If the user has entered a '+' sign, then Boolean variable NewRate is given a true value. (If the pos function fails to find a '+' in the string Short in line 159, it passes a value of zero to J.) Variable NewRate will be referred to by the calling procedure.

Called by: procedures EnterTimeDate and EnterMinutes.

ReadFile procedure Line 500

Purpose: to control the reading of data from a disk file; in particular to warn the user, and give a way out, if there is a danger of overwriting the data already in memory.

Called by: the main part of program DTDHA.

ReadIntF sub-function Line 217

Purpose: to read an integer from the keyboard, and to check the entry for a '+' sign.

The function is very similar to ReadInt which was described in the Preliminary section. It also has some similarity to the ReadDrawdown procedure described above.

SaveFile procedure Line 484

Purpose: to control the saving of data to a disk file.

Called by: the main part of program DTDHA.

TestDate sub-function Line 305

Purpose: to check that the entered date is a valid calendar date.

The greatest part of the code, lines 310 to 333, check that the entered day number is valid considering the length of the particular month; this involves checking for leap years if the month is February.

Called by: the parent procedure, EnterTimeDate.

6.2. Procedures and functions of file DTDHMEN2.SEG

This file contains all of, and only, overlay procedure MenuTwo. All of the 'procedures' listed below are therefore really subprocedures that are not available globally.

By calling this and the next file (DTDHMEN3.SEG) overlay procedures, Turbo can bring one or other into memory as required. The limitation of 64 kilo bytes of object code would be exceeded if an attempt were made to load both these files (in their compiled form) into memory at the same time and with the first file DTDHA.PAS. The calling into memory of the overlay procedures is handled by Turbo Pascal, and is not noticeable to the user except for the small delay due to disk access.

AddConst procedure Line 138

Purpose: to allow the user to modify any one of the three main vectors, times, drawdowns, or discharge rates, by adding a constant to all the records of a contiguous section of one of them.

Called by: the main part of overlay procedure MenuTwo.

Calls: procedure FirstLast, and functions CapOptions and ReadReal of file DTDHA.PAS.

CalcCorrection sub-procedure Line 12

Purpose: to calculate the amount of correction necessary to bring a measured drawdown into line with the readings from a background monitoring well. It is assumed that the background 'noise' is mainly barometric in nature, and that it effects both the test well and the background well in a linear relationship.

The procedure allows for the mis-match between the time of the drawdown reading and the time of the background reading by linearly interpolating two background readings using the equations;

$$S = \frac{s_{2,i} - s_{2,i-1}}{t_{2,i} - t_{2,i-1}} \qquad (1.13)$$

$$C = F * s_{2,i-1} + S * (t_1 - t_{2,i-1}) \qquad (1.14)$$

where S is the slope of the line between the two readings from the background well which bracket the time of the reading in the test well/piezometer.

$s_{2,i}$ is the ith drawdown reading taken from the background monitoring well. The ith reading has been chosen as the first reading in background wells data that has a time greater than that of the drawdown to be corrected.

$s_{2,i-1}$, from the above can be seen to be the last reading in the background wells data having a time less than, or equal to, that of the drawdown to be corrected.

$t_{2,i}$ is the time of the ith reading in the background well.

C is the required amount of correction, in metres.

F is the correction factor, which will be one if both wells have equal barometric susceptibility.

(s_1 or t_1 would apply to data from the file to be corrected.)

Called by: the parent procedure, CorrectForBackground.

ChangeDescription procedure Line 547

Purpose: to allow the user, simply and easily, to change the parts of the file which describe the test from which the data came.

Called by: the main part of procedure MENUTWO.

CorrectForBackground procedure Line 7

Purpose: to control the process of correcting the drawdowns in memory for background noise such as barometric variations.

The first section of the procedure explains what is to be done, then gives the user a chance to exit. Line 46 causes the file holding the

background data to be loaded into memory under VRP (vector record pointer) 2, and then line 51 asks for the correction factor (see Equation 1.14, above).

If the times in either file are not in ascending order the correction procedure will not work correctly, so lines 54 to 72 check and report on this. If some of the later times in the file to be corrected are greater than those in the background file then it is not possible to correct all the data. In this case those data that can be corrected will be, and the file will be truncated and a message displayed (line 82).

The actual correction of the data is done by adding the original drawdown to the value of function CalcCorrection, for the given time, in line 80.

Called by: the main part of overlay procedure DTDHMEN2.

Calls: sub-function CalcCorrection, and procedure ReadTestDataFile.

DeleteReading procedure Line 226

Purpose: to delete a particular reading from the data file currently in memory.

The procedure overwrites the specified reading with the data of the following reading, then overwrites the following reading with the data of the one after that, etc. until the end of the file. To clarify this explanation an illustration will be given.

Suppose that there is a file of data which consist of letters of the alphabet, each of which has a numbered position in the file, and it is desired to delete record number 6 (character F) of the file. We begin with,

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A B C D E F G H I J K L M N O P.

The first step is to copy record number 7 to 6:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A B C D E G G H I J K L M N O P.

This has got rid of the unwanted F, but now there are two copies of G, so next we copy record number 8 to 7, removing the superfluous G,

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A B C D E G H H I J K L M N O P,

and then repeat this process until the end of the file is reached. When we come to the point where there are two copies of the P, we no longer need to over-write, we simply delete the last record by shortening the file. Finally we have,

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
A B C D E G H I J K L M N O P,

which is what is required.

Procedure DeleteReading works in exactly this way, with the only difference being that there are three vectors, rather than one, and the same numbered record is removed from each.

Called by: the main part of overlay procedure DTDHMEN2.

Calls: function CapOptions.

DeleteReadings procedure Line 261

Purpose: to remove a contiguous group of readings.

This procedure works in very much the same way as the previously explained procedure DeleteReading, but instead of copying the records from one position to the adjacent position, now it is necessary to copy them over a greater distance. Using the same example as above, suppose we wish to remove records 6, 7, 8, and 9. The first step is to copy record 10 down to position 6,

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A B C D E J G H I J K L M N O P,

then record 11 is copied to position 7, and so on until the end of the file. Finally the file is shortened by four records.

Called by: the main part of overlay procedure DTDHMEN2.

Calls: function CapOptions.

DispMenu2 procedure Line 571

Purpose: to display the options of menu number two, the data modification menu.

Called by: the main part of overlay procedure DTDHMEN2.

LinRegCall sub-procedure Line 299

Purpose: to obtain a best fit slope and y intercept for a selected set of times and drawdowns with the times being treated as logarithms (base 10) of times.

The procedure simply places the logs of the times into one vector, TempVec1, and the corresponding drawdowns into another vector, TempVec2, and passes these two vectors to procedure LinReg. K1 holds the number of elements that is contained in each of the vectors.

Called by: the parent procedure, SimulateRec.

Calls: procedure LinReg of file DTDHA.PAS.

MarkRec function Line 424

Purpose: to find, and record the location of, the first period of zero discharge.

Line 432 gives counter I the value of the number of the first record having a zero discharge rate, or if there are no zero discharge rates, then I will be equal to the number of readings in the file. Line 436 sets StartRec equal to the number of the first recovery reading, and DischTime equal to the total duration of discharge preceding the first period of recovery, so long as a period of recovery was found. Line 441 records the number of the last record having a zero discharge rate.

The values picked out above are displayed, and the user is asked if he is satisfied with them. If he is not (perhaps the first recovery period on file is a very short period of pump failure which the user judges may be safely ignored) then he must enter the times manually. Manual entry of the number of the first recovery reading is also possible is the program is unable to find it (perhaps flow rates were recorded by an automatic device which recorded some very small number, other than zero, when the pump was stopped). The remainder of the code allows manual entry of reading numbers, and performs some validation of the entered values. Finally function MarkRec is given a value of true if the algorithm has judged the found or the entered values to be valid.

Called by: procedures TOverTOne (t over t one) and RTMinusRTOne (root t minus root t one).

Calls: functions Response, ReadInt, and ReadReal.

MultByConst procedure Line 173

Purpose: to allow the multiplication of all, or any contiguous group, of the data by a constant.

Line 183 calls procedure FirstLast to have the user enter the numbers of the first and last records that are to be altered. Line 186 sets ParNum (parameter number) to 1, 2, or 3, depending on whether Times, Drawdowns, or discharge rates are to be changed. The remainder of the code displays some appropriate conversion factors, asks the user for a constant, and multiplies the chosen data by that constant.

Called by: the main part of overlay procedure MENUTWO

Calls: procedure FirstLast of file DTDHA.PAS, and functions ReadReal and CapOptions of file FIRST.SEG.

RTMinusRTOne procedure Line 503

Purpose: to convert the times associated with residual drawdown readings recorded from a test carried out in a strip aquifer to the square of (root t minus root t one), so that the plotted recovery may parallel the plotted drawdown.

Line 510 calls function MarkRec to obtain the number of the first and last readings of the recovery period, the total duration of discharge (up to the first period of recovery), and to perform some checks on the validity of the data for the following operations. The conversion of times is carried out by lines 515 and 516 using the following equation:

$$R_i = (t_i^{1/2} - t'_i{}^{1/2})^2 \qquad (1.15)$$

where R_i is the ith value of (root t minus root t one) squared,

t_i is the time of the ith drawdown reading measured from the commencement of discharge,

t'_i is the time of the ith drawdown reading measured from the end of discharge (alternatively called t one).

It is important to bear in mind that the values of R_i replace the original values of t_i in the main time vector; the original times are unrecoverable from the t/t' data.

Called by: the main part of overlay procedure MENUTWO.

Calls: function MarkRec.

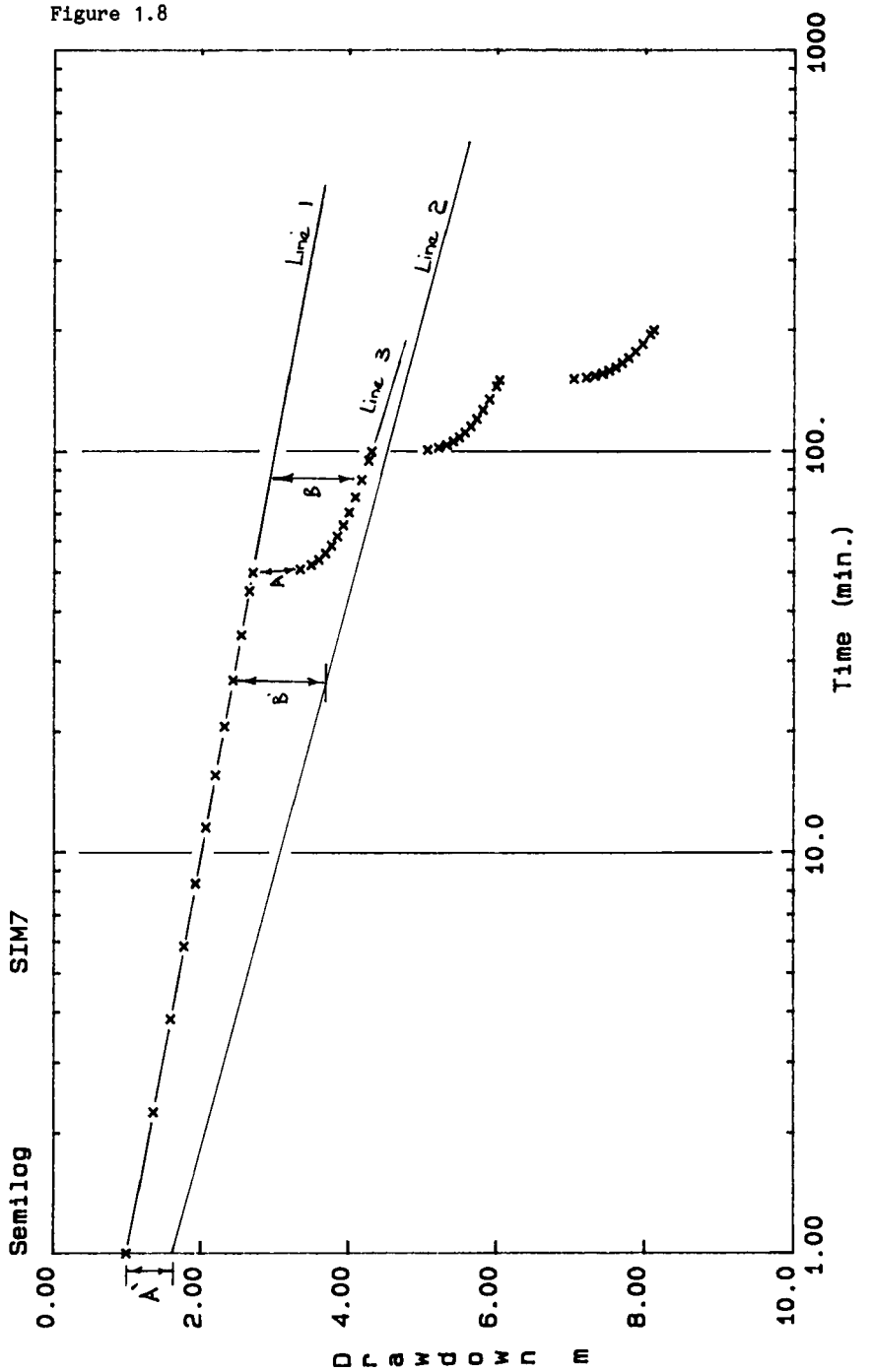
SimulateRec procedure Line 287

Purpose: to convert the data recorded from a discharge test consisting of several stages of discharge at different rates immediately following each other, to simulate a set of single stage tests separated by periods sufficiently long to give full recovery from discharge.

Of all the parts of the programs covered in this book, this has given me the most trouble and I still am not satisfied that the results are as good as might be obtainable. It is a complex procedure, and is difficult to follow fully. In spite of this I believe that it gives reasonably accurate results for data consisting of a set of discharge steps consisting of steps of equal, or nearly equal, lengths. When one uses this procedure, one assumes that the aquifer behaves as an infinite, isotropic, and homogeneous confined aquifer, such as would produce a graph basically similar to that of Figure 1.8. One failing of this procedure is that if a set of short steps is followed by a

(Continued on page 78)

Figure 1.8



A graphical representation of the process of converting multistage discharge test data to simulate full recovery between stages.

long step, then the later drawdowns of the last step will contain quite large errors.

Line 313 calls function CheckForStepData to test the validity of the data for this procedure. The conversion process starts with line 317, and will be described as a series of steps.

1/ The first discharge stage does not need any conversion. It's slope and y intercept are recorded at line 323.

2/ This procedure is an adaptation of a graphical procedure (I regret that I cannot give a published reference) that involves, in the first place, plotting the original data on a semilogarithmic graph. The next step (line 327) involves calculating, for each drawdown reading of step two, the additional drawdown, beyond that which would have been expected from the continuation of step one, due to the increased discharge rate. In effect, the drawdown line of step one is extended, (line 1 of Fig. 1.8) and the distance each drawdown reading of step two is below this line is recorded (eg. distances A and B of Fig. 1.8).

3/ It is reasoned that if an additional drawdown of x metres occurred at y minutes into the second stage (due to the increased discharge rate), and the drawdown due to the discharge rate of the first stage at y minutes (from the beginning of that stage) was z metres, then had the test began at the discharge rate of the second stage the drawdown at y minutes (from the beginning of the first stage) would be x plus z. Lines 326 to 336 use this reasoning to convert the data of step two. (eg. distances A and B are transferred to A' and B' in Fig. 1.8.) This produces line 2 of Fig. 1.8.

4/ The conversion of subsequent steps is more complex because there is no longer a straight line set of data from which the increased drawdown can be measured. To overcome this deficiency the curve of the plotted data of step two is extended to where it might have been had the duration of step two been doubled (line 3 of Fig. 1.8). This is done by adding to the drawdown due to step one (symbolized by line 1), the additional drawdown that would be expected due to pumping at the rate of step two for the given time (symbolized by line 2 of Fig. 1.8).

In the case of the example data, for example, there is a reading at time=58.4 min. As we are extending line 3 beyond the end of step 2, ie. beyond 100 minutes, we calculate the drawdown that would be due to discharging at the rate of step one for 50+58.4 minutes. (Line 347 of the program.) To this must be added the drawdown that would be expected had we pumped only at the rate of step two right from the beginning of the test, for 58.4 minutes (because we are adding the drawdown due to 58.4 minutes pumping at

the rate of step two to the drawdown due to 108.4 minutes pumping at the rate of step one). This figure is calculated by subtracting the drawdown due to rate one at 58.4 minutes (program line 348) from the drawdown calculated for rate two at 58.4 minutes (program line 349).

5/ The slope and y intercept of this line is calculated by the linear regression call of program lines 356 to 363. Now it is possible to calculate the additional drawdown due to the increased rate of the third step by measuring the distance from this line to the plots of the third stage (program lines 365 to 376).

Note that even in theoretical data the algorithm has an imperfection at this point. It assumes that line 3 is straight where-as it would in fact be slightly curved. It may, or may not, be that this is the only theoretical error in the algorithm.

6/ The above steps are repeated for all subsequent stages.

Called by: the main part of overlay procedure MENUTWO.

Calls: sub-procedure LinRegCall and procedure CheckForStepData.

SortForTime procedure Line 382

Purpose: to sort the data into order of increasing times.

A Shell-Metzner sorting algorithm is used. It is beyond the scope of this book to describe the operation of the algorithm, refer to Miller, 1981; Barden, 1980; or to any of the more complete books on general programming.

Called by: the main part of overlay procedure MENUTWO.

TOverTOne procedure Line 486

Purpose: to convert the times associated with residual drawdown readings recorded from a test carried out in an infinite confined aquifer to t/t', so that the plotted recovery may parallel the plotted drawdown.

Line 493 calls function MarkRec to obtain the number of the first and last readings of the recovery period, the total duration of discharge (up to the first period of recovery), and to perform some checks on the validity of the data for the following operations. The conversion of times is carried out by lines 495 and 500 using the following equation:

$$T_i = t_i / t'_i \qquad (1.16)$$

where T_i is the ith value of t/t',

t_i is the time of the ith drawdown reading measured from the commencement of discharge,

t'_i is the time of the ith drawdown reading measured from the end of discharge.

The values of T_i replace the original values of t_i in the main time vector.

Called by: the main part of overlay procedure MENUTWO.

Calls: function MarkRec.

6.3. Procedures and functions of file DTDHMEN3.SEG

DataValid function Line 109

Purpose: to check that the data of the first discharge stage is suitable for use in calculation of transmissivity.

The last four readings of the first stage are used for this calculation, and, as well storage may effect the first one or two readings, it is undesirable to use them. Therefore, if there are less than seven readings in the first stage, this function rejects the data for the purpose of calculation of transmissivity.

Lines 117 to 120 set the marker variable I to point at the last reading of the first stage. This having been found, I is reset to what will probably be the first reading to be used for the calculation of transmissivity (line 123). Lines 125 to 133 then display the discharge rate for each of the selected readings, and line 134 rejects the data if the selected readings are too close to the beginning of the first discharge stage.

Called by: procedure Rorabaugh.

DisplayData sub-procedure Line 464

Purpose: to display the data of the Sternberg vector.

Called by: sub-procedure StrnMenu of the parent procedure Sternberg.

DispMenu sub sub-procedure Line 646

Purpose: to display the small Sternberg menu.

This is a sub-procedure of sub-procedure StrnMenu.

Called by: sub-procedure StrnMenu of procedure Sternberg.

DispMenu3 procedure Line 728

Purpose: to display the Well Equation solution menu.

Called by: the main part of overlay procedure MENUTHREE.

EnterData sub-procedure Line 99

Purpose: to allow the user to enter the data to be used for the latter part of the calculation of the Well Equation by Rorabaugh's method, if either the data are unsuitable, or he/she so chooses.

This subprocedure also calculates the value of Delta s (base e) by using the equation [Bouwer, 1978 (modified)]:

$$\text{Delta } s = \frac{0.183\ Q}{T \ln 10} \qquad (1.17)$$

where Q is the discharge rate,

T is the transmissivity

The natural log of 10 (ln 10) is used to produce Delta s to the base e rather than the more familiar Delta s to the base 10.

Called by: the parent procedure Rorabaugh.

GetDdAndRate procedure Line 8

Purpose: Both the Rorabaugh method and the s/Q vs. Q method require a drawdown and a discharge rate for a given time in each of a number of discharge steps. This procedure produces those values, and at the same time, counts the number of discharge steps before any period of zero discharge.

There are several potential problems that this procedure must check for. The user enters the time at which the analysis method is to be applied, so the first test is whether the data will allow calculation of a drawdown and recording of a discharge rate for the given time. If the entered time is before the first reading then it is rejected with the message of line 22. If it is discovered that one of the steps ends before the specified time is reached, then the error message of line 33 is given. The final check in this section is that there are at least three steps (lines 37 to 41).

The linear interpolation equation of lines 45 to 47 produces the desired drawdown figure for each step at the time specified by the user. Note that at the time this equation is applied the specified time (in variable RorTime) is between the time VRP[1]^.TimeVec[I] and VRP[1]^.TimeVec[I+1], and that the linear interpolation considers only these two values. The rate for each step is recorded in the following line, 48.

Lines 53 to 55 move the pointer, I, on to the beginning of the next step; or to the end of the discharge part of the test data. The number of steps is recorded in line 59. Notice that the step counter, StepNum, is incremented in line 52, before it is proven that there is another steps data to follow. It is because of this that 'StepNum-1' is used in line 59.

Called by: procedures Rorabaugh and SQvsQ.

PrintSternberg procedure Line 428

Purpose: to produce a printed copy of the discharge test data including the Sternberg vector. Such a listing would be useful if the user wished to check the analysis by manual means.

Called by: subprocedure StrnMenu of procedure Sternberg.

Rorabaugh procedure Line 88

Purpose: to calculate the values of the coefficients of the well equation including the exponent.

This is a major procedure which, including it's several sub-procedures, uses 188 lines of code. The main part of the procedure begins at line 137 and ends at line 274. Rorabaugh's method of solution of Equation 1.6, and the method used to calculate the coefficients of the Well Equation from that point, have been explained previously, so this section will deal only with the 'mechanical aspects.

The Rorabaugh operation proper is handled by lines 167 to 225. The first aim of this section is to calculate the best fit formation constant (FormConst in the program, C_f in Equation 1.6). It will be recalled that the value of C_f is discovered by experiment and use of a graph of Q against $s/Q-C_f$ (where s is drawdown, and Q is discharge rate). This being so, and a negative value for C_f being ruled out on grounds of logic (if the formation constant was negative, then the greater the flow rate the less the head loss), the maximum possible value for C_f is s/Q. The experimental method used for determining C_f is that of successive bisection (Tremblay and Bunt, 1981).

Briefly, successive bisection is a method that can be used to find a solution indirectly, when no direct solution is known. It is workable when one has a test that can be applied to any potential solution to show whether that solution is correct, too great, or too small. Here we must test for experimental values of C_f. We have a maximum value for C_f (line 175), we now choose a very small number, 1E-36 (ten to the minus 36th power), as a minimum value (line 176). The true value of C_f should be between these limits. Now we choose a point half way between these two values (line 178). (This could be the arithmetic mean, the geometric mean, or any other mean. Here the geometric mean has been used.) This value is tested. There are three possibilities; it might be too large, too small, or acceptable. If it is found to be too large then it becomes the new upper limit, if too small

then it becomes the new lower limit, otherwise it is taken as our solution, the value of Cf.

The x and y coordinates are calculated in lines 182 and 183, and the slopes of the 'lines' joining these 'points' are calculated in lines 185 and 186. We now need to know whether, when joined together, these lines tend to have a concave or convex shape (looking at them from above). As there may well be more than two 'lines' (actually line segments) involved, it will quite probably be necessary to test for the average angle (loosely called curve here) between lines. This is done in lines 190 to 195.

With some discharge test data this method will indicate either a negligible, or even a negative value for the formation constant. (Presumably negative formation constants are a product of slight errors in the data, or are due to the model not accounting for all the variables involved in the real world situation.) This will cause an early completion to this part of the algorithm due to the invocation of the code of lines 198 to 202.

Lines 207 to 212 complete the Rorabaugh analysis proper, with the results being displayed by lines 213 to 225. The display shows the calculated formation loss, well loss, and total calculated head loss, together with the actual drawdowns (for the given time) for comparison. Please note that the variables called A and B at this point are not the A and B of the Well Equation, but hold temporary values for display.

It will be recalled that Cf above is identical to coefficient C of the Well Equation, and Exponent above is the exponent of the Well Equation. In order to calculate the values of the remaining coefficients, A and B, it is necessary to calculate the slope and y intercept of one of the discharge steps. The first discharge step is used because it is the only one that has not been changed by the conversion process, with the possible introduction of errors. Function DataValid (called in line 239) checks the validity of the data for this calculation, and sets a marker at the last reading of the first step (LastReading). Lines 249 to 255 call procedure LinReg to produce a best fit slope and y intercept from the drawdowns against the logs of times for the last five records of the first discharge stage. From that point it is a simple operation to calculate the remaining values, A (line 262), and B (line 263), using Equations 1.7 and 1.8.

The final significant operation is to allow the user to experiment with the derived Well Equation, and produce projections given any chosen discharge rate and duration of discharge. This can be used to both check the validity of the results, and to calculate the optimal discharge rate for the well.

Called by: the main part of procedure MENUTHREE.

Calls: procedures GetDdAndRate, EnterData, LinReg, and RunTrial; and functions ReadReal and DataValid.

RunTrial procedure Line 62

Purpose: to allow the user to produce trial projections based on the derived values for the Well Equation.

The Well Equation (Equation 1.4) is in two forms on lines 80 and 81, the first of these being used when the exponent is assumed to be (or perhaps happens to be calculated as) two. Unlike Basic and Fortran, Turbo Pascal does not include an exponentiation function which might be used to raise a number to any given power. Therefore, when a power of a number is to be obtained, the same method is used as would be employed with a set of logarithmic tables. ie. Obtain the logarithm of the number to be raised to a power [ln(Rate) in line 81], multiply by the exponent, and convert back from a logarithm to a simple number.

Called by: procedures Rorabaugh, SQvsQ, and sub-procedure SolvStrn of procedure Sternberg.

Calls: function ReadReal.

SolvStrn procedure Line 491

Purpose: to calculate the values of the coefficients of the Well Equation, given the Sternberg vector and the discharge test data.

A comment on terminology is worthwhile at this point. In the following discussion, 'discharge test stages' will be referred to. By this is meant those parts of the discharge test which consist of periods of constant flow rate. By this definition, a flow recession stage (where water flows from an artesian well against a constant head, at a declining rate), would not be considered to be a 'discharge test stage'.

The user will be referred back to the main text (section 4.3 of Chapter 1) as this procedure is explained. Line 501 calls on procedure MarkStepData to produce a discharge rate and first and last reading numbers for each distinct discharge stage of the test. This information is displayed by lines 506 to 512.

Lines 514 to 526 discovers which of the zero discharge rate stages contains the greatest number of readings. It is this stage that is considered to be the main recovery stage for the purposes of this analysis (point 5/ of the main text). TempInt1 holds the number of readings in the longest recovery period currently recorded as this loop checks through all the steps. The sequential number of the main recovery step is stored in variable

RecStepNo (line 522); this retains the value zero if there is no recovery stage.

The code of lines 527 to 544 calculates the slope and y intercept of all the discharge test stages by calling the linear regression procedure (LinReg, line 538). Also, the value of the y intercept divided by the discharge rate for each step is recorded at this time (line 543). The average of these slopes is the value of B in the Well Equation (line 545). This section is explained under point 6/ of the main text.

The greater part of the remaining portion of this procedure is divided into two sections, only one of which is used for the analysis of any given test's data. These are called the 'inferior' and 'superior' methods for producing the values of the coefficients A and C of the Well Equation. The inferior method uses the code of lines 552 to 569, while the more complex superior method is carried out by lines 570 to 634. The inferior method should be understandable by consulting the code in conjunction with the main text under point 8/.

The first task in the superior method is to calculate the one minute drawdowns for each of the (non zero rate) discharge test stages. Referring to the main text under point 9/, to figure 1.7, and to the code of lines 575 to 584; variable TempInt1 points to the middle plot of the non-zero discharge rate stage. Lines 579 and 580 then calculate the y coordinate on the zero discharge rate line at a point vertically (on a graph) above this point.

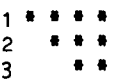
Please refer to the section of the main text on point 10/ regarding this part of the solution. As suggested by the literal strings which are displayed by lines 596 and 597, a number of values are calculated from the data of chosen pairs of discharge stages. The maximum number of sets of these values that may be calculated is given by the equation:

$$m = \frac{(n-1)\ n}{2} \qquad (1.18)$$

where m is the maximum number of sets,

and n is the number of discharge stages (including the main recovery stage). This number of sets of values will be used only when there is one period of zero discharge, and all discharge stage rates are different from each other by at least 8%.

As an example of the stage combinations that would be used in this ideal condition. Given that there are 4 pumped stages, followed by the recovery stage (stage number 5), the sets of values would be calculated for the stage combinations indicated by the asterisks below:



In the example data, the stage combinations are:

No combinations use stage 2 or 3 data because those stages are recovery stages other than the main recovery stage. Stage 3 data is not used because it has the same discharge rate as stage 5.

As these values are calculated, they are displayed, and sums are produced for use in a linear regression algorithm (lines 619 to 623). The linear regression is finally carried out on lines 628 and 629 [see 10/ (6) in the main text]. Coefficient A of the Well Equation is made equal to the calculated y intercept, and coefficient C, the slope (line 630).

Whether the inferior or superior methods are used, procedure RunTrial is called (lines 568 and 633) to allow use of the derived Well Equation for projections or checking purposes.

Called by: subprocedure StrnMenu of procedure Sternberg.

Calls: procedures MarkStepData, LinReg, and subprocedure RunTrial.

SQvsQ procedure Line 276

Purpose: to calculate the coefficients of the Well Equation.

This is the simplest solution of the well equation used in this program. The first part of the code (lines 291 to 315) does some checking on the form of the data, and counts and marks the beginnings and endings of the stages. Notice that the stages are counted by searching for a time that is less than the immediately preceding time (line 298). Line 318 calls procedure GetDdAndRate mainly to do some more checking on the validity of the data.

As was the case in the Rorabaugh analysis, a transmissivity figure is required for the calculation of the value of coefficient B, and this may either be entered manually, or calculated from the data of the first discharge stage using the equation:

$$T = \frac{0.183\ Q}{\text{Delta s}} \qquad (1.19)$$

where Q is the discharge rate,

and Delta s is the slope of the drawdown curve when plotted on semilogarithmic paper. This equation is on line 338. The slope is calculated by placing the times and drawdowns of the first discharge step into two temporary vectors, and then calling the linear regression procedure to find the best fit slope (line 337).

The main part of this procedure uses the product of drawdown (at time 1 minute, or log time = zero) and discharge rate for each step, 'plotted' against discharge rate to produce coefficients A (y intercept) and C (slope). Rather than relying on one reading at time one minute for each step, all the data of the step are used to produce a best fit straight line (semi-logarithmic), and the one minute drawdown is taken as the y intercept of this line (line 353). The values of s/Q for each step are calculated in line 362, and again procedure linReg is called for the best fit straight line (line 365). Finally, the coefficients of the Well Equation are all calculated on line 366.

Called by: the main part of procedure MENUTHREE.

Calls: procedures LinReg, GetDdAndRate, RunTrial and functions Response, Log, and ReadReal.

Sternberg procedure Line 380

Purpose: to calculate the coefficients of the Well Equation by use of the modified Sternberg analysis.

This major procedure (consisting of program lines 380 to 726) contains a number of sub-procedures; the main part of procedure Sternberg begins at line 671. The first operations are several checks on the suitability of the data. Lines 676 to 686 test for a continuous sequence of increasing times, lines 687 to 702 test for the presence of initial conditions (drawdown and discharge rate at time zero).

Assuming that the tests find the data acceptable, lines 704 to 718 calculate the values for the vector StrnVec using Equation 1.5 (see the main text). An intermediate step in the calculation of the elements for this vector is the calculation of the vector DeltaQ which contains the changes in the discharge rate (line 710).

From line 719, the remainder of the modified Sternberg analysis is called via procedure StrnMenu.

Called by: procedure MENUTHREE.

Calls: procedure StrnMenu.

StrnMenu subprocedure Line 642

Purpose: to allow the user to display or print the values of the vectors used for the modified Sternberg analysis, or to move on to the later part of the analysis.

Called by: parent procedure Sternberg.

Calls: subprocedures PrintSternberg, DisplayData, and SolvStrn.

7. KEY LINES OF PROGRAM DTDHA

7.1. File DTDHA.PAS, key lines

```
Line      Text
  6 {#} {$I FIRST.SEG}
  8 {# Declaration of variables specially for program DTDHA}
 36 Function FileExist {Test for the existence of a given file}
 48 {#} {$I READSAVE.PRC}
 50 Function Log {Log to base ten}
 56 Procedure FirstLast; {Get first and last record numbers for an operation}
 87 Procedure LinReg {Linear regression}
105 {#} {$I DTDHMEN2.SEG}
107 {#} {$I DTDHMEN3.SEG}
109 Procedure PrintData; {Paper copy of discharge test data}
149 {#----- Beginning of section for entry of data via keyboard -----#}
150 Procedure ReadDrawdown; {Reads a real number, rejects input errors}
175 Procedure DisplayRecentData {Display last few readings entered}
206 {#----- Beginning of major procedure EnterTimeDate -----#}
207 Procedure EnterTimeDate; {Entry of time and date and related operations}
217   function ReadIntF {Read an integer, check for + sign}
239   procedure Display {Display a Short string at current cursor position}
249   procedure EnterDate {Entry and checking of time and date}
305   function TestDate {Test that entered date is a valid date}
341   function GetMinutes {Convert time and date to minutes}
362 begin  {# main part of EnterTimeDate}
422 {#----- End of major procedure EnterTimeDate -----#}
424 Procedure EnterMinutes; {Enter a time in minutes}
456 Procedure EnterWtd; {Control of entry of discharge (or well) test data}
482 {#----- End of section for data entry via keyboard -----#}
484 Procedure SaveFile; {Control of disk save of discharge test data}
500 Procedure ReadFile; {Control of disk read of discharge test data}
517 Procedure DispMenu; {Display main menu of program DTDHA}
555 begin {# Controlling part of program DTDHA}
```

7.2. Include file DTDHMEN2.SEG, key lines

```
Line      Text
   2 {#} OVERLAY PROCEDURE MENUTWO;
   7 Procedure CorrectForBackground; {Correct drawdowns for 'noise'}
  12   function CalcCorrection {for a given reading time}
  32 begin {# main part of procedure CorrectForBackground}
 138 Procedure AddConst; {Option to add a constant to readings}
 173 Procedure MultByConst; {Option to multiply readings by a constant}
 226 Procedure DeleteReading; {Option to delete a given reading}
 261 Procedure DeleteReadings; {Option to delete a given group of readings}
 287 Procedure SimulateRec {Simulate recovery for each discharge stage}
```

```
 299   procedure LinRegCall; {Linear regression call}
 382 Procedure SortForTime {Sort into order of increasing times}
 424 Function MarkRec {Find and mark the first recovery stage}
 486 Procedure TOverTOne; {Convert recovery times to t/t1}
 503 Procedure RTMinusRTOne; {Convert recovery times to root t minus root t1}
  547  Procedure  ChangeDescription; {Alter data describing discharge test and
          well}
 571 Procedure DispMenu2; {Display Menu number 2, editing menu}
 593 BEGIN {# main part of overlay procedure DTDHMEN2}
 622 {# End of include file DTDHMEN2.SEG}
```

7.3. Include file DTDHMEN3.SEG, key lines

```
Line        Text
  3 {#} OVERLAY PROCEDURE MENUTHREE;
  8 Procedure GetDdAndRate {Find the drawdown and rate at a given time}
 62 Procedure RunTrial {Produce trial drawdowns from calculated coefficients}
 87 {#----- Major procedure Rorabaugh -----#}
 88 Procedure Rorabaugh; {Rorabaugh's analysis of step test data}
 99   procedure EnterData; {for manual entry}
109         function DataValid {Check data is valid for calculation of
          transmissivity}
137 begin  {# main part of Procedure Rorabaugh}
274 {#----- End of major procedure Rorabaugh -----#}
276 Procedure SQvsQ;  {Evaluation of Well Equation by the s/Q vs. Q method}
379 {#----- Begin major procedure Sternberg -----#}
380 Procedure Sternberg; {Sternberg's analysis of step test data}
428   procedure PrintSternberg; {Print the Sternberg vectors on paper}
464   procedure DisplayData; {Display the Sternberg vectors}
491   procedure SolvStrn; {Solution part of procedure}
642   procedure StrnMenu; {Sternberg menu}
646     procedure DispMenu; {Display Sternberg menu}
671 begin  {# main part of Procedure Sternberg}
726 {#----- End major procedure Sternberg -----#}
728 procedure DispMenu3; {Display of menu 3, analysis menu}
748 BEGIN {# main part of overlay procedure MENUTHREE}
766 {# End of include file DTDHMEN3.SEG}
```

8. DTDHA PROGRAM LISTING

8.1. File DTDHA.PAS, listing

```
Program DTDHA_PAS; {Discharge Test Data Handling and analysis
  of well test data.}

{$R+}

{#} {$I FIRST.SEG}

{# Declaration of variables specially for program DTDHA}
type
  PointerVec=array[1..20] of integer;
  PointerVec2=array[1..20] of real;
  SmallVec=array[1..100] of real;
  VecRec=record
    TimeVec, DdVec, RateVec: MainVec;
  end;
  TimeUnits=(Minutes, Days);
var
  DataSaved, NewRate, Finished: boolean;
  Ch: char;
  Last: integer;
  Slope, YIntercept: real;
  NumData, NumOfDdLogs: array[1..2] of integer;
  Distance: array[1..2] of real;
  FirstDatum, LastDatum: PointerVec;
  TimeStartStep, TimeEndStep: PointerVec2;
  ProgFileName: ShortString;
  TestType: array[1..2] of Test; WellType: array[1..2] of Well;
  TimeUnit: TimeUnits;
  VRP: array[1..2] of ^VecRec;
const
  ValidResponse: set of char=['1','2','3','R','S','V','P','E'];
  SpaceLine='
  MinDayLog=3.1583625;
  LnTen=2.302585093;

Function FileExist {Test for the existence of a given file}
(FileName: ShortString): boolean;
var ThisFile: File of byte;
begin
  Assign(ThisFile,FileName);
  {$I-}
  Reset(ThisFile);
  {$I+}
  FileExist:=(IOResult=0);
  close(ThisFile);
end; {Function FileExist}

{#} {$I READSAVE.PRC}

Function Log {Log to base ten}
(Num: real): real;
begin
  Log:=Ln(Num)/LnTen;
end; {Function Log}

Procedure FirstLast; {Get first and last record numbers for an
operation}
```

```
var
  valid: boolean;
  TempIntOne, TempIntTwo: integer;
begin
  writeln;
  write('   Please enter the first, and then the last, reading number
');
  writeln('of interest.');
  write('The defaults are the first and last readings of the current ');
  writeln('file; ie.');
  write('if you press Enter without having entered any numbers, all ');
  writeln('the data will');
  writeln('be used.'); writeln;
  Valid:=false; First:=1; Last:=NumData[1];
  while not Valid do
  begin
    write('Enter the number of the first reading for this operation ');
    TempIntOne:=ReadIntInput(1);
    write('Enter the number of the last reading for this operation ');
    TempIntTwo:=ReadIntInput(2);
    if (TempIntOne<0) or ((TempIntOne>TempIntTwo) and (TempIntTwo<>0))
      or (TempIntTwo>NumData[1])
      then Valid:=false else Valid:=true;
    if Valid then
    begin
      if TempIntOne<>0 then First:=TempIntOne;
      if TempIntTwo<>0 then Last:=TempIntTwo;
    end; {if}
  end; {while}
end; {Procedure FirstLast}

Procedure LinReg {Linear regression}
(Vec1, Vec2: Smallvec; TempInt: integer);
var
  I: byte;
  Sum1, Sum2, Sum3, Sum4: real;
begin
  Sum1:=0; Sum2:=0; Sum3:=0; Sum4:=0;
  for I:=1 to TempInt do
  begin
    Sum1:=Sum1+Vec1[I]*Vec2[I];
    Sum2:=Sum2+Vec1[I];
    Sum3:=Sum3+Vec2[I];
    Sum4:=Sum4+Sqr(Vec1[I]);
  end;
  Slope:=(TempInt*Sum1-Sum2*Sum3)/(TempInt*Sum4-Sqr(Sum2));
  YIntercept:=(Sum3*Sum4-Sum2*Sum1)/(TempInt*Sum4-Sqr(Sum2));
end; {Procedure LinReg}

{#} {$I DTDHMEN2.SEG}

{#} {$I DTDHMEN3.SEG}

Procedure PrintData; {Paper copy of discharge test data}
var
  I, J, LineCount: integer;
const
  StringDash='-------------------------------------------';
  Heading=
```

```
115     'No.       min.     days     Drawdown  Discharge rate';
116 begin
117   ClrScr;
118   write('Print data in memory on paper. ');
119   if CapOptions('Continue or Exit?')=1 then
120   begin
121     writeln('The printer should be switched on and at top of form.');
122     writeln(lst,'        File name, ',FileName);
123     writeln(lst,'      DTDHA ver. 2.0:  output data');
124     writeln(lst);
125     writeln(lst,'       ',Heading); LineCount:=4;
126     for I:=1 to NumData[1] do
127     begin
128       writeln(lst,'      ',I:3,'  ',VRP[1]^.TimeVec[I]:9:1,'  '
129         ,VRP[1]^.TimeVec[I]/1440:9:4,' ',VRP[1]^.DdVec[I]:9:3,'  '
130         ,VRP[1]^.RateVec[I]:9:1);
131       LineCount:=LineCount+1;
132       if LineCount=55 then
133       begin
134         for J:=1 to 12 do writeln(lst);
135         writeln(lst,'      ',Heading);
136         LineCount:=2
137       end
138       else begin
139         if (LineCount div 7)*7=LineCount then
140         begin
141           writeln(lst,'       ',StringDash);
142           LineCount:=LineCount+1;
143         end;
144       end; {if-then-else}
145     end; {for I}
146   end; {if really required}
147 end; {Procedure PrintData}
148
149 {#----- Beginning of section for entry of data via keyboard -----#}
150 Procedure ReadDrawdown; {Reads a real number, rejects input errors}
151 var
152   I, J, X, Y: byte;
153   Result: integer;
154 begin
155   X:=WhereX; Y:=WhereY; Result:=1;
156   while Result<>0 do
157   begin
158     read(Short);
159     J:=pos('+',Short);
160     if J<>0 then
161     begin
162       NewRate:=true;
163       Delete(Short,J,1);
164     end
165     else NewRate:=false;
166     Val(Short,Num,Result);
167     if Result<>0 then
168     begin
169       GotoXY(X,Y); write('Invalid'); Delay(1000);
170       GotoXY(X,Y); write('       '); GotoXY(X,Y);
171     end;
172   end;
173 end; {Procedure ReadDrawdown}
```

```
174
175 Procedure DisplayRecentData {Display last few readings entered}
176 (I: integer);
177 begin
178   GotoXY(1,18);
179   writeln('Record #    Time (min.)    Drawdown (m)    Rate (m*m*m/d)');
180   GotoXY(1,19);
181   if I>3 then
182   begin
183     write(I-3:3,'        ',VRP[1]^.TimeVec[I-3]:10:2,'     '
184     ,VRP[1]^.DdVec[I-3]:8:4);
185     write('          ',VRP[1]^.RateVec[I-3]:8:4);
186   end
187   else write(SpaceLine);
188   GotoXY(1,20);
189   if I>2 then
190   begin
191     write(I-2:3,'        ',VRP[1]^.TimeVec[I-2]:10:2,'     '
192     ,VRP[1]^.DdVec[I-2]:8:4);
193     write('          ',VRP[1]^.RateVec[I-2]:8:4);
194   end
195   else write(SpaceLine);
196   GotoXY(1,21);
197   if I>1 then
198   begin
199     write(I-1:3,'        ',VRP[1]^.TimeVec[I-1]:10:2,'     '
200     ,VRP[1]^.DdVec[I-1]:8:4);
201     write('          ',VRP[1]^.RateVec[I-1]:8:4);
202   end
203   else write(SpaceLine);
204 end; {Procedure DisplayRecentData}
205
206 {#----- Beginning of major procedure EnterTimeDate -----#}
207 Procedure EnterTimeDate; {Entry of time and date and related operations}
208 var
209   Finished, SecondTime, GoOn, GoBack, Valid: boolean;
210   I, Min1, Min2, Hr1, Hr2, Day1, Day2, Mth1, Mth2, Yr1, Yr2: integer;
211   DefaultM, DefaultY: integer;
212   First, Second, Minutes: real;
213 const
214   SpaceString=
215   '                                                                    ';
216
217   function ReadIntF {Read an integer, check for + sign}
218   : integer;
219   var
220     I, J, X, Y: byte;
221     Num, Result: integer;
222   begin
223     X:=WhereX; Y:=WhereY; Result:=1; GoOn:=false;
224     while Result<>0 do
225     begin
226       read(Short);
227       if pos('+',Short)<>0 then
228       begin GoOn:=true; I:=pos('+',Short); delete(Short,I,1); end;
229       Val(Short,Num,Result);
230       if Result<>0 then
231       begin
232         GotoXY(X,Y); write('Invalid'); Delay(1000);
```

```
233         GotoXY(X,Y); write('       '); GotoXY(X,Y);
234       end;
235     end;
236     ReadIntF:=Num;
237   end; {sub function ReadIntF}
238
239   procedure Display {Display a Short string at current cursor position}
240   (Short: ShortString);
241   var
242     X, Y: integer;
243   begin
244     X:=WhereX; Y:=WhereY;
245     GotoXY(X,Y); write(Short); Delay(1000);
246     GotoXY(X,Y); write('                    '); GotoXY(X,Y);
247   end; {sub procedure Display}
248
249   procedure EnterDate {Entry and checking of time and date}
250   (var Min, Hr, Day, Mth, Yr: integer);
251   var
252     MinValid: boolean;
253   begin
254     GotoXY(1,16);
255     writeln(SpaceString);
256     repeat
257       GotoXY(3,16); MinValid:=true; Min:=ReadInt(0);
258       if (Min=-1) and (I>1) then GoBack:=true else GoBack:=false;
259       if ((I=1) and ((Min=-1) or (Min<-2) or (Min>59))) or
260       ((I>1) and ((Min<-2) or (Min>59))) then
261       begin
262         Display('Invalid!'); MinValid:=false;
263       end;
264     until MinValid;
265     if Min<-1 then Finished:=true;
266     if (not Finished) and (not GoBack) then
267     begin
268       repeat
269         GotoXY(13,16); Hr:=ReadInt(0);
270         if (Hr<0) or (Hr>23) then
271         begin
272           Display('Invalid!');
273         end;
274       until (Hr>-1) and (Hr<24);
275       if SecondTime then
276       begin
277         GotoXY(1,7);
278         writeln('Suffix Day with ''+'' to go on to entry of Month.');
279       end;
280       GotoXY(23,16); Day:=ReadIntF;
281       GotoXY(1,7);
282       writeln(SpaceString);
283       GotoXY(23,16);
284       if GoOn or not SecondTime then
285       begin
286         if SecondTime then
287         begin
288           GotoXY(1,7);
289           writeln('Suffix Month with ''+'' to go on to entry of Year.');
290         end;
291         GotoXY(33,16); Mth:=ReadIntF;
```

```
292         GotoXY(1,7);
293         writeln(SpaceString);
294         GotoXY(33,16);
295         if GoOn or not SecondTime then
296         begin
297           GotoXY(43,16); Yr:=ReadInt(0);
298         end {second GoOn}
299         else Yr:=0;
300       end {first GoOn}
301       else begin Mth:=0; Yr:=0; end; {first GoOn}
302     end; {if not Finished}
303   end; {sub procedure EnterData}
304
305   function TestDate {Test that entered date is a valid date}
306   (Day, Mth, Yr: integer): boolean;
307   var
308     Valid: boolean;
309   begin
310     Valid:=true;
311     if Mth in [1,3,5,7,8,10,12] then
312     begin
313       if (Day>31) or (Day<1) then
314       begin Display('Day is invalid!'); Valid:=false end;
315     end;
316     if Mth in [4,6,9,11] then
317     begin
318       if (Day>30) or (Day<1) then
319       begin Display('Day is invalid!'); Valid:=false end;
320     end;
321     if Mth=2 then
322     begin
323       if ((Yr div 4)*4<>Yr) or
324          (((Yr div 100)*100=Yr) and not ((Yr div 400)*400=Yr)) then
325       begin
326         if (Day>28) or (Day<1) then
327         begin Display('Day is invalid!'); Valid:=false end;
328       end
329       else begin
330         if (Day>29) or (Day<1) then
331         begin Display('Day is invalid!'); Valid:=false end;
332       end; {if-then-else}
333     end; {Month of Feb.}
334     if (Mth<1) or (Mth>12) then
335     begin Display('Month is invalid!'); Valid:=false end;
336     if (Yr<1) or (Yr>3000) then
337     begin Display('Year is invalid!'); Valid:=false end;
338     TestDate:=Valid;
339   end; {sub function TestDate}
340
341   function GetMinutes {Convert time and date to minutes}
342   (Min, Hr, Day, Mth, Yr: integer): real;
343   var
344     TempReal: real;
345   const
346     Md=1440.;
347     CumDays: array[1..12] of real =
348       (0,31,59,90,120,151,181,212,243,273,304,334);
349   begin
```

```
350     TempReal:=Min+Hr*60.+CumDays[Mth]*Md+Yr*525600.+
351       (Yr div 4)*Md+Day*Md+Md-(Yr div 100)*Md+(Yr div 400)*Md;
352     if ((Yr div 4)*4<>Yr) or
353        (((Yr div 100)*100=Yr) and not ((Yr div 400)*400=Yr))
354     then GetMinutes:=TempReal
355     else begin
356       if Mth<3
357         then GetMinutes:=TempReal-Md
358         else GetMinutes:=TempReal;
359     end; {if-then-else}
360   end; {function GetMinutes}
361
362 begin  {# main part of EnterTimeDate}
363   I:=NumData[1]+1;
364   DisplayRecentData(I);
365   SecondTime:=false; TextColor(green); Finished:=false;
366   GotoXY(2,15);
367   writeln('Minute    Hour      Day       Month     Year '
368     ,'  Drawdown (m)  Rate (m^3/d)');
369   GotoXY(1,10);
370   writeln('Enter the starting time.');
371   repeat
372     EnterDate(Min1, Hr1, Day1, Mth1, Yr1);
373     Valid:=TestDate(Day1, Mth1, Yr1);
374   until Valid;
375   DefaultM:=Mth1; DefaultY:=Yr1;
376   GotoXY(1,10);
377   writeln('Starting time:  ',Min1:2,' min, ',Hr1:2,' hrs., ',Day1:2,
378     ' day, ',Mth1:2,' month, ',Yr1:4,' yr.');
379   GotoXY(1,13);
380   writeln('Enter the time of the next reading.');
381   First:=GetMinutes(Min1, Hr1, Day1, Mth1, Yr1);
382   SecondTime:=true;
383   repeat
384     repeat
385       repeat
386         EnterDate(Min2, Hr2, Day2, Mth2, Yr2);
387         if GoBack then I:=I-1;
388         if GoBack then DisplayRecentData(I);
389       until not GoBack;
390       if not Finished then
391       begin
392         if Mth2=0 then Mth2:=DefaultM;
393         if Yr2=0 then Yr2:=DefaultY;
394         DefaultM:=Mth2; DefaultY:=Yr2;
395         Valid:=TestDate(Day2, Mth2, Yr2);
396       end;
397     until Valid or Finished;
398     if not Finished then
399     begin
400       Second:=GetMinutes(Min2, Hr2, Day2, Mth2, Yr2);
401       VRP[1]^.TimeVec[I]:=Second-First;
402       if I>1 then
403       begin
404         GotoXY(1,7);
405         writeln('Suffix Drawdown with ''+'' to enter a new rate.');
406       end;
407       GotoXY(53,16); ReadDrawdown; VRP[1]^.DdVec[I]:=Num;
408       if NewRate or (I=1) then
```

```
409       begin
410         GotoXY(63,16); VRP[1]^.RateVec[I]:=ReadReal(0);
411       end
412       else VRP[1]^.RateVec[I]:=VRP[1]^.RateVec[I-1];
413       GotoXY(1,11);
414       writeln('Previous entry: ',Min2:2,' min, ',Hr2:2,' hrs., ',Day2:2,
415         ' day, ',Mth2:2,' month, ',Yr2:4,' yr.');
416     end;
417     If not Finished then I:=I+1;
418     DisplayRecentData(I);
419   until Finished;
420   NumData[1]:=I-1; DataSaved:=false;
421 end; {Procedure EnterTimeDate}
422 {#----- End of major procedure EnterTimeDate -----#}
423
424 Procedure EnterMinutes; {Enter a time in minutes}
425 var
426   I: integer;
427 begin
428   I:=NumData[1]+1;
429   repeat
430     if I<>1 then
431     begin
432       GotoXY(1,15);
433       write('   Suffix the drawdown figure with a ''+'' if you ');
434       writeln('want to enter a');
435       writeln('different discharge rate.');
436     end;
437     DisplayRecentData(I);
438     GotoXY(1,22); write(SpaceLine); GotoXY(1,22);
439     write(I:3); GotoXY(16,22); VRP[1]^.TimeVec[I]:=ReadReal(0);
440     if (VRP[1]^.TimeVec[I]<>-1) and (VRP[1]^.TimeVec[I]<>-2)
441     then begin
442       GotoXY(29,22); ReadDrawdown; VRP[1]^.DdVec[I]:=Num;
443       if NewRate or (I=1) then
444       begin
445         GotoXY(45,22); VRP[1]^.RateVec[I]:=ReadReal(0);
446       end
447       else VRP[1]^.RateVec[I]:=VRP[1]^.RateVec[I-1];
448     end;
449     if VRP[1]^.TimeVec[I]=-2 then Finished:=true;
450     if (VRP[1]^.TimeVec[I]=-1) and (I>1) then I:=I-2;
451     if not Finished then I:=I+1;
452   until Finished;
453   NumData[1]:=I-1; DataSaved:=false;
454 end; {Procedure EnterMinutes}
455
456 Procedure EnterWtd; {Control of entry of discharge (or well) test data}
457 var
458   Answer: integer;
459 begin
460   ClrScr; Finished:=false;
461   if NumData[1]=0 then
462   begin
463     writeln('Please enter data as indicated by the prompts below;');
464     writeln;
465     writeln('What type of test,');
466     TestType[1]:=Test(CapOptions('Discharge, Recovery, Simulation?')-1);
467     write('What type of well, ');
```

```
468     WellType[1]:=Well(CapOptions('Pumped or Observation')-1);
469     if WellType[1]=Observation then
470       write('Distance from pumped well to observation well? ')
471       else write('Effective radius of discharge well? ');
472     Distance[1]:=ReadReal(1);
473     ClrScr;
474   end; {if NumData[1]=0}
475   writeln('   Data entry.  Enter a time of -1 to back up to the last
    entry');
476   writeln('if you made a mistake and wish to correct it.');
477   writeln('Enter minutes = -2 to end data entry.');
478   write('Do you want to enter times as ');
479   Answer:=CapOptions('Minutes or Time and date?');
480   if Answer=1 then EnterMinutes else EnterTimeDate;
481 end; {Procedure EnterWtd}
482 {#----- End of section for data entry via keyboard -----#}
483
484 Procedure SaveFile; {Control of disk save of discharge test data}
485 begin
486   if NumData[1]<>0
487   then begin
488     ClrScr;
489     FirstLast;
490     SaveData(VRP[1]^.TimeVec,VRP[1]^.DdVec,
491       VRP[1]^.RateVec,TestType[1],WellType[1],Distance[1],First,Last);
492     if (first=1) and (last=NumData[1]) then DataSaved:=true;
493   end
494   else begin
495     writeln('No data to save!');
496     delay(2000);
497   end; {if-then-else}
498 end; {Procedure SaveFile}
499
500 Procedure ReadFile; {Control of disk read of discharge test data}
501 var
502   Answer: integer;
503 begin
504   Answer:=0;
505   if NumData[1]<>0 then
506   begin
507     write('  Proceeding will destroy the data in memory.  ');
508     Answer:=CapOptions('Continue or Exit? ');
509   end;
510   if (NumData[1]=0) or (Answer=1) then
511   ReadTestDataFile(VRP[1]^.TimeVec,VRP[1]^.DdVec,
512     VRP[1]^.RateVec,TestType[1],WellType[1],
513     Distance[1],NumData[1]);
514   DataSaved:=true;
515 end; {Procedure ReadFile}
516
517 Procedure DispMenu; {Display main menu of program DTDHA}
518 begin
519   ClrScr; TextColor(Green);
520   writeln('        DTDHA Version 2.0:   Main Menu');
521   writeln;
522   if NumData[1]=0
523   then writeln('There are no data at present in memory.')
524   else begin
525     writeln('Currently ',NumData[1]:3,' readings in memory.');
```

```
526     if FileName<>''
527       then writeln('Last file name used was ',FileName)
528       else writeln('There is no current file name.');
529     case TestType[1] of
530       Discharge:  write('Discharge test:  ');
531       Recovery:   write('Recovery test:  ');
532       Simulation: write('Simulated test data:  ');
533     end; {of cases}
534     if WellType[1]=Pumped then write('Data from the pumped well:  ')
535       else write('Data from a piezometer:  ');
536     writeln('R = ',Distance[1]:6:2);
537   end; {if-then-else}
538   if not DataSaved then
539     writeln('Data have not been saved to disk file since alteration.');
540   writeln;
541   writeln('  Which option?');
542   writeln('Press the indicated number or letter key.');
543   writeln;
544   writeln('1: Enter discharge test data via the key board;');
545   writeln('2: Edit the data in memory (Menu Number 2);');
546   writeln('3: Solve the well equation (Menu Number 3);');
547   writeln('R: Read a file from disk;');
548   writeln('S: Save the data in memory to a disk file;');
549   writeln('V: View the data in memory;');
550   writeln('P: Print the data in memory;');
551   writeln;
552   writeln('E: End DTDHA and return to primary menu.');
553 end; {Procedure DispMenu}
554
555 begin {# Controlling part of program DTDHA}
556   new(VRP[1]); new(VRP[2]);
557   NumData[1]:=0; FileName:=''; DataSaved:=true;
558   DispMenu;
559   repeat
560     repeat
561       Ch:='x'; repeat read(kbd,Ch) until Ch<>'x'; Ch:=UpCase(Ch);
562     until Ch in ValidResponse;
563     case UpCase(Ch) of
564       '1': EnterWtd;
565       '2': MenuTwo;
566       '3': MenuThree;
567       'R': ReadFile;
568       'S': SaveFile;
569       'V': begin
570              if NumData[1]<>0
571              then begin
572                ClrScr;
573                ViewReadings(VRP[1]^.TimeVec, VRP[1]^.DdVec,
574                  VRP[1]^.RateVec,NumData[1]);
575              end
576              else begin
577                writeln('No data in memory.');
578                delay(2000);
579              end; {if-then-else}
580            end;
581       'P': begin
582              if NumData[1]<>0 then PrintData
583              else begin
584                writeln('No data in memory.');
```

```
585                delay(2000);
586              end; {else}
587            end;
588      end; {of cases}
589      if Ch<>'E' then DispMenu;
590      if (Ch='E') and (not DataSaved) then
591      begin
592        write('You want to exit DTDHA without saving the data? ');
593        if Response('YN')='N' then Ch:='x';
594      end;
595    until Ch='E';
596    dispose(VRP[1]); dispose(VRP[2]);
597    {The code below should only be used when this program is used as a
598     chain file.}
599    ChainTo('GWMENU.CHN',IOCode);
600    if IOCode<>0 then
601    writeln('Unable to chain to program GWMENU.CHN!');
602 end.
```

8.2. Include file DTDHMEN2.SEG, listing

```
 1 {Include file DTDHMEN2.SEG}
 2 {#} OVERLAY PROCEDURE MENUTWO;
 3 const
 4   ValidResponse:
 5     set of char=['1','2','3','4','5','6','7','A','B','C','M','S','R'];
 6
 7 Procedure CorrectForBackground; {Correct drawdowns for 'noise'}
 8 var
 9   OutOfSeq, EndData: boolean;
10   CorFac, OldTime: real;
11
12   function CalcCorrection {for a given reading time}
13   (Elapmin: real): real;
14   var
15     I: integer;
16     Slope, Correction, Drawdn: real;
17   begin
18     I:=1;
19     repeat
20       I:=I+1  {I points at second background reading to use for
    correction}
21     until (VRP[2]^.TimeVec[I]>ElapMin) or (I=NumData[2]);
22     if (I=NumData[2]) and (VRP[2]^.TimeVec[I]<ElapMin) then EndData:=true
23     else begin
24       Slope:=(VRP[2]^.DdVec[I]-VRP[2]^.DdVec[I-1])/
25         (VRP[2]^.TimeVec[I]-VRP[2]^.TimeVec[I-1]);
26       Correction:=CorFac*(VRP[2]^.DdVec[I-1]+Slope*
27         (ElapMin-VRP[2]^.TimeVec[I-1]));
28     end;
29     if not EndData then CalcCorrection:=-Correction;
30   end; {sub function CalcCorrection}
31
32 begin {# main part of procedure CorrectForBackground}
33   EndData:=false;
34   ClrScr;
35   writeln('  This option corrects discharge test data for '
36     ,'background ''noise'' as');
37   writeln('recorded in an observation well in the same aquifer at a'
38     ,' great distance from');
```

```
39   writeln('the discharging well.');
40   writeln('   The data to be corrected are expected to be in memory, and'
41     ,' the background');
42   writeln('data must be loaded from another file.');
43   writeln;
44   write('Do you want to ');
45   if CapOptions('Continue or Exit? ')=1 then
46   begin
47     writeln('The file holding the background data must be specified.');
48     ReadTestDataFile(VRP[2]^.TimeVec, VRP[2]^.DdVec, VRP[2]^.RateVec,
49     TestType[2], WellType[2], Distance[2], NumData[2]);
50     writeln('NumData 1, 2, =',NumData[1]:4, NumData[2]:4);
51     write('What correction factor (eg. 1, if both wells have equal bar.'
52       ,' susept.) ');
53     readln(CorFac);
54     OldTime:=VRP[1]^.TimeVec[1]; OutOfSeq:=false;
55     for I:=2 to NumData[1] do
56     begin
57       if OldTime>=VRP[1]^.TimeVec[I] then OutOfSeq:=true;
58       OldTime:=VRP[1]^.TimeVec[I];
59     end;
60     if OutOfSeq then
61       writeln('Data in file to be corrected is out of sequence.');
62     if not OutOfSeq then
63     begin
64       OldTime:=VRP[2]^.TimeVec[1]; OutOfSeq:=false;
65       for I:=2 to NumData[2] do
66       begin
67         if OldTime>=VRP[2]^.TimeVec[I] then OutOfSeq:=true;
68         OldTime:=VRP[2]^.TimeVec[I];
69       end;
70       if OutOfSeq then
71         writeln('Data in background file is out of sequence.');
72     end;
73     if not OutOfSeq then
74     begin
75       I:=0;
76       while (I<NumData[1]) and not EndData do
77       begin
78         I:=I+1;
79         VRP[1]^.DdVec[I]:=
80           VRP[1]^.DdVec[I]+CalcCorrection(VRP[1]^.TimeVec[I]);
81       end;
82       if EndData then writeln('Can''t finish; end of background data!');
83       if EndData then NumData[1]:=I-1 else NumData[1]:=I;
84     end; {if not OutOfSeq}
85     if EndData or OutOfSeq then
86     begin
87       writeln('Press any key to continue.');
88       Ch:='x'; repeat read(kbd, Ch) until Ch<>'x';
89     end; {if error}
90   end; {if CapOptions}
91 end; {Procedure CorrectForBackground}
92
93 Function CheckForStepData {Count steps, mark beginning and end of each}
94 (var NumOfSteps: integer): boolean;
95 var
96   Valid: boolean;
97   I, J: integer;
```

```
 98 begin
 99   Valid:=true;
100   writeln('Checking that data is valid......');
101   I:=1; FirstDatum[1]:=1; LastDatum[1]:=1; J:=1;
102   if VRP[1]^.RateVec[1]=0 then
103     begin J:=2; FirstDatum[1]:=2; LastDatum[1]:=2; end;
104   repeat
105     while (VRP[1]^.RateVec[J]=VRP[1]^.RateVec[FirstDatum[I]])
106       and (J<=NumData[1]) do
107       begin J:=J+1; LastDatum[I]:=J-1; end;
108     if J<NumData[1] then
109     begin
110       if LastDatum[I]-FirstDatum[I]<2 then
111       begin
112         Valid:=false;
113         writeln('Too few data in step ',I:3);
114         I:=I+1; Delay(2000);
115       end; {if}
116       if Valid then
117       begin I:=I+1; FirstDatum[I]:=J; LastDatum[I]:=J; end;
118     end; {if J<=NumData}
119   until (J>=NumData[1]) or (not Valid);
120   if (I=1) and Valid then
121   begin
122     writeln('Single step only!'); Valid:=false; Delay(2000);
123   end;
124   if Valid then
125   begin
126     writeln('Step #   First rec.   Last rec.');
127     NumOfSteps:=I;
128     for I:=1 to NumOfSteps do
129     begin
130       writeln(I:4,'       ',FirstDatum[I]:3,'           ',
    LastDatum[I]:3);
131       TimeStartStep[I]:=VRP[1]^.TimeVec[FirstDatum[I]];
132       TimeEndStep[I]:=VRP[1]^.TimeVec[LastDatum[I]];
133     end;
134   end; {if Valid}
135   CheckForStepData:=Valid;
136 end; {Function CheckForStepData}
137
138 Procedure AddConst; {Option to add a constant to readings}
139 var
140   ParNum, I: integer;
141   Constant: real;
142 begin
143   ClrScr;
144   writeln('        Add a constant to all, or some readings');
145   write('If you want to leave the data as it is, then add zero ');
146   writeln('to all readings.');
147   writeln;
148   FirstLast; {Get the first and last reading numbers}
149   writeln;
150   write('Do you want to change ');
151   ParNum:=CapOptions('Times, Drawdowns, or discharge Rates?');
152   case ParNum of
153     1: begin
154       write('Enter constant to add to specified times ');
155       Constant:=ReadReal(2);
```

```
156       for I:=First to Last do VRP[1]^.TimeVec[I]:=
157         VRP[1]^.TimeVec[I]+Constant;
158     end;
159     2: begin
160       write('Enter constant to add to specified drawdowns ');
161       Constant:=ReadReal(2);
162       for I:=First to Last do VRP[1]^.DdVec[I]:=
    VRP[1]^.DdVec[I]+Constant;
163     end;
164     3: begin
165             write('Enter constant to add to specified discharge rates ');
166       Constant:=ReadReal(2);
167       for I:=First to Last do VRP[1]^.RateVec[I]:=
168         VRP[1]^.RateVec[I]+Constant;
169     end;
170   end; {of cases}
171 end; {Procedure AddConst}
172
173 Procedure MultByConst; {Option to multiply readings by a constant}
174 var
175   ParNum, I: integer;
176   Constant: real;
177 begin
178   ClrScr;
179   writeln('        Multiply all or some readings by a constant.');
180   write('If you want to leave the data as it is, then multiply all ');
181   writeln('readings by 1.');
182   writeln;
183   FirstLast; {Get the first and last reading numbers}
184   writeln;
185   write('Do you want to change ');
186   ParNum:=CapOptions('Times, Drawdowns, or discharge Rates?');
187   case ParNum of
188     1: begin
189       writeln;
190       writeln('   Some conversion factors');
191       writeln('Min. to days, multiply by 0.0006944; inverse 1440');
192       writeln('Min. to hours, by 0.01666');
193       writeln;
194       write('Enter constant to multiply specified times by ');
195       Constant:=ReadReal(2);
196       for I:=First to Last do
197         VRP[1]^.TimeVec[I]:=VRP[1]^.TimeVec[I]*Constant;
198     end;
199     2: begin
200       writeln;
201       writeln('   Some conversion factors');
202       writeln('Feet to metres, multiply by 0.3048; inverse 3.281');
203       writeln('kPa to metres head water, mult. by 0.102; inverse 9.8');
204       writeln('For angle hole conversion, multiply readings by
    Sin(angle)');
205       writeln('where angle is measured from the horizontal.');
206       writeln;
207       write('Enter constant to multiply specified drawdowns by ');
208       Constant:=ReadReal(2);
209       for I:=First to Last do VRP[1]^.DdVec[I]:=
    VRP[1]^.DdVec[I]*Constant;
210     end;
```

```
211     3: begin
212       writeln;
213       writeln('   Some conversion factors');
214       writeln('Imp. gall/hr to m^3/day, multiply by 0.1091; inverse
    9.166');
215       writeln('US gall/hour to m^3/day, by 0.09084; inverse 11.008');
216       writeln('l/sec. to m^3/day, by 86.4; inverse 0.01157');
217       writeln;
218       write('Enter constant to multiply specified discharge rates by ');
219       Constant:=ReadReal(2);
220       for I:=First to Last do VRP[1]^.RateVec[I]:=
221         VRP[1]^.RateVec[I]*Constant;
222     end;
223   end; {of cases}
224 end; {Procedure MultByConst}
225
226 Procedure DeleteReading; {Option to delete a given reading}
227 var
228   Valid: boolean;
229   TempInt, I: integer;
230 begin
231   ClrScr;
232   writeln('        Delete a specified reading');
233   writeln;
234     write('You must specify the reading to be deleted by it''s reading ');
235   writeln('number.');
236   TempInt:=CapOptions('Continue or Exit?');
237   if TempInt=1 then
238   begin
239     write('Enter the reading (record) number '); TempInt:=ReadInt(2);
240     if (TempInt<1) or (TempInt>NumData[1]) then Valid:=false
241       else Valid:=true;
242     if Valid
243     then begin
244       for I:=TempInt to NumData[1]-1 do
245       begin
246         VRP[1]^.TimeVec[I]:=VRP[1]^.TimeVec[I+1];
247         VRP[1]^.DdVec[I]:=VRP[1]^.DdVec[I+1];
248         VRP[1]^.RateVec[I]:=VRP[1]^.RateVec[I+1];
249       end; {for}
250       write('The record has been deleted, ');
251       writeln('higher numbered records have been renumbered.');
252       NumData[1]:=NumData[1]-1; delay(1500)
253     end {then}
254     else begin
255       writeln('The entered record number is invalid!');
256       delay(2000);
257     end; {if then else}
258   end; {if TempInt}
259 end; {Procedure DeleteReading}
260
261  Procedure  DeleteReadings;  {Option  to delete a given group of readings}
262 var
263   I, TempInt: integer;
264 begin
265   ClrScr;
266   writeln('        Delete a specified group of readings');
267   writeln;
268   write('You must specify the readings to be deleted ');
```

```
269   writeln('by their reading numbers.');
270   TempInt:=CapOptions('Continue or Exit?');
271   if TempInt=1 then
272   begin
273     FirstLast;
274     for I:=First to First+NumData[1]-Last-1 do
275     begin
276       VRP[1]^.TimeVec[I]:=VRP[1]^.TimeVec[I+Last-First+1];
277       VRP[1]^.DdVec[I]:=VRP[1]^.DdVec[I+Last-First+1];
278       VRP[1]^.RateVec[I]:=VRP[1]^.RateVec[I+Last-First+1];
279     end; {for}
280     NumData[1]:=NumData[1]-(Last-First)-1;
281     write('The records have been deleted, ');
282     writeln('higher numbered records have been renumbered.');
283     delay(1500)
284   end; {if TempInt}
285 end; {Procedure DeleteReadings}
286
287 Procedure SimulateRec {Simulate recovery for each discharge stage}
288 (NumData: integer);
289 {Version 3}
290 var
291   StepData: boolean;
292   Ch: char;
293   I, J, K, K1, L, Pointer1, Pointer2, NumOfSteps: integer;
294   Real1, Real2, Real3, ConvTime, AdditDd: real;
295   TempDd: real;  {Drawdown due to previous rate at current time}
296   TempVec1, TempVec2, SlopeOfStep, YIntOfStep: SmallVec;
297   TestDdExt, TestTimeExt, SlopeOfStepExt, YIntOfStepExt: SmallVec;
298
299   procedure LinRegCall; {Linear regression call}
300   begin
301     K1:=0;
302     for  K:=Pointer1 to Pointer2 do
303     begin
304       K1:=K1+1;
305       TempVec1[K1]:=Log(VRP[1]^.TimeVec[K]);
306       TempVec2[K1]:=VRP[1]^.DdVec[K];
307     end;
308     LinReg(TempVec1, TempVec2, K1);
309   end; {procedure LinRegCall}
310
311 begin
312   ClrScr;
313   StepData:=CheckForStepData(NumOfSteps);
314   if StepData then
315   begin    {Calculate trend of first stage}
316     writeln('Data appears to be valid.');
317     writeln('   Note that this is an approximate procedure only.');
318     Pointer1:=FirstDatum[1]; Pointer2:=LastDatum[1]; I:=1;
319     writeln('Step 2');
320     if Pointer2-Pointer1>6
321       then Pointer1:=Pointer2-(Pointer2-Pointer1) div 3;
322     LinRegCall;
323     SlopeOfStep[1]:=Slope; YIntOfStep[1]:=YIntercept;
324                            {Convert data of step 2}
325         for J:=FirstDatum[2] to LastDatum[2] do          {All of step 2}
```

```
326     begin
327       AdditDd:=VRP[1]^.DdVec[J]-(SlopeOfStep[1]*Log(VRP[1]^.TimeVec[J])
328         +YIntOfStep[1]);      {Additional drawdown due to increased rate}
329       ConvTime:=VRP[1]^.TimeVec[J]-TimeEndStep[1];
330       {Time for the corrected time/drawdown data}
331       TempDd:=SlopeOfStep[1]*Log(ConvTime)+YIntOfStep[1];
332       VRP[1]^.DdVec[J]:=AdditDd+TempDd;       {Calculation of new
    drawdown}
333       VRP[1]^.TimeVec[J]:=ConvTime;
334       writeln('Time ',VRP[1]^.TimeVec[J]:10:3,' Drawdown '
335         ,VRP[1]^.DdVec[J]:10:3);
336     end; {for J.  Conversion for step 2 data completed.}
337     for I:=2 to NumOfSteps-1 do
338       begin                          {Extend line of currents steps test
    data}
339         writeln('Step ',I+1:4);
340         Pointer1:=FirstDatum[I]; Pointer2:=LastDatum[I];
341         LinRegCall; SlopeOfStep[I]:=Slope; YIntOfStep[I]:=YIntercept;
342         {Slope and Y int. of the last corrected step is now recorded}
343         L:=1;
344         for J:=FirstDatum[I] to LastDatum[I] do
345         begin
346           TestTimeExt[L]:=VRP[1]^.TimeVec[J]+TimeEndStep[I];
347           Real1:=SlopeOfStep[I-1]*Log(TestTimeExt[L])+YIntOfStep[I-1];
348           Real2:=SlopeOfStep[I]*Log(TimeEndStep[I-1]-TimeStartStep[I-1]
349             +VRP[1]^.TimeVec[J])+YIntOfStep[I];
350           Real3:=SlopeOfStep[I-1]*Log(TimeEndStep[I-1]-TimeStartStep[I-1]
351             +VRP[1]^.TimeVec[J])+YIntOfStep[I-1];
352           TestDdExt[L]:=Real1+Real2-Real3;
353           L:=L+1;
354         end;  {We now have the test time/drawdown line extended}
355         K1:=0;
356         for  K:=1 to L-1 do
357         begin
358           K1:=K1+1;
359           TempVec1[K1]:=Log(TestTimeExt[K]);
360           TempVec2[K1]:=TestDdExt[K];
361         end;
362         LinReg(TempVec1, TempVec2, K1);
363         SlopeOfStepExt[I]:=Slope; YIntOfStepExt[I]:=YIntercept;
364         for J:=FirstDatum[I+1] to LastDatum[I+1] do      {All of step
    I+1}
365         begin
366           AdditDd:=VRP[1]^.DdVec[J]-(SlopeOfStepExt[I]*
367             Log(VRP[1]^.TimeVec[J])+YIntOfStepExt[I]);
368                {Additional drawdown due to increased rate}
369           ConvTime:=VRP[1]^.TimeVec[J]-TimeEndStep[I];
370           {Time for the corrected time/drawdown data}
371           TempDd:=SlopeOfStep[I]*Log(ConvTime)+YIntOfStep[I];
372           VRP[1]^.DdVec[J]:=AdditDd+TempDd;   {Calculation of new
    drawdown}
373           VRP[1]^.TimeVec[J]:=ConvTime;
374           writeln('Time ',VRP[1]^.TimeVec[J]:10:3,' Drawdown '
375             ,VRP[1]^.DdVec[J]:10:3);
376         end; {for J.  Conversion for step I data completed.}
377     end; {for I}
378   end {if StepData}
379   else writeln('Data is not suitable for conversion.');
380 end; {Procedure SimulateRec version 3}
```

```
381
382 Procedure SortForTime {Sort into order of increasing times}
383 (NumData: integer);
384 var
385   Change: boolean;
386   I, J, K, StepSize, Pass: integer;
387   TempReal: real;
388 begin
389   if NumData>1 then
390   begin
391     writeln('Sorting has started.');
392     Pass:=0; StepSize:=NumData; Change:=false;
393     while StepSize>1 do
394     begin
395       StepSize:=StepSize div 2;
396       Pass:=Pass+1; writeln('Pass = ',Pass:3,',  StepSize =
    ',StepSize:3);
397       for K:=1 to StepSize do
398       begin
399         repeat
400           I:=K; J:=K+StepSize; Change:=false;
401           repeat
402             if VRP[1]^.TimeVec[I]>VRP[1]^.TimeVec[J] then
403             begin
404               Change:=true;
405               TempReal:=VRP[1]^.TimeVec[I];
406               VRP[1]^.TimeVec[I]:=VRP[1]^.TimeVec[J];
407               VRP[1]^.TimeVec[J]:=TempReal;
408               TempReal:=VRP[1]^.DdVec[I];
409               VRP[1]^.DdVec[I]:=VRP[1]^.DdVec[J];
410               VRP[1]^.DdVec[J]:=TempReal;
411               TempReal:=VRP[1]^.RateVec[I];
412               VRP[1]^.RateVec[I]:=VRP[1]^.RateVec[J];
413               VRP[1]^.RateVec[J]:=TempReal;
414             end; {if VRP[1]^.TimeVec[I]>VRP[1]^.TimeVec[J]}
415             I:=J; J:=J+StepSize;
416           until J>NumData;
417         until not Change;
418       end; {for K}
419     end; {while StepSize}
420   end {if NumData>1}
421   else writeln('Insufficient data in memory!');
422 end; {Procedure SortForTime}
423
424 Function MarkRec {Find and mark the first recovery stage}
425 (var StartRec, EndRec: integer; var DischTime: real; NumData: integer):
426   boolean;
427 var
428   Valid, ManualEnt: boolean;
429   I: integer;
430 begin
431   writeln('Searching for recovery data'); I:=0;
432   repeat I:=I+1 until (VRP[1]^.RateVec[I]=0) or (I=NumData);
433   if I=NumData
434     then begin Valid:=false; StartRec:=0; EndRec:=0 end
435     else begin
436       Valid:=true; StartRec:=I; DischTime:=VRP[1]^.TimeVec[I-1]
437     end;
438   if Valid then
```

```
439   begin
440     repeat I:=I+1 until (VRP[1]^.RateVec[I]<>0) or (I=NumData);
441     if VRP[1]^.RateVec[I]<>0 then EndRec:=I-1 else EndRec:=I;
442   end; {if Valid}
443   if not Valid then writeln('The search for recovery was unsuccessful.')
444   else begin
445     writeln('The search for recovery gave the first recovery reading as
    ');
446     writeln('reading No. ',StartRec:2,', time ='
447       ,VRP[1]^.TimeVec[StartRec]:7:1,'min., and');
448     writeln('end of recovery at reading No. ',EndRec:2,', time ='
449       ,VRP[1]^.TimeVec[EndRec]:7:1,'min.');
450     writeln('The duration of discharge obtained was ',
    DischTime:7:1,'min.');
451   end; {else EndRec}
452   if EndRec=0
453   then begin
454     write('  Please enter the record number of the first recovery ');
455     writeln('reading:');
456     write('Enter 0 to exit '); StartRec:=ReadInt(1);
457     if StartRec<>0 then ManualEnt:=true else ManualEnt:=false;
458   end
459   else begin
460     write('  Do you want to accept the above values? ');
461     if Response('YN')='N'
462     then begin
463       ManualEnt:=true;
464       write('  Please enter the record number of the first recovery ');
465       write('reading:');
466       StartRec:=ReadInt(1);
467     end
468     else ManualEnt:=false;
469   end; {if-then-else}
470   if ManualEnt then
471   begin
472     write('What last record number for recovery? ');
473     EndRec:=ReadInt(1);
474     write('Total duration of discharge? ');
475     DischTime:=ReadReal(2);
476   end; {if Manual entry}
477   if (EndRec<=StartRec) or (EndRec>NumData) or (DischTime<1)
478   then begin
479     Valid:=false;
480     writeln('The data are invalid!'); delay(2000);
481   end
482   else Valid:=true;
483   MarkRec:=Valid;
484 end; {Function MarkRec}
485
486 Procedure TOverTOne; {Convert recovery times to t/t1}
487 var
488   Valid: boolean;
489   StartRec, EndRec: integer;       {First and last record number of
    recovery}
490   DischTime: real;
491 begin
492   ClrScr;
493   Valid:=MarkRec(StartRec, EndRec, DischTime, NumData[1]);
494   if Valid then
```

```
495   begin
496     writeln('   Converting times to t/t1');
497     for I:=StartRec to EndRec do
498       VRP[1]^.TimeVec[I]:=
499         VRP[1]^.TimeVec[I]/(VRP[1]^.TimeVec[I]-
    VRP[1]^.TimeVec[StartRec-1]);
500   end; {if valid}
501 end; {Procedure TOverTOne}
502
503 Procedure RTMinusRTOne; {Convert recovery times to root t minus root t1}
504 var
505   Valid: boolean;
506   StartRec, EndRec: integer;    {First and last record number of
    recovery}
507   DischTime: real;
508 begin
509   ClrScr;
510   Valid:=MarkRec(StartRec, EndRec, DischTime, NumData[1]);
511   if Valid then
512   begin
513     writeln('   Converting times to (root t minus root t1) squared');
514     for I:=StartRec to EndRec do
515       VRP[1]^.TimeVec[I]:=sqr(sqrt(VRP[1]^.TimeVec[I])-
516         sqrt(VRP[1]^.TimeVec[I]-VRP[1]^.TimeVec[StartRec-1]));
517   end; {if valid}
518 end; {Procedure RTMinusRTOne}
519
520 Procedure MergeFiles; {Join the data of two files, end to end}
521 var
522   I, J, Answer: integer;
523 begin
524   ClrScr;
525   writeln('       Merge files utility');
526   write('   A file read from disk will be appended to the data in ');
527   writeln('memory.  The');
528   writeln('file details for the combined file will be those of the
    data');
529   writeln('originaly in memory.');
530   write('   Do you want to '); Answer:=CapOptions('Continue, or Exit.');
531   if Answer=1 then
532   begin
533     ReadTestDataFile(VRP[2]^.TimeVec,VRP[2]^.DdVec,VRP[2]^.RateVec,
534       TestType[2],WellType[2],Distance[2],NumData[2]);
535     J:=0;
536     for I:=NumData[1]+1 to NumData[1]+NumData[2] do
537     begin
538       J:=J+1;
539       VRP[1]^.TimeVec[I]:=VRP[2]^.TimeVec[J];
540       VRP[1]^.DdVec[I]:=VRP[2]^.DdVec[J];
541       VRP[1]^.RateVec[I]:=VRP[2]^.RateVec[J];
542     end; {for}
543     NumData[1]:=NumData[1]+NumData[2];
544   end; {if Answer=1}
545 end; {Procedure MergeFiles}
546
547 Procedure ChangeDescription; {Alter data describing discharge test and
    well}
548 begin
549   writeln('Do you want to change the type of test? ');
```

```
  if Response('YN')='Y' then
  begin
    write('What type of test, ');
    TestType[1]:=Test(CapOptions('Discharge, Recovery or
Simulation? ')-1);
  end;
  writeln('Do you want to change the type of well? ');
  if Response('YN')='Y' then
  begin
    write('What type of well, ');
    WellType[1]:=Well(CapOptions('Pumped or Observation')-1);
  end;
  writeln('Do you want to change the distance (R) ? ');
  if Response('YN')='Y' then
  begin
    if WellType[1]=Observation then
      write('Distance from pumped well to observation well (m)? ')
      else write('Effective radius of discharge well? (m) ');
    Distance[1]:=ReadReal(1);
  end;
end; {Procedure ChangeDescription}

Procedure DispMenu2; {Display Menu number 2, editing menu}
begin
  ClrScr;
  writeln('        DTDHA:   Menu Number 2');
  writeln('  Which option?');
  writeln('Press the indicated number or letter key.');
  writeln;
  writeln('1: Add a constant to entries;');
  writeln('2: Multiply entries by a constant;');
  writeln('3: Delete a reading;');
  writeln('4: Delete a number of readings;');
  writeln('5: Simulate full recovery between discharge stages;');
  writeln('6: Convert recovery times to t/t1;');
  writeln('7: Convert recovery times to (root t minus root t1)
squared;');
  writeln('A: Alter an individual entry;');
  writeln('B: Correct data for background ''noise'';');
  writeln('C: Change test description data;');
  writeln('M: Merge current data with another file;');
  writeln('S: Sort into order of increasing time;');
  writeln('R: Return to main menu.');
end; {Procedure DispMenu2}

BEGIN {# main part of overlay procedure DTDHMEN2}
  if NumData[1]>0 then
  begin
    DispMenu2;
    repeat
      repeat
        Ch:='x'; repeat read(kbd,Ch) until Ch<>'x'; Ch:=UpCase(Ch);
      until Ch in ValidResponse;
      case UpCase(Ch) of
        '1': AddConst;
        '2': MultByConst;
        '3': DeleteReading;
        '4': DeleteReadings;
        '5': SimulateRec(NumData[1]);
```

```
607         '6': TOverTOne;
608         '7': RTMinusRTOne;
609         'A': AlterEntry(VRP[1]^.TimeVec, VRP[1]^.DdVec, VRP[1]^.RateVec,
610           NumData[1]);
611         'B': CorrectForBackground;
612         'C': ChangeDescription;
613         'M': MergeFiles;
614         'S': SortForTime(NumData[1]);
615       end; {of cases}
616       DataSaved:=false;
617       if Ch<>'R' then DispMenu2;
618     until Ch='R';
619   end {if NumData[1]>0}
620   else begin writeln('No data in memory!'); delay(2000); end;
621 END; {OVERLAY PROCEDURE MENUTWO}
622 {# End of include file DTDHMEN2.SEG}
```

8.3. Include file DTDHMEN3.SEG, listing

```
 1
 2 {Include file DTDHMEN3.SEG}
 3 {#} OVERLAY PROCEDURE MENUTHREE;
 4 const
 5   ValidResponse:
 6     set of char=['1','2','3','4','5','6','7','8','9','C','M','R'];
 7
 8 Procedure GetDdAndRate {Find the drawdown and rate at a given time}
 9   (RorTime: real; var Error: boolean;
10   var NumOfSteps: integer; var DdForStep, RateForStep: SmallVec);
11 var
12   Finished: boolean;
13   I, StepNum: integer;
14   TempRate: real;
15 begin
16   Error:=false; Finished:=false; StepNum:=1;
17   StepNum:=1; I:=1; writeln; writeln('Calculating');
18   while VRP[1]^.RateVec[I]=0 do I:=I+1;
19   TempRate:=VRP[1]^.RateVec[I];
20   if VRP[1]^.TimeVec[I]>RorTime then
21   begin
22     writeln('Given time is too small: procedure aborted.');
23     Error:=true; delay(3000);
24   end;
25   if not Error then
26   begin
27     repeat
28       while (VRP[1]^.TimeVec[I+1]<RorTime) and
29         (VRP[1]^.RateVec[I+1]=TempRate) and (I<NumData[1]) do I:=I+1;
30       if VRP[1]^.RateVec[I+1]<>TempRate then
31       begin
32         Error:=true;
33         writeln('Step No. ',StepNum:3,' has too short a duration.');
34         writeln('I=',I:3,',  Rate =',VRP[1]^.RateVec[I]:10:2,
35           ',  Time =',VRP[1]^.TimeVec[I]:10:3);
36       end;
37       if (I=NumData[1]) and (StepNum<3) then
38       begin
39         Error:=true;
40         writeln('Insufficient discharge steps; there is/are only '
41           ,StepNum:3);
```

```
42        end;
43        if (VRP[1]^.TimeVec[I+1]>=RorTime) and not Error then
44        begin
45          DdForStep[StepNum]:=
    VRP[1]^.DdVec[I]+(RorTime-VRP[1]^.TimeVec[I])/
46              (VRP[1]^.TimeVec[I+1]-VRP[1]^.TimeVec[I])*
47              (VRP[1]^.DdVec[I+1]-VRP[1]^.DdVec[I]);
48          RateForStep[StepNum]:=VRP[1]^.RateVec[I];
49          writeln('Step No. ',StepNum:3);
50          write('Drawdown =',DdForStep[StepNum]:10:3);
51          writeln(' step rate =',RateForStep[StepNum]:10:2);
52          StepNum:=StepNum+1;
53          while (VRP[1]^.RateVec[I]=TempRate) and (I<NumData[1]) do I:=I+1;
54          if (I=NumData[1]) or (VRP[1]^.RateVec[I]=0) then Finished:=true;
55          if not Finished then TempRate:=VRP[1]^.RateVec[I];
56        end; {if}
57      until Finished or Error;
58    end; {if not Error}
59    NumOfSteps:=StepNum-1;
60  end; {of Procedure GetDdAndRate}
61
62  Procedure RunTrial {Produce trial drawdowns from calculated coefficients}
63  (A, B, C, Exponent: real);
64  var
65    Rate, Time, Drawdown: real;
66  begin
67    Rate:=1;
68    repeat
69      write('Enter trial discharge rate (0 to exit) ');
70      Rate:=ReadReal(1);
71      if Rate<>0 then
72      begin
73        write('Will you enter times in ');
74        TimeUnit:=TimeUnits(CapOptions('Minutes or Days')-1);
75        if TimeUnit=Minutes
76          then write('Enter time (min.) ')
77          else write('Enter time (days) ');
78        Time:=ReadReal(2);
79        if TimeUnit=Days then Time:=Time*1440;
80        if Exponent=2 then Drawdown:=A*Rate+B*Rate*Log(Time)+C*sqr(Rate)
81          else Drawdown:=A*Rate+B*Rate*Log(Time)+C*exp(ln(Rate)*Exponent);
82        writeln('Calculated drawdown = ',Drawdown:8:3);
83      end;
84    until Rate=0;
85  end; {Procedure RunTrial}
86
87  {#----- Major procedure Rorabaugh -----#}
88  Procedure Rorabaugh; {Rorabaugh's analysis of step test data}
89  var
90    Error, Finished: boolean;
91    I, NumOfSteps, Pointer, LastReading, ItCount: integer;
92    MinDdOverRate, ThisDrawdown, ThisTime, FinalRate, FormConst: real;
93    A, B, C, LowerLim, MidPoint, UpperLim, SumOfCurves, WellConst: real;
94    AverageCurve, Exponent, Trans, DeltaS, RorTime, Rate: real;
95    SumOfSlopes, AverageSlope, RateFirstStep, PreviousTime: real;
96    Curve, DdForStep, DdOverRate, RateForStep, SlopeForStep, X, Y:
     SmallVec;
97    Vec1, Vec2: SmallVec;
98
```

```
 99   procedure EnterData; {for manual entry}
100   begin
101     write('Transmissivity = '); Trans:=ReadReal(1);
102     write('Final discharge rate of the test = '); FinalRate:=ReadReal(1);
103     write('Time of the drawdown measurement = '); ThisTime:=ReadReal(1);
104     write('Drawdown at that time and rate = ');
    ThisDrawdown:=ReadReal(2);
105     DeltaS:=0.183*FinalRate/(Trans*ln(10));
106     YIntercept:=ThisDrawdown-DeltaS*ln(ThisTime)
107   end; {sub procedure EnterData}
108
109   function DataValid {Check data is valid for calculation of
    transmissivity}
110   : boolean;
111   var
112     Valid: boolean;
113     I, FirstReading: integer;
114     FirstQ: real;
115   begin
116     Valid:=true; I:=1;
117     while (VRP[1]^.RateVec[I]=0) and (I<NumData[1]) do I:=I+1;
118     FirstQ:=VRP[1]^.RateVec[I]; FirstReading:=I;     {Get first discharge}
119     while (VRP[1]^.RateVec[I+1]=FirstQ) and (I<NumData[1]) do
120       I:=I+1;
121       {I has the rec. number of the last reading of the first step.}
122       LastReading:=I; RateFirstStep:=VRP[1]^.RateVec[LastReading];
123     I:=I-4; if I<1 then I:=1;
124     writeln('First rate =',FirstQ:10:2);
125     repeat
126       I:=I+1;
127       writeln('Rate ',I:3,' =',VRP[1]^.RateVec[I]:10:2);
128       if VRP[1]^.RateVec[I]<>FirstQ then
129       begin
130         writeln('Change of rate in data!');
131         Valid:=false;
132       end;
133     until (I=LastReading) or not Valid;
134     if I<FirstReading+6 then Valid:=false; DataValid:=Valid;
135   end; {sub function DataValid}
136
137 begin  {# main part of Procedure Rorabaugh}
138   TextColor(green);
139   ClrScr;
140   writeln('        Rorabaugh''s analysis');
141   writeln;
142   writeln('  Data must be in the full recovery between stages form for');
143   writeln('this operation.'); writeln;
144   writeln('   Data within each step must be in order of increasing
    times.');
145   NumOfSteps:=0;
146   I:=1; while VRP[1]^.RateVec[I]=0 do I:=I+1;
147   PreviousTime:=VRP[1]^.TimeVec[I];
148   repeat
149     I:=I+1;
150     if (PreviousTime>VRP[1]^.TimeVec[I])
151       then begin NumOfSteps:=NumOfSteps+1; Rate:=VRP[1]^.RateVec[I] end;
152     PreviousTime:=VRP[1]^.TimeVec[I];
153   until (VRP[1]^.RateVec[I+1]=0) or (I=NumData[1]);
```

```
154  if Rate<>0
155    then NumOfSteps:=NumOfSteps+1; {Count last step}
156  if NumOfSteps<3 then
157  begin
158    Error:=true;
159         writeln('Insufficient steps, check the data and written notes.');
160  end;
161  if not Error then
162  begin
163    write('What time for Rorabaugh? (min.) '); RorTime:=ReadReal(2);
164         GetDdAndRate(RorTime, Error, NumOfSteps, DdForStep, RateForStep);
165  end;
166  if not Error then
167  begin
168    Finished:=false;
169    MinDdOverRate:=1000;
170    for I:=1 to NumOfSteps do
171    begin
172      DdOverRate[I]:=DdForStep[I]/RateForStep[I];
173         if DdOverRate[I]<MinDdOverRate then MinDdOverRate:=DdOverRate[I];
174    end;
175    UpperLim:=MinDdOverRate*0.999;
176    LowerLim:=1e-36; ItCount:=0;
177    repeat
178      MidPoint:=exp((ln(UpperLim)+ln(LowerLim))/2);
179      FormConst:=MidPoint;
180      for I:=1 to NumOfSteps do
181      begin
182        X[I]:=ln(RateForStep[I]);
183        Y[I]:=ln(DdOverRate[I]-FormConst)
184      end; {for I}
185      for I:=1 to NumOfSteps-1 do SlopeForStep[I]:=
186        (Y[I+1]-Y[I])/(X[I+1]-X[I]);
187      for I:=1 to NumOfSteps-2 do Curve[I]:=
188        SlopeForStep[I+1]-SlopeForStep[I];
189                                              {If Curve>0, curve is concave}
190      if NumOfSteps=3 then AverageCurve:=Curve[1]
191      else begin
192        SumOfCurves:=0;
193        for I:=1 to NumOfSteps-2 do SumOfCurves:=SumOfCurves+Curve[I];
194        AverageCurve:=SumOfCurves/(NumOfSteps-2)
195      end; {if-then-else}
196      ItCount:=ItCount+1;
197      if abs(AverageCurve)<0.00001 then Finished:=true;
198      if FormConst<1e-20 then
199      begin
200        Finished:=true;
201        writeln('Formation constant is negligible.');
202      end;
203      writeln('Upper limit = ',UpperLim:10:8,
204        ',  Lower limit = ',LowerLim:10:8);
205      if AverageCurve<0 then UpperLim:=MidPoint else LowerLim:=MidPoint;
206    until Finished or (ItCount>50);
207    SumOfSlopes:=0;
208    for I:=1 to NumOfSteps-1 do SumOfSlopes:=SumOfSlopes+SlopeForStep[I];
209    AverageSlope:=SumOfSlopes/(NumOfSteps-1);
210    Exponent:=AverageSlope+1;  {Exponent = 1+Slope}
211    WellConst:=(DdForStep[NumOfSteps]-FormConst*RateForStep[NumOfSteps])/
212      exp(ln(RateForStep[NumOfSteps])*Exponent);
```

```
213     writeln;
214     writeln('Exponent = ',Exponent:5:3,', well constant =
    ',WellConst:10:8);
215     writeln('Best ''curve'' = ',AverageCurve:9:7,
216       ', formation constant = ',FormConst:10:8);
217     writeln;
218     writeln('Step No.   Formation loss     Well loss     Totals
    Actual');
219     for I:=1 to NumOfSteps do
220     begin
221       A:=FormConst*RateForStep[I];
222       B:=WellConst*exp(ln(RateForStep[I])*Exponent);
223       write('  ',I:3,'       ',A:10:3,'    ',B:10:3,'   ',A+B:10:3);
224       writeln('  ',DdForStep[I]:10:3);
225     end; {for I}
226     writeln;
227     write('   If the data are suitable, the program will be able to ');
228     writeln('calculate');
229     write('a transmissivity, discharge rate, and drawdown for ');
230     writeln('solution of the');
231     write('well equation.  Alternatively, the user may enter these ');
232     writeln('values.');
233     write('   Do you want to enter transmissivity, discharge rate
    etc. ');
234     writeln('for');
235     write('the calculation of the well equation? ');
236     if Response('YN')='Y'
237     then EnterData
238     else begin
239       if not DataValid
240       then begin
241         write('  The data are not suitable.  ');
242         writeln('The values must be entered manually.');
243         EnterData
244       end
245       else begin
246          writeln('Trans. will be calculated from latter part of step 1.');
247         Pointer:=LastReading-4;
248         writeln('Log time     Drawdown');
249         for I:=1 to 5 do
250         begin
251           Vec1[I]:=Log(VRP[1]^.TimeVec[Pointer+I-1]);
252           Vec2[I]:=VRP[1]^.DdVec[Pointer+I-1];
253           writeln(Vec1[I]:8:3,'    ',Vec2[I]:8:3);
254         end;
255         LinReg(Vec1, Vec2, 5); I:=LastReading-4;
256         writeln('Delta s = ',Slope:7:2);
257         Trans:=0.183*RateFirstStep/(Slope);
258         writeln('Discharge rate =',RateFirstStep:8:2,
259           ', Transmissivity =',Trans:8:2);
260       end; {if-then-else}
261     end; {if Response}
262     A:=(YIntercept-WellConst*exp(ln(RateFirstStep)*Exponent))
    /RateFirstStep;
263     B:=0.183/Trans; C:=WellConst;
264     writeln('A (min) =',A:10:8,', A (day) =',A+B*MinDayLog:10:8,
265       ', B =',B:10:8,', C =',C:10:8);
266     writeln('Exponent =',Exponent:5:2);
267     RunTrial(A, B, C, Exponent);
```

```
  end {if not Error}
  else begin
    writeln('Solution of well equation not attempted.');
    delay(7000);
  end; {if-then-else}
end; {Procedure Rorabaugh}
{#----- End of major procedure Rorabaugh -----#}

Procedure SQvsQ;  {Evaluation of well equation by the s/Q vs. Q method}
var
  Error: boolean;
  Ch: char;
  I, J, NumOfSteps, first, last: integer;
  A, B, C, Trans, AnalTime, PreviousTime: real;
  DdAtTZero, RateForStep: SmallVec;
  TempVec1, TempVec2, DdForStep, SOverQ: SmallVec;
begin
  ClrScr;
  writeln('     Solve the well equation by the s/Q vs. Q method.');
  writeln;
  writeln('  Data must be in the full recovery between stages form for');
  writeln('this operation.'); writeln;
  writeln('   Data within each step must be in order of increasing
    times.');
  NumOfSteps:=1; I:=1;
  while VRP[1]^.RateVec[I]=0 do I:=I+1;
  FirstDatum[1]:=I;
  PreviousTime:=VRP[1]^.TimeVec[FirstDatum[1]];
  while (I<NumData[1]) and (VRP[1]^.RateVec[I+1]<>0) do
  begin
    I:=I+1;
    if (PreviousTime>VRP[1]^.TimeVec[I]) then
    begin
      NumOfSteps:=NumOfSteps+1;
      LastDatum[NumOfSteps-1]:=I-1; FirstDatum[NumOfSteps]:=I;
    end; {if}
    PreviousTime:=VRP[1]^.TimeVec[I]
  end; {while I}
  LastDatum[NumOfSteps]:=I;
  writeln('Step     First rec.    Last rec.');
  for I:=1 to NumOfSteps do
  writeln(I:3,'         ',FirstDatum[I]:3,'          ',LastDatum[I]:3);
  if NumOfSteps<2 then
  begin
    Error:=true;
    write('The data are not suitable for this method, ');
    writeln('check form of data and written notes.');
  end;
  if not Error then
  begin
    write('What time for this analysis? (min.) '); AnalTime:=ReadReal(2);
    GetDdAndRate(AnalTime, Error, NumOfSteps, DdForStep, RateForStep);
    if not Error then
    begin
      write('Will you enter transmissivity, or do you want it
    calculated ');
      writeln('from the');
      write('data of the first step? (E or C) ');
      if Response('EC')='E' then
```

```
325       begin
326         write('What value for transmissivity for this calculation? ');
327         Trans:=ReadReal(2);
328       end
329       else begin
330         First:=FirstDatum[1];
331         Last:=LastDatum[1];
332         for J:=1 to Last-First+1 do
333         begin
334           TempVec1[J]:=Log(VRP[1]^.TimeVec[First+J-1]);
335           TempVec2[J]:=VRP[1]^.DdVec[First+J-1];
336         end; {for J}
337         LinReg(TempVec1, TempVec2, Last-First+1);
338         Trans:=0.183*VRP[1]^.RateVec[First]/Slope;
339         writeln('Calculated Delta s is ',Slope:8:3,', Trans.
    is ',Trans:8:2);
340       end;
341       writeln('Step     Slope     Y intercept');
342       for I:=1 to NumOfSteps do
343       begin
344         First:=FirstDatum[I];
345         Last:=LastDatum[I];
346         for J:=1 to Last-First+1 do
347         begin
348           TempVec1[J]:=ln(VRP[1]^.TimeVec[First+J-1]);
349           TempVec2[J]:=VRP[1]^.DdVec[First+J-1];
350         end; {for J}
351         LinReg(TempVec1, TempVec2, Last-First+1);
352         writeln(I:3,'   ',Slope:10:7,'   ',YIntercept:10:7);
353         DdAtTZero[I]:=YIntercept;
354         RateForStep[I]:=VRP[1]^.RateVec[(First+Last) div 2];
355       end; {for I}
356       writeln;
357       write('Below are the values used to calculate the slope ');
358       writeln('and intercept.');
359       writeln('    s/Q             Q');
360       for I:=1 to NumOfSteps do
361       begin
362         SOverQ[I]:=DdAtTZero[I]/RateForStep[I];
363         writeln(SOverQ[I]:10:7,'    ',RateForStep[I]:10:2);
364       end; {for I}
365       LinReg(RateForStep, SOverQ, NumOfSteps);
366       B:=0.183/Trans; C:=Slope; A:=YIntercept;
367       writeln;
368       writeln('A (min) =',A:10:8,', A (day) =',A+B*MinDayLog:10:8,
369         ', B =',B:10:8,', C =',C:10:8);
370       RunTrial(A, B, C, 2);
371     end; {first if not Error}
372   end {second if not Error}
373   else begin
374     writeln('Solution of well equation not attempted.');
375     delay(7000);
376   end; {if-then-else}
377 end; {Procedure SQvsQ}
378
379 {#----- Begin major procedure Sternberg -----#}
380 Procedure Sternberg; {Sternberg's analysis of step test data}
381 type
382   PointerVec=Array[1..20] of integer;
```

```
383   PointerVec2=Array[1..20] of real;
384 var          {Special variables for Procedure Sternberg}
385   Error, Valid: boolean;
386   Ch: Char;
387   NumOfSteps: integer;
388   A, B, C, PreviousTime, WellExp: real;
389   Pointer1, Pointer2, FirstDatum, LastDatum: array[1..20] of integer;
390   StepRate, TimeStartStep, TimeEndStep: PointerVec2;
391   DeltaQ, StrnVec: MainVec;
392
393   procedure MarkStepData; {Mark the beginning and end of each stage}
394   var
395     Again: boolean;
396     I, J: integer;
397   begin
398     writeln('Marking beginning and end of steps ......');
399     I:=1; FirstDatum[1]:=1; LastDatum[1]:=1; J:=1;
400     repeat
401       repeat
402         repeat
403           J:=J+1;
404           if VRP[1]^.RateVec[J]=VRP[1]^.RateVec[FirstDatum[I]] then
405             LastDatum[I]:=J;
406         until (VRP[1]^.RateVec[J]<>VRP[1]^.RateVec[FirstDatum[I]])
407           or (J>NumData[1]);
408         if (LastDatum[I]-FirstDatum[I]<2) and (J<=NumData[1]) then
409         begin
410           FirstDatum[I]:=J; LastDatum[I]:=J; Again:=true;
411         end else Again:=false;
412       until not Again;
413       if J<=NumData[1] then
414       begin
415         I:=I+1; FirstDatum[I]:=J; LastDatum[I]:=J;
416       end;
417     until J>=NumData[1];
418     writeln('Step #   First rec.   Last rec.');
419     NumOfSteps:=I; LastDatum[I]:=NumData[1];
420     for I:=1 to NumOfSteps do
421     begin
422       writeln(I:4,'       ',FirstDatum[I]:3,'           ',
    LastDatum[I]:3);
423       TimeStartStep[I]:=VRP[1]^.TimeVec[FirstDatum[I]];
424       TimeEndStep[I]:=VRP[1]^.TimeVec[LastDatum[I]];
425     end;
426   end; {sub procedure MarkStepData}
427
428   procedure PrintSternberg; {Print the Sternberg vectors on paper}
429   var
430     I, J, LineCount: integer;
431   const
432     StringDash=
433     '------------------------------------------------------------';
434     Heading=
435       'No.     Q         min.     days     Drawdown  Delta Q  Stern.
    vec.';
436   begin
437     writeln('The printer should be switched on and at top of form.');
438     writeln(lst,'              File name, ',FileName);
439     writeln(lst,'      Sternberg output data');
```

```
440     writeln(lst);
441     writeln(lst,'       ',Heading); LineCount:=4;
442     for I:=1 to NumData[1] do
443     begin
444       writeln(lst,'      ',I:3,' ',VRP[1]^.RateVec[I]:8:1,' '
445         ,VRP[1]^.TimeVec[I]:9:1,' ',VRP[1]^.TimeVec[I]/1440:9:4
446         ,' ',VRP[1]^.DdVec[I]:9:3,' ',DeltaQ[I]:9:2,'  ',StrnVec[I]:9:2);
447       LineCount:=LineCount+1;
448       if LineCount=55 then
449       begin
450         for J:=1 to 12 do writeln(lst);
451         writeln(lst,'       ',Heading);
452         LineCount:=2
453       end
454       else begin
455         if (LineCount div 7)*7=LineCount then
456         begin
457           writeln(lst,'       ',StringDash);
458           LineCount:=LineCount+1;
459         end;
460       end; {if-then-else}
461     end; {for I}
462   end; {sub procedure PrintSternberg}
463
464   procedure DisplayData; {Display the Sternberg vectors}
465   var
466     I, J: integer;
467   const
468     Heading=
469     'No.     Q         min.     days     Drawdown  Delta Q  Stern. vec.';
470   begin
471     ClrScr; J:=0;
472     writeln(Heading);
473     for I:=1 to NumData[1] do
474     begin
475       J:=J+1;
476       writeln(I:3,' ',VRP[1]^.RateVec[I]:8:2,' ',VRP[1]^.TimeVec[I]:9:1
477         ,' ',VRP[1]^.TimeVec[I]/1440:9:4,' ',VRP[1]^.DdVec[I]:9:3,' '
478         ,DeltaQ[I]:9:2,'  ',StrnVec[I]:9:2);
479       if J=20 then
480       begin
481         writeln('Press any key to continue.');
482         Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
483         writeln(Heading);
484         J:=0;
485       end; {if J} 486     end; {for I} 487     writeln('Press any key to
    continue.');
488     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
489   end; {sub procedure DisplayData}
490
491   procedure SolvStrn; {Solution part of procedure}
492   var
493     Valid, ZeroRate, TooClose: boolean;
494     Ch: char;
495     RecStepNo, TempInt1, TempInt2, TempInt3, I, J, K: integer;
496     Sum1, Sum2, Sum3, Sum4, SumOfSlopes, QZeroDd: real;
497     OneMinDd, Pointer, YInterceptVec, SlopeVec, YQ: SmallVec;
498     TempVec1, TempVec2: SmallVec;
499     D2, Q1, Q2: real;
```

```
120   Data Handling

500   begin
501     MarkStepData; Valid:=true;
502     if Valid then
503     begin
504       writeln; writeln('Calculating');
505                                       {Record discharge rate for each step}
506       writeln('Step #   First rec.   Last rec.   Rate');
507       for J:=1 to NumOfSteps do
508       begin
509         StepRate[J]:=VRP[1]^.RateVec[FirstDatum[J]];
510         writeln(' ',J:3,'        ',FirstDatum[J]:3,
            ',LastDatum[J]:3,
511         '    ',StepRate[J]:9:2);
512       end;
513       writeln;
514       RecStepNo:=0; TempInt1:=0; TempInt2:=0;
515       for J:=1 to NumOfSteps do
516       begin
517         if StepRate[J]=0 then
518         begin
519           TempInt2:=TempInt2+1;
520           if LastDatum[J]-FirstDatum[J]>TempInt1 then
521           begin
522             RecStepNo:=J;
523             TempInt1:=LastDatum[J]-FirstDatum[J];
524           end;
525         end; {if StepRate}
526       end; {for J.  RecStepNo marks the largest recovery step.}
527       SumOfSlopes:=0;
528                               {Calculate slope, Y intercept of each step}
529       writeln('   Step #   Y intercept   Slope');
530       for I:=1 to NumOfSteps do
531       begin
532         for J:=1 to LastDatum[I]-FirstDatum[I]+1 do
533         begin
534           TempVec1[J]:=StrnVec[FirstDatum[I]+J-1];
535           TempVec2[J]:=VRP[1]^.DdVec[FirstDatum[I]+J-1];
536         end;
537         TempInt3:=LastDatum[I]-FirstDatum[I]+1;
538         LinReg(TempVec1, TempVec2, TempInt3);
539         YInterceptVec[I]:=YIntercept; SlopeVec[I]:=Slope;
540         writeln('   ',I:3,'    ',YIntercept:9:5,'   ',Slope:11:6);
541         SumOfSlopes:=SumOfSlopes+Slope;
542         if StepRate[I]<>0 then
543           YQ[I]:=YInterceptVec[I]/StepRate[I] else YQ[I]:=0;
544       end; {for I}
545       B:=SumOfSlopes/NumOfSteps;
546       writeln('    Average slope, B =',B:9:6);
547       writeln;
548       writeln('Press any key to continue.');
549       Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
550       writeln;
551       if RecStepNo=0
552       then begin
553         writeln('No recovery step present, using inferior method.');
554         TempInt3:=0;
555         for I:=1 to NumOfSteps do
556         begin
557           if YQ[I]<>0 then
```

```
558           begin
559             TempInt3:=TempInt3+1;
560             TempVec1[TempInt3]:=StepRate[I];
561             TempVec2[TempInt3]:=YQ[I];
562           end;
563         end; {if YQ}
564         LinReg(TempVec1, TempVec2, TempInt3);
565         A:=YIntercept; C:=Slope; WellExp:=2;
566         writeln('A (min) =',A:10:8,', A (day) =',A+B*MinDayLog:10:8,
567           ', B =',B:10:8,', C =',C:10:8);
568         RunTrial(A, B, C, 2);
569       end {if RecStep=0}
570       else begin
571         writeln('Recovery step present, using superior method.');
572                         {Final part of anal. of data including recovery}
573         for I:=1 to NumOfSteps do
574         begin
575           if StepRate[I]<>0
576           then begin
577             TempInt1:=(FirstDatum[I]+LastDatum[I]) div 2;
578                                              {Point to mid. datum of step}
579             QZeroDd:=YInterceptVec[RecStepNo]+SlopeVec[RecStepNo]*
580               StrnVec[TempInt1];
581             OneMinDd[I]:=QZeroDd-VRP[1]^.DdVec[TempInt1];
582             writeln('Drawdown at 1 min. for step',I:3,' is '
583               ,-OneMinDd[I]:7:3);
584           end {if StepRate}
585           else OneMinDd[I]:=0;
586         end; {for I}
587         for I:=1 to NumOfSteps-1 do
588         begin
589           for J:=I+1 to NumOfSteps do
590           begin
591             if (StepRate[I]=StepRate[J]) and (StepRate[I]<>0) then
592             begin Pointer[I]:=1; TempInt2:=TempInt2+1 end;
593           end; {for J}
594         end; {for I}
595         Sum1:=0; Sum2:=0; Sum3:=0; Sum4:=0; TempInt1:=0;
596         write('            Dd(I)-Dd(J)  Q(J)-Q(I)  Q(J)+Q(I)  ');
597         writeln('(Dd(I)-Dd(J))/(Q(J)-Q(I))');
598         write('  I    J        D2          Q1          Q2      ');
599         writeln('       D2/Q1');
600         for I:=1 to NumOfSteps-1 do
601         begin
602           if Pointer[I]<>1
603           then begin
604             for J:=I+1 to NumOfSteps do
605             begin
606                  if (StepRate[I]=0) or (StepRate[J]=0) then ZeroRate:=true
607                 else ZeroRate:=false;
608               if not ZeroRate then if (Abs(1-(StepRate[I]/StepRate[J]))
    <0.08)
609                 then TooClose:=true else TooClose:=false;
610               if not (((StepRate[J]=0) and (RecStepNo<>J)) or
611                 ((StepRate[I]=0) and (RecStepNo<>I)) or
612                 TooClose) then
613               begin
614                 D2:=OneMinDd[I]-OneMinDd[J];
615                 Q1:=StepRate[J]-StepRate[I];
```

```
616                Q2:=StepRate[J]+StepRate[I];
617                writeln(I:3,'  ',J:3,'  ',D2:9:3,'     ',Q1:9:3,
618                  '  ',Q2:9:3,'         ',D2/Q1:9:5);
619                Sum1:=Sum1+(D2/Q1)*Q2;
620                Sum2:=Sum2+Q2;
621                Sum3:=Sum3+D2/Q1;
622                Sum4:=Sum4+Q2*Q2;
623                TempInt1:=TempInt1+1;
624              end; {if not}
625            end; {for J}
626          end; {if Pointer}
627        end; {for I}
628        Slope:=(TempInt1*Sum1-Sum2*Sum3)/(TempInt1*Sum4-Sum2*Sum2);
629        YIntercept:=(Sum3*Sum4-Sum2*Sum1)/(TempInt1*Sum4-Sum2*Sum2);
630        A:=YIntercept; C:=Slope;
631        writeln('A (min) =',A:10:8,', A (day) =',A+B*MinDayLog:10:8,
632          ', B =',B:10:8,', C =',C:10:8);
633        RunTrial(A, B, C, 2);
634      end; {else; End of superior method}
635    end {if Valid}
636    else begin
637      writeln('Press any key to continue.');
638      Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
639    end; {if-then-else}
640  end; {sub procedure SolvStrn}
641
642  procedure StrnMenu; {Sternberg menu}
643  const
644    ValidResponse: set of char=['1','2','3','R'];
645
646    procedure DispMenu; {Display Sternberg menu}
647    begin
648      ClrScr;
649      writeln('      Sternberg Menu'); writeln;
650      writeln('   Which option?'); writeln;
651      writeln('1: List data on printer;');
652      writeln('2: Display data on screen;');
653      writeln('3: Last part of well equation solution;');
654      writeln('R: Return to analysis menu.');
655    end; {sub sub procedure DispMenu}
656
657  begin
658    repeat
659      DispMenu;
660      repeat
661        Ch:='x'; repeat read(kbd,Ch) until Ch<>'x'; Ch:=UpCase(Ch);
662      until Ch in ValidResponse;
663      case UpCase(Ch) of
664        '1': PrintSternberg;
665        '2': DisplayData;
666        '3': SolvStrn;
667      end; {of cases}
668    until Ch='R';
669  end; {sub procedure StrnMenu}
670
671 begin  {# main part of Procedure Sternberg}
672   ClrScr; Error:=false;
673   writeln('       Modified Sternberg Analysis');
674   writeln;
```

```
675   PreviousTime:=VRP[1]^.TimeVec[1];
676   for I:=2 to NumData[1] do
677   begin
678     if PreviousTime>=VRP[1]^.TimeVec[I] then Error:=true;
679     PreviousTime:=VRP[1]^.TimeVec[I]
680   end; {for I} 681   if Error then
682   begin
683     write('The data are not suitable for this method, ');
684     writeln('check form of data and written notes.');
685   end;
686   if not Error then
687   begin
688     if VRP[1]^.TimeVec[1]<>0
689     then begin
690       writeln('Data invalid!  Initial conditions are not present.');
691         writeln('Please enter drawdown and discharge rate at time zero.');
692       Error:=true;
693     end; {if VRP}
694   end; {if not Error}
695   if not Error then
696   begin
697     if VRP[1]^.RateVec[1]<>0
698     then begin
699       writeln('Data invalid!  First discharge rate is not zero.');
700       Error:=true;
701     end; {if VRP}
702   end; {if not Error}
703   if not Error then
704   begin
705     StrnVec[1]:=0;
706     writeln('Calculating vectors delta Q and the Sternberg vector');
707     for I:=2 to NumData[1] do
708     begin
709       StrnVec[I]:=0;
710       DeltaQ[I]:=VRP[1]^.RateVec[I]-VRP[1]^.RateVec[I-1];
711       for J:=2 to I do
712       begin
713         if DeltaQ[J]<>0 then
714           StrnVec[I]:=StrnVec[I]+DeltaQ[J]
715             *Log(VRP[1]^.TimeVec[I]-VRP[1]^.TimeVec[J-1]);
716       end; {for J}
717       write(StrnVec[I]:8:0);
718     end; {for I}
719     StrnMenu;
720   end {if not Error}
721   else begin
722     writeln('Solution of well equation not attempted.');
723     delay(7000);
724     end; {if-then-else}
725 end; {Procedure Sternberg}
726 {#----- End major procedure Sternberg -----#}
727
728 procedure DispMenu3; {Display of menu 3, analysis menu}
729 begin
730   ClrScr;
731   writeln('        DTDHA:   Menu Number 3');
732   writeln('   Solution of the well equation');
733   writeln('Note that each method will probably give a different
     result.');
```

```
734   write('The modified Sternberg method is the only one which does not ');
735   writeln('require the');
736   writeln('data in a form having full recovery between steps.');
737   writeln('   See the written notes for more information.');
738   writeln;
739   writeln('   Which method?');
740   writeln('Press the indicated number or letter key.');
741   writeln;
742   writeln('1: The modified Sternberg method;');
743   writeln('2: Rorabaugh''s method (evaluate the exponent);');
744   writeln('3: The sQ vs. Q method;');
745   writeln('R: Return to main menu.');
746 end; {Procedure DispMenu3}
747
748 BEGIN {# main part of overlay procedure MENUTHREE}
749   if NumData[1]>0 then
750   begin
751     DispMenu3;
752     repeat
753       repeat
754                 Ch:='x'; repeat read(kbd,Ch) until Ch<>'x'; Ch:=UpCase(Ch);
755       until Ch in ValidResponse;
756       case UpCase(Ch) of
757         '1': Sternberg;
758         '2': Rorabaugh;
759         '3': SQvsQ;
760       end; {of cases}
761       if Ch<>'R' then DispMenu3;
762     until Ch='R';
763   end {if NumData[1]>0}
764   else begin writeln('No data in memory!'); delay(2000); end;
765 END; {OVERLAY PROCEDURE MENU3}
766 {# End of include file DTDHMEN3.SEG}
```

9. REFERENCES

Barden, W. Jr., 1980. Programming Techniques for Level II BASIC. Tandy Corp., Texas, 76107 USA. 224pp.

Bouwer, H., 1978. Groundwater Hydrology. McGraw-Hill Kogakusha Ltd. 480pp.

Clarke, D.K., 1987. Microcomputer Programs for Groundwater Studies, 268pp. Developments in Groundwater Science, 30. Elsevier, Amsterdam/Oxford/New York/Tokyo.

Eden, R.N., and Hazel, C.P., 1973. Computer and Graphical Analysis of Variable Discharge Pumping Tests of Wells. Civil Eng. Trans., Vol. 15(1+2), Inst. Eng. Australia. pp 5-10.

Jacob, C.E., 1947. Drawdown test to determine effective radius of artesian well. Trans. Am. Soc. Civ. Eng. 112: pp 1047-1070.

Lennox, D. H., 1966. Analysis and application of step-drawdown test. J. Hydraul. Div., Proc. Am. Soc. Civ. Eng. 92(HY6): pp 25-48.

Miller, A.R., 1981. Basic Programs for Scientists and Engineers. Sybex, Berkley, California. 318 pp.

Rorabaugh, M.I., 1953. Graphical and theoretical anal. of step-drawdown tests of artesian wells. Proc. Am. Soc. Civ. Eng. 79, separate No. 362, 23 pp.

Theis, C.V., 1935. The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using groundwater storage. Trans. Am. Geophs. Un. 16:pp519-524.

Tremblay, J. & Bunt, R.B., 1981. An Introduction to Computer Science; An Algorithmic Approach. McGraw-Hill Int. Book Co. pp 636.

Chapter 2

This chapter explains the program DRAWDOWN.PAS, which deals with the calculation of drawdown in a piezometer at some distance from a discharging well, and in response to discharge from that well. Program DRAWDOWN was developed from several previously published Basic language programs (Clarke, 1987). It is more comprehensive (because additional aquifer types have been included) and faster (because it is written in a compiled language) than the earlier programs.

In keeping with the practice adopted throughout the greater part of this book, the earlier part of this chapter explains the use of the compiled program DRAWDOWN.CHN; the second part gives a more technical description by procedure and function in alphabetical order. This is followed by a listing, in the order in which they appear in the program, of the key lines of the program. In operation, the program has two stages. Finally comes the program listing itself.

In running the program it will be found that the first stage consists of a number of multiple choice questions which allow the user to specify the type of aquifer, boundary configuration, and the type of simulation. The second stage requires the user to enter the values for aquifer parameters, discharge rates and durations, distances between discharging well and piezometer, and distances from wells to boundaries etc.

1. AN EXPLANATION OF THE AQUIFER TYPES

It is assumed that this program will be used only by people who are competent hydrogeologists, so these notes will not give an explanation of the properties of aquifers, aquitards, and aquicludes.

The simplest aquifer covered by the program is the confined aquifer. In theory this is bounded both top and bottom by aquicludes, and it is assumed to be homogeneous, isotropic, and of infinite extent. As the Theis equation is used for the simulation, the validating assumptions that apply to the use of that equation apply also to the use of this section of the program.

The second aquifer type is the so called Leaky Artesian Aquifer, where the aquifer is bounded below by an aquiclude, and bounded above by an aquitard which in turn is overlain by an unconfined aquifer. The theoretical solution assumes no change in storage in the aquiclude, infinite storage in the overlying source aquifer, and (as with all other solutions presented here) no storage in the discharging well or in the piezometer. (See Bouwer,

1978, pp103-104; Hantush and Jacob, 1955; and Hantush, 1956). In many practical cases these assumptions are approximately met, the change in storage in the aquitard being small, and the specific yield of the overlying unconfined aquifer being several orders of magnitude greater than the storage coefficient of the pumped aquifer. Storage in the wells and/or piezometers becomes negligible at later times.

An unconfined aquifer, for the purpose of this program, is one which is open to the atmosphere above, and bounded by an aquiclude below. Drainage is assumed to take place immediately on discharge from the well. Note that in a real unconfined aquifer there will more likely be a delay, and in the short term the aquifer will behave as if it is confined; therefore this type of simulation should not be taken too seriously at early times. Program NEUMAN.PAS, which is dealt with in Chapter 4, provides a simulation of discharge from an unconfined aquifer showing delayed yield. In an unconfined aquifer transmissivity decreases as the saturated thickness of the aquifer decreases; this effect is approximated by the Cooper-Jacob correction (Cooper and Jacob, 1946). (The equations used are given later in this chapter.) In all cases, first the confined drawdown is calculated, and then the inverse of the Cooper-Jacob correction is applied to that figure to give an approximate unconfined drawdown.

The user is encouraged to study the source code, and check the results of a program run. If an error is found, I would appreciate being notified of such error, how it arose, and a full description of the data used in the simulation in which the error occurred.

2. AN EXPLANATION OF BOUNDARIES

Three types of boundaries are considered, water tight (discharge boundaries), constant head (recharge boundaries), and partial boundaries. For simplicity all boundaries are considered to be vertical planes which in plan are straight lines of infinite length. A discharge boundary is a boundary between an aquifer which contains a discharging well (and one or more piezometers) and an area of impervious rock. A recharge boundary is a plane of contact between the aquifer and an source of infinite recharge, such as a fully penetrating stream. A partial (or semi) boundary is a boundary between areas of differing transmissivity; all aquifer properties other than transmissivity are considered to be constant on both sides of the partial boundary.

All boundaries are simulated by means of image wells (Bouwer, 1978; Freeze and Cherry, 1979; Marino and Luthin, 1982; Clarke, 1987). A discharge

boundary is simulated by an image well having the same discharge rate as the real discharging well, while a recharge boundary is simulated by an image well having a recharge rate equal in magnitude to the discharge rate of the real discharging well (ie. Q has the same absolute value, but opposite sign). The partial boundary is simulated by an image well having a discharge rate between those of the two cases above (a more detailed explanation is given below).

In all cases the discharging well and piezometer both lie on the same side of any boundary. None of the simulations given here are capable of indicating the drawdown that might be expected on the far side of some partial boundary.

In theory, strip aquifers may be simulated by an infinite regression of image wells. An analogy is given by two men standing in a room which has large mirrors on two opposite walls. The mirrors represent the two parallel boundaries, and the two men represent the two wells; the observer being the piezometer, and the other man being the discharging well. When the observer looks at the reflection of the second man he sees not one reflection, but a series of reflections of reflections disappearing into the distance. The series will be repeated on both sides of the room. So it is with the image wells, each successive image causes a little more drawdown in the observation well, but because it is further away than the previous image it's contribution to the total drawdown is less (just as the reflections vanish into the distance). Plainly, in practical applications, it is necessary to decide at some point to stop adding the effects of new image wells.

2.1. Simulation of partial boundaries

This theory was apparently first published by Muskat in 1937; Bear 1972, Walton 1984, and Clarke 1987, all mentioned it. The discharge rate of the image well can be calculated by the equation:

$$Q_i = Q * - \frac{T_2/T-1}{T_2/T+1} \qquad (2.1)$$

where T is the transmissivity on the pumped side of the partial boundary,

T_2 is the transmissivity of the side of the partial boundary furthest from the wells,

Q is the discharge rate of the real well,

and Q_i is the discharge rate for the image well.

The semiboundary simulation in this program use the above equation. The semistrip simulation uses equation 2.1 for the first reflection, but the discharge rate for the second and subsequent reflections are taken from equation 2.2 (which is an extrapolation of equation 2.1).

$$Q_{i2} = Q * -\left[\frac{T_2/T - 1}{T_2/T + 1}\right]^2, \quad Q_{i3} = Q * -\left[\frac{T_2/T - 1}{T_2/T + 1}\right]^3, \ . \ . \ . \qquad (2.2)$$

I can offer no proof, nor independent reference, for equation 2.2.

3. SOME DEMONSTRATION RUNS

3.1. A single drawdown in a simple confined aquifer

This first demonstration will use the simplest options of DRAWDOWN, and will calculate one drawdown for an observation well in an unbounded confined aquifer. Run the program by typing GW, (capitals or lower case) and pressing Enter. (Please refer to the section on 'Getting started' in the introduction if you do not know how to get this far.) When the menu appears, press the number (2) against the word DRAWDOWN. An opening message will be printed, and the first multiple choice question will appear; "What type of aquifer? Confined, Unconfined, Leaky?". In this, as in most of the multiple choice questions, the Capital letters in the latter part of the question will be highlighted as an indication that it is one of these letters which must be used as an answer. For this demonstration, press 'C'.

The next question is "Extent of aquifer: Infinite, One boundary, Strip, sEmi bounded, seMi strip?". Press 'I', indicating that you want the simplest case, an infinite, unbounded aquifer.

Answer the question "Do you want calculations for One time, or a Series of times?" with 'O'. Here you have told the computer that you want a drawdown calculated for only one specific time, and you do not want a series of time-drawdown-discharge rate values calculated.

The last multiple choice question is "What time unit, Minute or Day?" Press 'D'. This entry effects only the entry of times, and the screen output of times (if any); if minutes had been chosen then you would enter times in minutes. The calculations are always done in days, and disk output is always in minutes because it is usual practice to record and plot discharge test data in minutes.

Now you must begin to enter data describing the aquifer and configuration of wells for which the drawdown is to be calculated. The first request in this section is for the "Distance of piezometer from discharging well? (m)". ie. How far is it, in metres, from the pumped well to the

observation well (or piezometer) for which you want the drawdown calculated. Type '25' and press the Enter key.

Next you will be asked for the transmissivity. Note that with the exception of times as mentioned above, all units must be consistent; so transmissivity is entered in square metres per day (which is the same as cubic metres per day per metre). Enter '100'.

The storage coefficient is dimensionless, enter '.0001'.

For discharge rate, enter '1100' (cubic metres per day).

You will now be reminded to enter time in days, and asked for the time from the beginning of discharge. (ie. For what time, measuring from the moment that the pump was started, do you require the drawdown calculated?) Enter '1'.

For this simple calculation, all the required data has now been entered, so after a very brief pause the calculated drawdown, 7.167 (m) will be displayed. The drawdown calculation used the Theis equation, as will be explained in the part of these notes on procedures and functions.

3.2. A single drawdown in an unconfined aquifer

For a second demonstration run, press 'Y' at the prompt "Go again? Y/N", then enter 'U' to indicate that you want a drawdown calculated for an unconfined aquifer. Enter all the same answers that you did for the first run, and then enter 15 metres when asked for the 'Thickness of the saturated part of the aquifer'.

The result will be displayed as:

"Calculated confined drawdown is 7.167
Converting to unconfined using the inverse Jacob correction:
Final calculated drawdown is 11.837"

A specific yield of 0.0001 is much too low for an unconfined aquifer, but the purpose of this illustration is to show how the calculated (confined) drawdown, which is a significant part of the aquifer thickness, is increased by the inverse Jacob's correction.

3.3. A simulation of a discharge test in a bounded leaky aquifer

Now try producing a time series. Run the program and answer the questions as below.

1/ Leaky aquifer.

2/ One boundary.

3/ Discharge boundary. ie. the boundary is a water tight boundary, such as an approximately straight line contact with a granite mass having

negligible secondary porosity.

4/ You want a Series of drawdown values against exponentially increasing times. Such as you would get during a discharge test.

5/ Specify time units of Days.

6/ Enter a distance from the pumped well to the observation well of 15m.

7/ The transmissivity will be 300 m*m/day.

8/ Storage coefficient; 0.0002.

9/ 1 step only (you would use two steps if you wanted recovery to be simulated as well as the drawdown phase).

10/ The finishing time for the first (and only) step will be 3 days.

11/ The discharge rate is 300 cubic metres per day.

12/ Enter 0.0005 as the vertical hydraulic conductivity of the aquitard.

13/ The thickness of the aquitard is 1m exactly.

14/ The discharge well is 60m from the boundary.

15/ 50m separates the piezometer from the boundary. These three distances, points 6/, 14/, and 15/, above, are sufficient to fix the geometry of the wells and the boundary.

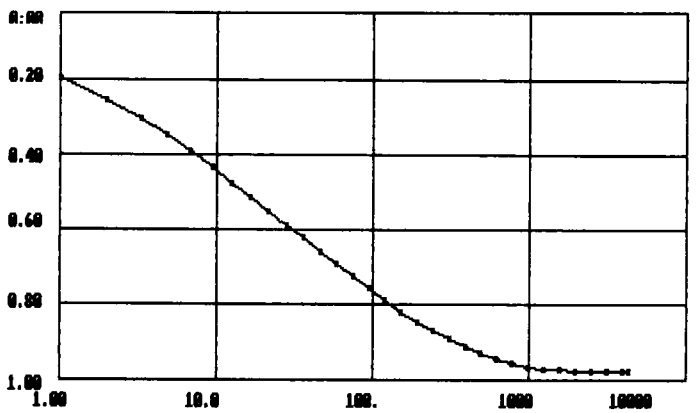
Now you will start getting some results. The first drawdown to be displayed is 0.383m at 0.0007 days (1 minute). I would treat this with scepticism, as the program does not allow for the time required for pressure to equalize between the piezometer and the aquifer; remember at all times that you are dealing with mathematical models of reality, and that the model is much simpler than the reality. Some other of the drawdowns displayed will be 1.023m at 0.0115 days, 1.385m at 0.0408 days, 1.743m at 0.169 days. At 1.366 days the drawdown is distinctly heading toward the final horizontal line of the later stages of withdrawal from a leaky aquifer, with a drawdown of 1.952m. Figure 2.1 shows a copy of a screen graph of this data, as produced by program PLOTWTD.

You will be given the option of saving the output data to disk, or of ending the program without saving the data. Save it this time, for the exercise if nothing more. After pressing S, you will be asked whether you want a 'fast' save or a 'human readable' save. For more information on the disk files see Appendix A, for the present chose the 'human readable' save by pressing H.

Now you must enter the first part of a disk file name; do not enter an extension, but do precede the name with a drive and/or path specification if

you want your data file placed on any disk or directory other than the default. If you are unfamiliar with file names, drive specifications, and the use of paths, you could consult the DOS manual that came with your computer. In general, you can't go far wrong if you use a name which starts with a letter, is no longer that eight character, and consists of only letters or numerals. If you should chose a name that already exists on you logged drive and directory, you will be warned, and given a chance to change it. (Saving a file under a name already in use will replace the old files data with the new, the old data being lost in the process.)

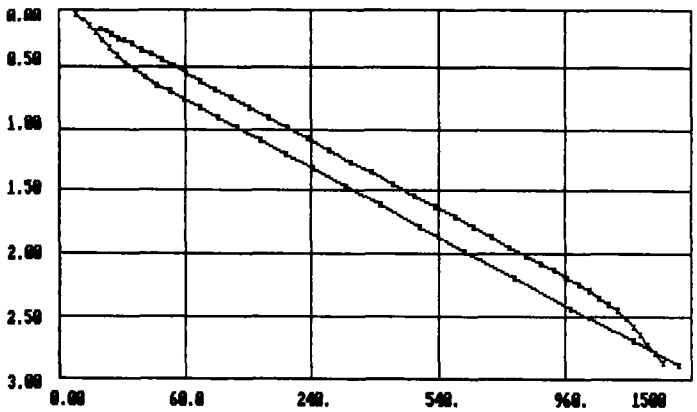
Figure 2.1



Note the downward curve due to the boundary, followed by the levelling out due to the leakage. A non bounded aquifer, under the same circumstances, would give a maximum drawdown of around 1.3m.

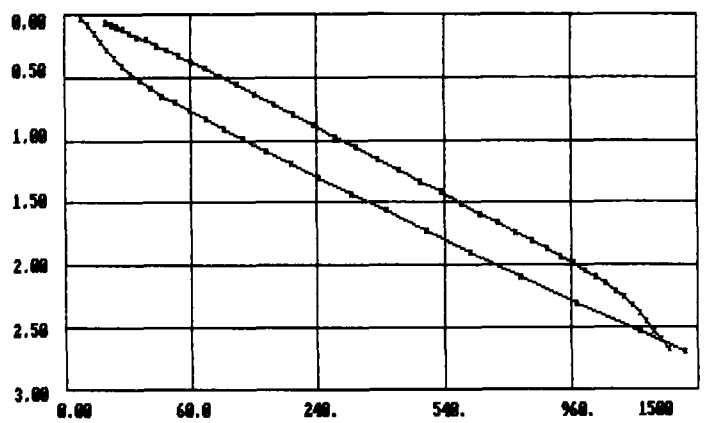
Some other examples of graphed simulations are given below. Try some more yourself, it is by experimenting that you will learn the capabilities and limitations of the program, and at the same time become comfortable with it's use.

Figure 2.2



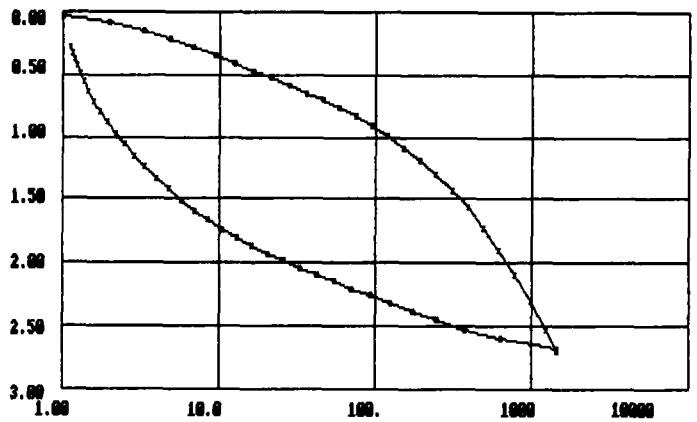
Drawdown in a confined strip aquifer. R=20, T=100, S=0.001, 2 steps, t1=1 day, Q1=300, t2=60 days, Q2=0, discharge to boundary = 100m, piezo. to boundary = 100m, width of strip = 200m. Lower curve is drawdown, upper is the recovery converted to root t minus root t1 by program DTDHA. Note that the drawdown curve is straight after about 20 minutes, and the recovery curve approaches the origin, but still has a significant distance to go even after 59 days recovery.

Figure 2.3



This simulation used the same data as for Figure 2.2 except that the strip aquifer has been replaced by a 'semistrip' aquifer with T2=5 (the transmissivity outside of the strip). Note that the drawdown line has a slight upward curve, and the earlier (right side) part of the recovery curve trends toward a point above the origin.

Figure 2.4



This is the same simulation as was graphed in Figure 2.3, but instead of using root t minus root t1 for the recovery data t/t1 was used (still using program DTDHA). Also, the plot is semilogarithmic rather than square root of time. Here the 'leaky' nature of the strip boundaries is betrayed by the slight straightening towards the end of the drawdown curve. A longer duration of discharge would show this more clearly.

3.4. A three part unconfined drawdown curve

If you wish to simulate discharge from an aquifer which behaves at early times as a confined aquifer, at later times as a leaky aquifer, and finally as an unconfined aquifer, then a single simulation is not possible in this program. (In practice, it seems, most 'unconfined' aquifers behave in this way, perhaps due to stratification and great variability in hydraulic conductivity at different levels. Delayed drainage produces a similar curve.) However, it is possible to simulate the first two stages of the drawdown curve with the leaky aquifer model, and then in a second program run, simulate the last, fully unconfined stage. The results of the two runs are stored on disk files, and then a separate program, JOINWTD, is used to pick out the appropriate parts of the two simulations, and connect them. This simulation gives a result somewhat similar to the Unconfined Well Function of Neuman (1975). Neuman's Well Function itself may be simulated by using program NEUMAN, described in Chapter 4 of this book. Note, however, that the two conceptual models are different, in that the former has a semiconfining layer, and the latter does not. Apparently, in Neuman's model, the delayed drawdown is due to the time taken for water to drain from the aquifer, while in the leaky aquifer model the delay is caused by the time taken for water to pass through the semiconfining layer.

4. PROGRAM DRAWDOWN, TECHNICAL COMMENTS

If you are content to run the program without studying the way it works, you need not look at the source code and you can skip this section. I suggest that even if you do not feel up to understanding the source code, you should at least read through the relevant parts of the following notes to develop an understanding of how the various simulations arrive at their results. In any case, always treat the output of the program with some scepticism. Remember, you are dealing with models of the real thing, and a model can never give exactly the same results as the real world, because all the variables are not taken into account. Also, the program was written by a human, and I can assure you that that human is fallible.

The source code for the main part of the program is in the file DRAWDOWN.PAS, and there are three subsidiary files called FIRST.SEG, LEAKFUN2.FUN, and SAVE.PRC. The first of these contains all those type, variable, and constant definitions, as well as basic procedures and functions, that are common to all the GW programs described in this book. The second contains the solution to the leaky artesian well function (originally developed in Fortran by Cobb et. al., 1982). The last 'Include' file contains the procedures used to save the data to a disk file, and a few simple, but useful, housekeeping routines.

All these files must be available to Turbo Pascal at the time of compilation of the program.

4.1. Some selected program variables

Factor; (actually a global constant) set approximately equal to the cube root of three. This is used to increment the times of the simulated discharge test drawdown readings. It may be reduced to shorten the interval between 'readings' or increased to lengthen the intervals. There is no real need to use a special number such as a root of two.

NumSteps; the number of discharge rate steps or stages.

OneDd; one drawdown, holds the current drawdown or the part of the current drawdown which has thus far been calculated.

PiezoToBoun; distance from the piezometer to the boundary.

PumpToBoun; distance from the discharging well to the boundary.

ThickAq; thickness of the saturated part of an unconfined aquifer before any discharge induced drawdown.

ThickTard; thickness of the aquitard.

Trans; transmissivity of the aquifer being pumped, and in which lies the piezometer.

Trans2; transmissivity beyond the boundary in a partially bounded aquifer.
VertCond; vertical hydraulic conductivity of the aquitard.

4.2. File DRAWDOWN.PAS, description by program section

4.2.1. The main part of program DRAWDOWN

This begins at line 224 and consists of a number of major sections which are executed sequentially.

Entry of information from keyboard Line 235

This section is entirely made up of two parts, multiple choice questions, and entry of numerical data. These parts are explained below.

End of entry of information from keyboard Line 372

Entry of answers to multiple choice questions Line 236

Function CapOptions is called repeatedly from this section, and Turbo Pascal's reverse type conversion is used to convert from the integer returned by CapOptions to the required scalar. See, for example, line 239. Variable AqType can have any one of three values, 'Confined', 'Unconfined', or 'Leaky' because it is defined as being of the type 'TypeOfAquifer'. Suppose that when presented with the choice of any one of the three aquifer types (line 238), the user pressed the 'C' key indicating that he was interested in a confined aquifer. This would cause function CapOptions to return a value of 1, as 'C' is the first capital letter in the string that was passed to it. In type conversion, the first value of type 'TypeOfAquifer' is considered by Turbo Pascal to be equivalent to 0 (so 'Confined'=0, 'Unconfined'=1, and 'Leaky'=2). Therefore if the variable AqType is to be given the correct value of 'Confined' by the assignment of line 239, the integer 1 returned by CapOptions must be reduced by 1 to become 0 (line 239).

End of multiple choice questions Line 259

Entry of numerical data Line 261

This section, while being of a fair length, is quite simple. The only complication here is that certain values are, or are not, required, depending on the answers that were given to the multiple choice questions. eg. if AqType is not equal to 'Leaky', then plainly a value for the thickness of the semiconfining layer is not required.

Note that if the user indicated that the simulation was to consist of more than one discharge stage, then the vector RateChange is filled in line 315. Each element (after the first) of this vector is given the amount of

the increase in the discharge rate from the previous to the current stage. The values in this vector will be used in the 'multiple pump' model described below.

End entry of numerical data Line 371

Start of solution section Line 374

This is the part of the program that supervises all simulations done by DRAWDOWN. It consists of two major sections, lines 376 to 387 deal with a single calculation of drawdown, while lines 388 to 436 deal with the case of a time-series set of drawdowns.

The 'single' solution section needs no explanation so I will move on to the 'series' solution section. As the user may require a simulation consisting of more than one discharge rate stage, there is an outer loop (lines 391 to 424) associated with the integer variable I which counts discharge stages.

Recall that times used in calculations are in days. Variables TimeInc (Time increment), and OneTime are both set to one minute in lines 393 and 394 if the program is considering the first stage (I=1). Variable OneTime is used to hold the current time, being the time since the beginning of the first discharge stage. In line 395 we reach the beginning of a loop imbedded in the discharge stage counting loop, this loop causes drawdowns to be calculated while the value of the current time is not greater than that entered (by the user) as the end of the current stage.

MULTIPLE PUMP MODEL

To understand the operation of this loop it is first necessary to understand the concept of a multistage discharge test being simulated by (imaginary) multiple pumps. As an example, if for the first stage Q=100 (Q, the discharge rate, equals 100 kilolitres per day), and at t=30 (a time of 30 minutes from the beginning of the test) Q is increased to 150, then the resultant drawdown will be the same as if a second pump, operating at Q=50 began at 30 minutes while the original pump continued at the same rate (100 kl/day). In this way, any number of discharge rates can, in theory, be accommodated. The drawdown, negative or positive, due to each being summed to obtain the drawdown due to the cumulative effects of all the stages. In fact the program is set up for a maximum of 10 discharge stages (line 25), although this could very easily be greatly increased.

4.2.2. Procedures and functions of file DRAWDOWN.PAS

Boundary subfunction Line 82

Purpose: to calculate the drawdown due to one image well, simulating a single straight line boundary.

This function is very similar to function Unbounded which is described below. The apparent distance from the image well to the piezometer must be calculated (line 88) before evaluating the drawdown due to that image well. Note that in this case, the absolute discharge rate of the image well is the same as that of the real well, it is only the sign of that rate which may differ depending on the type of boundary.

Note that line 92 treats the unconfined case in the same way as the confined case, if the aquifer is unconfined then the drawdown will be adjusted later by use of the Jacob correction.

Called by: parent function CalcDrawdown.

Calls: functions WellFunc and Leakfunc.

CalcDrawdown function Line 58

Purpose: to calculate the drawdown in the piezometer at one given time, and for one discharge rate.

This is the most important function in program DRAWDOWN. It is this function which, with it's sub functions and procedure, calculates the drawdown at a given time and discharge rate; the other variables having previously been fixed. Flow through this part of the program is primarily controlled by the 'aquifer extent'; ie. whether or not the aquifer has boundaries, and if so, then what type of boundaries they are.

The function has three sections.

1/ The simplest case, that of the unbounded aquifer, is dealt with in the section from line 159 to 162 by a call to subfunction Unbounded.

2/ The case of either a bounded or semibounded aquifer is handled by the code from line 158 to 175. First the drawdown due to the discharge well itself is calculated by a call to function Unbounded in line 166, then the drawdown due to the image well is calculated by calls to function Boundary or SemiBoundary, as appropriate. (The total drawdown in a bounded aquifer is calculated as the sum of the drawdown that would be measured in an infinite aquifer due to the real discharging well, plus the drawdown due to a discharging or recharging image well.)

3/ A little more complex is the strip aquifer (or semistrip aquifer case), lines 177 to 192. Here there are three drawdown components to be summed, the drawdown due to the real discharge well, and those due to all the

image wells on each of the two sides of the strip. The value in the variable Toggle controls which side of the strip is calculated by the subfunction SideOfStrip.

Called by: the main part of program DRAWDOWN.

Calls: subfunctions Unbounded, Boundary, and SemiBoundary; and subprocedure SideOfStrip.

SemiBoundary subfunction Line 108

Purpose: to calculate the drawdown due to one image well simulating a single straight line boundary between the part of the aquifer containing the discharge well and piezometer and another part of the aquifer having a different transmissivity.

The difference between this and subfunction Boundary is that here the image well has a discharge rate of different magnitude, and perhaps a different sign, to that of the real well. The calculation of drawdown due to a 'partial', or 'semi' boundary is explained under the section 'Simulation of partial boundaries', earlier in this chapter.

Called by: the parent function CalcDrawdown.

Calls: function WellFunc (this program is not written to deal with partial boundaries in leaky of unconfined aquifers).

SideOfStrip subprocedure Line 122

Purpose: to calculate the drawdown due to all image wells on one side of a strip aquifer, whether that strip is a fully bounded strip, or a partially bounded strip.

The distance to a particular image well is a function of three variables, where the first variable is either the distance from the piezometer to the boundary, or the negative of that distance (depending on which side of the strip is concerned), the second variable is a multiplier of the strip width, and the third variable alternates (with each successive image well) between the distance from the discharging well to the boundary, and the negative of that distance. The first variable is fixed before procedure SideOfStrip is called (lines 184, and 188 of the main part of procedure CalcDrawdown).

Figure 2.5 shows the pattern of image wells that will be produced given data entered into program DRAWDOWN as follows:

distance from discharging well to piezometer, 45m;
" " " " " boundary, 9m;
" " piezometer to boundary, 23m;

width of strip aquifer, 30m;

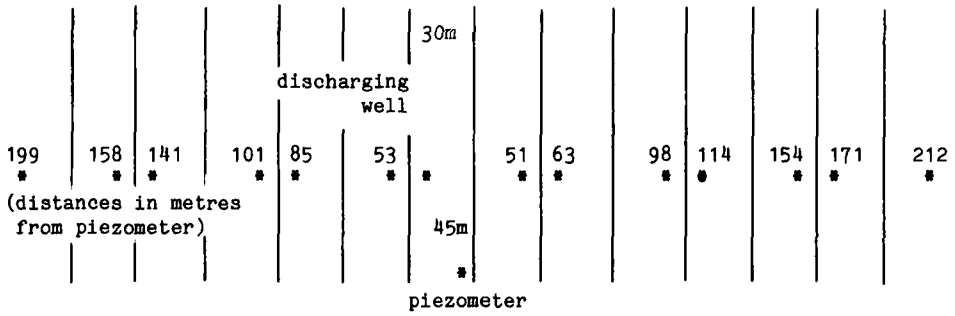
transmissivity, 100m^2/day;

storage coefficient, 0.001;

discharge rate, 1000m3/day; (does not effect radius of influence)

time, 20min.

Figure 2.5



The distance between the discharging well and the boundary to the left of it is 9m. The piezometer is 23m from the same boundary.

The distances of the image wells on the first (left) side of the aquifer are calculated (line 139) by the series;

R_{i1}=sqrt[(Rpb+0*W+Rdb)^2 + R'2], (= 53.41) (2.3)

R_{i2}=sqrt[(Rpb+2*W-Rdb)^2 + R'2], (= 85.47)

R_{i3}=sqrt[(Rpb+2*W+Rdb)^2 + R'2], (=101.45)

R_{i4}=sqrt[(Rpb+4*W-Rdb)^2 + R'2], (=140.66)

R_{i5}=sqrt[(Rpb+4*W+Rdb)^2 + R'2], (=157.90)

R_{i6}=sqrt[(Rpb+6*W-Rdb)^2 + R'2], (=198.66)

. . ..

The second (right) side uses the series;

R_{i7} =sqrt[(-Rpb+2*W-Rdb)^2 + R'2], (= 51.12)

R_{i8} =sqrt[(-Rpb+2*W+Rdb)^2 + R'2], (= 62.81)

R_{i9} =sqrt[(-Rpb+4*W-Rdb)^2 + R'2], (= 97.84)

R_{i10}=sqrt[(-Rpb+4*W+Rdb)^2 + R'2], (=114.30)

R_{i11}=sqrt[(-Rpb+6*W-Rdb)^2 + R'2], (=154.06)

R_{i12}=sqrt[(-Rpb+6*W+Rdb)^2 + R'2], (=171.42)

R_{i13}=sqrt[(-Rpb+8*W-Rdb)^2 + R'2], (=212.35)

. . ..

where R_{pb} is the distance from the piezometer to a boundary,

R_{db} is the distance from the discharging well to the same boundary,

W is the width of the strip aquifer,

R_{i1} is the distance to the first image well,

sqrt represents the square root operation,

and R' is the component in the direction along the strip of the distance between the piezometer and the discharging well, and hence also between the piezometer and all the image wells.

The value of R' is calculated (line 182) by the equation:

$$R'=(R2-(R_{db}-R_{pb})2)1/2 \text{ (Pythagoras)} \qquad (2.4)$$

The program takes a small short cut in carrying out these operations. Instead of using R', it is actually R'2 which is used, so avoiding the extraction of the square root in equation 2.4, and the squaring of R' in the equation series 2.3.

If the aquifer has been specified as a semistrip, (a strip of one transmissivity in contact with an infinite aquifer having a different transmissivity) then with each successive reflection, line 144 calculates a value for the discharge rate of the new image well by use of equation 2.2.

If the drawdown is being calculated for a single specified time then the calculated distances of the image wells, and the cumulative drawdown due to all image wells on the current side of the strip, will all be displayed (lines 152 and 153). The image wells for simulation of a strip aquifer go on for an infinite distance, so some method must be used to decide when to stop the calculations. Calculations are terminated when either the drawdown due to the current image well is less than one ten thousandth of the cumulative drawdown, or when it is less than one tenth of a millimetre (line 155). If greater accuracy is required, then the value of the constant 'Tolerance' (line 32) may be increased, and the value of the literal '0.0001' on line 155 may be decreased.

Called by: the parent function CalcDrawdown.

Unbounded subfunction Line 66

Purpose: To calculate the drawdown in an infinite aquifer.

The calculation is performed either by referring to the Well Function (line 73), in the case of a simple confined aquifer, or the Leaky Artesian Well Function (line 78), in the case of a leaky aquifer. In the first case it is the Theis equation (Theis, 1935; Bouwer, 1978; Marino and Luthin, 1982,

pp 249-252; etc.) that is used, in the second case, Hantush's solution (Hantush, 1960, 1964; Marino and Luthin, 1982, pp 266-275; etc.) is used.

UnconfinedDd function Line 195

Purpose: to convert the drawdown calculated for a confined aquifer to that which would be expected for an unconfined aquifer.

Greater drawdown must be expected in an unconfined aquifer than in a confined aquifer (assuming the specific yield equals storage coefficient) because as the water table is drawn down, so the effective thickness of the aquifer, and hence the transmissivity, is reduced. To calculate the unconfined drawdown the inverse of the Cooper-Jacob correction is applied in line 203. The Cooper Jacob approximation is given by the equation:

$$s_1 = s_2 - \frac{s_2^2}{2D} \qquad (2.5)$$

where s_1 is the drawdown that would be observed in a confined aquifer having a storage coefficient equal to the specific yield of the unconfined aquifer. (Since the aquifer is confined, then transmissivity is not reduced by drawdown as it is the piezometric surface only that is drawn down. Hence s_1 is less than s_2.)

s_2 is the observed drawdown in the unconfined aquifer,

and D is the saturated thickness of the unconfined aquifer.

The inverse of this is:

$$s_2 = D(1-\mathrm{sqrt}(1-(2s_1/D)) \qquad (2.6)$$

where s_1 is now the drawdown to be expected from the confined aquifer,

and s_2 is the drawdown to be expected from the same set of causes, in an unconfined aquifer. (ie. given the same transmissivity, discharge rate, etc.)

It can be seen from the above equation that if the calculated confined drawdown is greater than half the aquifer thickness, then the conversion is impossible (square root of a negative number) and an appropriate message is displayed by line 200.

Called by: the main part of DRAWDOWN.

VideoOutput procedure Line 206

Purpose: to display time-drawdown-discharge rate data as they are calculated.

Called by: the main part of DRAWDOWN.

WellFunc function Line 35

Purpose: to calculate the value of the Theis well function.

The method used is Huntoon's (1980) polynomial approximations. One or other of Huntoon's approximations are used depending on whether the value of u is greater or less than one. As the equations can plainly be seen in the program there seems no point in repeating them here. Huntoon did present one polynomial approximation having higher accuracy than one of these, but as the accuracy of the approximations as they are is at least three or four significant decimal digits there seems no point in increasing the complexity and computational requirements of the program.

4.3. Procedures and functions of file LEAKFUN2.FUN

Note that this file (and this version of function LeakFunc) is slightly different to file LEAKFUNC.FUN which is used in the compilation of program ANALYZE. That version contains some variables which are used in the curve fitting algorithm, LeakyFit.

4.3.1. Definition of inverse leakage coefficient

The term called inverse leakage coefficient (Lc) in these notes has the dimension of length. The equation linking this to vertical hydraulic conductivity of the aquitard, K_v, is;

$$K_v=(T*b_{cl})/Lc^2 \qquad (2.7)$$

where T is the transmissivity of the aquifer,

b_{cl} is the thickness of the semiconfining layer. (This equation is from Cobb et al., 1982, p35.)

Kinzelbach, (1986) p8-10 also uses this definition.

It was felt necessary to include this section in the notes because the variable "Leakage Coefficient" does not seem to be consistently defined in various references. The equation above can serve to define the variable as used in these notes, and in this program.

4.3.2. Definition of RB

The variable named RB in these notes is sometimes symbolized as r/B (where r is the distance from the discharging well to the piezometer in which the drawdown is measured) in other works (eg. Marino and Luthin, 1982). It is defined as being equal to the product of r and Lc in Cobb et. al. (1982). Marino and Luthin use:

$$B=\mathrm{sqrt}(T^{*}b_{cl}/Kv) \qquad (2.8)$$

Consequently, it can be seen that the inverse leakage coefficient of this program is identical to Marino and Luthin's B.

Bessel1 subfunction Line 8

Purpose: to evaluate the Bessel function of the first kind and zero order.

The first kind of Bessel function of zero order (usually symbolised by I_0) is required in this program for evaluation of the second kind of Bessel function in cases when RB is not greater than two (see above for the definition of RB), and is also used in the evaluation of the leaky artesian well function.

The algorithm used to solve the Bessel function is a polynomial approximation adapted from Cobb et. al. (1982). This polynomial called for a value (RB/3.75) to be raised to a number of integral powers. As exponentiation requires the use of the natural log (ln) function, and that is comparatively very slow, the raising to powers has been done by repeated multiplication instead. Thus RB3fourth is the fourth power of RB/3.75, etc.

Called by: subfunction Bessel2, and the main part of the parent function LeakFunc.

Bessel2 subfunction Line 45

Purpose: to evaluate the Bessel function of the second kind and zero order.

The second kind of Bessel function of zero order (usually symbolised by K_0) is required for evaluation of the leaky artesian well function. Again, the function is evaluated from a polynomial approximation adapted from Cobb et. al. (1982). It is very similar in operation to the subfunction Bessel1.

Called by: the main part of the parent function LeakFunc.

Calls: subfunction Bessel1.

IntFact subfunction Line 86

Purpose: to evaluate small, integral, factorials.

The factorial of a negative number is undefined, so any attempt to pass a negative number to this function is answered with an error message in line 98. Similarly, if the function were to attempt to evaluate the factorial of a number greater than seven, then integer overflow would occur. This is handled by the error message in line 94.

As there are only eight possible values that may be calculated (0! to 7!) perhaps a look-up table might have resulted in faster operation?

Called by: subfunction SUMURB.

IntPower subfunction Line 144

Purpose: to raise a real number to an integral power.

As mentioned above, the direct calculation of exponents requires the use of the natural logarithm function, and this is rather slow. When the required power is relatively small, less than about 20, iterative multiplication is probably faster. (The break even number would depend upon whether or not a maths co-processor is available.)

Called by: subfunction SUMURB.

LeakFunc function Line 1

Purpose: to evaluate the leaky artesian well function.

This is a complex function which calls on several other embedded functions. The set of routines used here were derived from a Fortran program written by Cobb et. al. (1982). Functions called, directly or indirectly, by LeakFunc are:

1/ the Bessel function of the first kind, zero order, Bessel1;

2/ the Bessel function of the second kind, zero order, Bessel2;

2/ the Theis Well Function, WellFunc;

3/ function IntFact, a simple solution for factorials;

4/ function Ssurb;

5/ function IntPower. This is a fast function for raising a real number to an integer power;

6/ function Sumurb.

All the called functions mentioned above are present in the second include file of program DRAWDOWN, file LEAKFUN2.FUN; except for function WellFunc which is in the main file, DRAWDOWN.PAS.

SSURB subfunction Line 107

Refer to Cobb et. al. (1982) for an explanation of the derivation and operation of this algorithm.

SUMURB subfunction Line 155

Refer to Cobb et. al. (1982) for an explanation of the derivation and operation of this algorithm.

5. PROGRAM DRAWDOWN, KEY LINES

5.1. File Drawdown.Pas, key lines

```
  5 {#}{$I First.seg}
 35 Function WellFunc {Polynomial approx. to the Theis Well Function}
 54 {#}{$I Leakfun2.fun}
 56 {#}{$I Save.prc}
 58 Function CalcDrawdown {Calculate one drawdown value, given aq. config.}
 66   function Unbounded {Drawdown in confined or leaky confined aquifer}
 82   function Boundary {Calculates the drawdown due to one boundary,
107   function SemiBoundary {Calculates the drawdown due to one semiboundary;
121     procedure SideOfStrip; {Drawdown due to all image wells one side strip
          aq.}
157 begin {# main part of CalcDrawdown}
192 end; {# Function CalcDrawdown}
194 Function UnconfinedDd {Convert confined to unconfined, inverse Jacobs}
205 Procedure VideoOutput {Display data as it is calculated}
223 begin {# main program}
234     {#----- Entry of information from keyboard -----#}
235     {#----- Entry of answers to multiple choice questions ----#}
258     {#----- End of multiple choice questions -----#}
260     {#----- Entry of numerical data -----#}
370     {#----- End entry of numerical data -----#}
371     {#----- End of entry of information from keyboard -----#}
373     {#----- Start of solution section -----#}
442 end. {# of Program Drawdown.Pas}
```

5.2. Include file LeakFun2.Fun, key lines

```
  1 Function LeakFunc {Functions for solution of leaky artesian well
    function.
  8 Function Bessel1 {Bessel function of the first kind and zero order}
 45 Function Bessel2 {Bessel function of the second kind and zero order}
 86 Function IntFact {Solution for integral, small factorials}
107 Function SSURB {Solution of an indefinite integral}
144 Function INTPOWER {Raise a real, A to an integral power, B.}
155 Function SUMURB(u,RB: real): real;
179 begin {# main part of Leakfunc}
```

6. DRAWDOWN, PROGRAM LISTING

6.1. File DRAWDOWN.PAS, listing

```
  1 Program DRAWDOWN_PAS; {For the calculation of drawdowns in various
    aquifers}
  2
  3 {$R+}
  4
  5 {#}{$I First.seg}
  6
```

```
 7 type
 8   Calc=(Single, Series);
 9   TypeOfAquifer=(Confined, Unconfined, Leaky);
10   TypeOfBoundary=(Dischge, Recharge);
11   ExtentOfAquifer=(Infinite, Bounded, Strip, SemiBounded, SemiStrip);
12   TimeUnits=(Minute, Day);
13 var
14   TimeError: boolean;
15   Again, NumSteps: byte;
16   Ch: Char;
17   NumData, NumOfDdLogs: integer;
18   PumpCount, RecNum: integer;
19   Distance: real;
20   OneDd, OneRate, TempRate, Storage, Trans, Trans2, OneTime: real;
21   VertCond, ThickTard, StripWidth, PumpToBoun, PiezoToBoun: real;
22   ThickAq, TimeInc, TimeAtRate: real;
23   TestType: Test; WellType: Well;
24   TimeVec, DdVec, RateVec: MainVec;
25   Finish, StepRate, RateChng: array[1..10] of real;
26   TypeCalc: Calc;
27   AqType: TypeOfAquifer;
28   BounType: TypeOfBoundary;
29   AqExtent: ExtentOfAquifer;
30   TimeUnit: TimeUnits;
31 const
32   Tolerance=10000;
33   Factor=1.2599211; {Cube root of 2}
34
35 Function WellFunc {Polynomial approx. to the Theis Well Function}
36 (u:real): Real;
37 const
38   C0=-0.57721566; C1=0.99999193; C2=-0.24991055; C3=0.05519968;
39   C4=-0.00976004; C5=0.00107857; C6=0.250621;    C7=2.334733;
40   C8=1.681534;    C9=3.330657;
41 var
42   u2,u3:real;
43 begin
44   if u<=0 then u:=1e-35;
45   if u>80 then Wellfunc:=0
46   else begin
47     U2:=U*U; U3:=U2*U;
48   if u<1
49     then WellFunc:=-ln(u)+C0+C1*U+C2*U2+C3*U3+C4*U2*U2+C5*U2*U3
50     else WellFunc:=1/(U*EXP(U))*(C6+C7*U+U2)/(C8+C9*U+U2)
51   end {else WellFunc=}
52 end; {Function WellFunc}
53
54 {#}{$I Leakfun2.fun}
55
56 {#}{$I Save.prc}
57
58 Function CalcDrawdown {Calculate one drawdown value, given aq. config.}
59 (OneTime, OneRate:real):real;
60 var
61   OneDd: real;
62   I, J, Num, NumStrips, Toggle: integer;
63   Temp, TempDrawdown: real;
64   AccStrip, FirstTerm, ThirdTerm: real;
65
```

```
66   function Unbounded {Drawdown in confined or leaky confined aquifer}
67   (Distance, OneTime, OneRate: real):real;
68   var u, Rb: real;
69   begin
70     if (AqType=Confined) or (AqType=Unconfined)
71     then begin
72       u:=sqr(Distance)*Storage/(4*Trans*OneTime);
73       Unbounded:=OneRate*Wellfunc(u)/(4*Pi*Trans);
74     end {Confined or Unconfined aquifer case}
75     else begin
76       u:=sqr(Distance)*Storage/(4*Trans*OneTime);
77       Rb:=Distance/sqrt(Trans/(VertCond/ThickTard));
78       Unbounded:=OneRate*Leakfunc(u,Rb)/(4*Pi*Trans);
79     end; {Leaky aquifer case}
80   end; {sub function Unbounded}
81
82   function Boundary {Calculates the drawdown due to one boundary,
83     confined, or leaky confined}
84   (OneTime, OneRate, BounDist, BounDist2: real):real;
85   var ImageDist, u, Rb: real;
86   begin
87     ImageDist:=
88      sqrt(sqr(BounDist+BounDist2)+Sqr(Distance)-
   sqr(BounDist-BounDist2));
89     if TypeCalc=Single then
90       writeln('Distance from piezometer to boundary is '
91         ,ImageDist:10:1,'m');
92     if (AqType=Confined) or (AqType=Unconfined)
93     then begin
94       u:=sqr(ImageDist)*Storage/(4*Trans*OneTime);
95       if BounType=Dischge
96         then Boundary:=OneRate*WellFunc(u)/(4*Pi*Trans)
97         else Boundary:=-OneRate*WellFunc(u)/(4*Pi*Trans);
98     end {Confined or Unconfined aquifer case}
99     else begin
100      u:=sqr(ImageDist)*Storage/(4*Trans*OneTime);
101      Rb:=ImageDist/sqrt(Trans/(VertCond/ThickTard));
102      if BounType=Dischge
103        then Boundary:=OneRate*LeakFunc(u,Rb)/(4*Pi*Trans)
104        else Boundary:=-OneRate*LeakFunc(u,Rb)/(4*Pi*Trans);
105    end; {Leaky aquifer case}
106  end; {sub function Boundary}
107
108  function SemiBoundary {Calculates the drawdown due to one
   semiboundary;
109    ie. one image well having a different Q to the real well}
110  (OneTime, OneRate, BounDist, BounDist2: real):real;
111  var ImageDist, u: real;
112  begin
113    ImageDist:=
114      sqrt(sqr(BounDist+BounDist2)+Sqr(Distance)-
   sqr(BounDist-BounDist2));
115    u:=sqr(ImageDist)*Storage/(4*Trans*OneTime);
116    TempRate:=OneRate*-(Trans2/Trans-1)/(Trans2/trans+1);
117    if TypeCalc=Single
118      then writeln('Effective Q of image well =',TempRate:9:2,'
   m^/day');
119    SemiBoundary:=TempRate*WellFunc(u)/(4*Pi*Trans)
120  end; {sub function SemiBoundary}
```

```
121
122   procedure SideOfStrip; {Drawdown due to all image wells one side strip
    aq.}
123   var ImageDist: real;
124   begin
125     NumStrips:=0; TempDrawdown:=0;
126     if AqExtent=SemiStrip then TempRate:=OneRate;
127     repeat {Begin calculating distances to boundaries, and images of
128               boundaries, on one side.}
129       begin
130         if Toggle=1 then
131         begin
132           NumStrips:=NumStrips+2; ThirdTerm:=-PumpToBoun; Toggle:=0;
133         end
134         else begin
135           ThirdTerm:=PumpToBoun; Toggle:=1
136         end; {else}
137       end; {if}
138       ImageDist
139         :=sqrt(sqr(FirstTerm+NumStrips*StripWidth+ThirdTerm)+AccStrip);
140       case AqExtent of
141         Strip: Temp:=Unbounded(ImageDist, OneTime, OneRate);
142         SemiStrip:
143         begin
144           TempRate:=TempRate*-(Trans2/Trans-1)/(Trans2/trans+1);
145           if TypeCalc=Single then write('Effec. Q =',TempRate:9:2,'
    m^3/day');
146           Temp:=Unbounded(ImageDist, OneTime, TempRate);
147         end; {of SemiStrip case}
148       end; {of cases}
149       TempDrawdown:=TempDrawdown + Temp;
150       if TypeCalc=Single
151       then begin
152         write('  R, image well =',ImageDist:9:2,' m');
153         writeln('  Cumul. dd. =',tempdrawdown:7:3,' m');
154       end; {if then}
155     until ((abs(Temp) < abs(OneDd/Tolerance)) or (abs(Temp)<0.0001));
156   end; {sub procedure SideOfStrip}
157
158 begin {# main part of CalcDrawdown}
159   if AqExtent=Infinite
160   then begin
161     CalcDrawdown:=Unbounded(Distance, OneTime, OneRate);
162   end; {Infinite aquifer case}
163
164   if (AqExtent=Bounded) or (AqExtent=SemiBounded)
165   then begin
166     OneDd:=Unbounded(Distance, OneTime, OneRate);
167     if TypeCalc=Single then
168       writeln('Drawdown due to discharge well is ',OneDd:7:3,' m');
169     Case AqExtent of Bounded:
170       OneDd:=OneDd+Boundary(OneTime, OneRate, PumpToBoun, PiezoToBoun);
171       SemiBounded:
172       OneDd:=OneDd+SemiBoundary(OneTime, OneRate, PumpToBoun,
    PiezoToBoun);
173     end; {of cases}
174     CalcDrawdown:=OneDd;
175   end; {Bounded aquifer case}
176
```

```
177   if (AqExtent=Strip) or (AqExtent=SemiStrip)
178   then begin
179     OneDd:=Unbounded(Distance, OneTime, OneRate); NumStrips:=0;
180     if TypeCalc=Single then
181       writeln('Drawdown due to discharge well is ',OneDd:7:3,' m');
182     AccStrip:=sqr(Distance)-sqr(PumpToBoun-PiezoToBoun); {Square of
    distance
183             component accross aquifer}
184     FirstTerm:=PiezoToBoun; Toggle:=0;
185     if TypeCalc=Single then writeln('First side of strip');
186     SideOfStrip; {Calculate drawdown due to image wells on first side}
187     OneDd:=OneDd+TempDrawdown;
188     FirstTerm:=-PiezoToBoun; Toggle:=1;
189     if TypeCalc=Single then writeln('Second side of strip');
190     SideOfStrip; {Calculate drawdown due to image wells on second side}
191     CalcDrawdown:=OneDd+TempDrawdown;
192   end; {Strip aquifer case}
193 end; {# Function CalcDrawdown}
194
195 Function UnconfinedDd {Convert confined to unconfined, inverse Jacobs}
196 (ConfinedDd: real):real;
197 begin
198   if ConfinedDd>=0.5*ThickAq
199   then begin
200     writeln('ERROR: Unconfined aquifer dewatered.');
201     UnconfinedDd:=ThickAq;
202   end
203   else UnconfinedDd:=ThickAq*(1-sqrt(1-(2*ConfinedDd)/ThickAq));
204 end; {Function UnconfinedDd}
205
206 Procedure VideoOutput {Display data as it is calculated}
207 (I, NumOfPumps: integer; Time, Drawdown, StepRate: real);
208 var
209   Ch: char;
210 const
211   Head='No.       min.     days     Drawdown    Rate     Pumps';
212 begin
213   if I=1 then writeln(Head);
214   writeln(I:3,'  ',Time*1440:8:0,'  ',Time:9:4,'  ',Drawdown:9:3,
215     '  ',StepRate:9:2,'  ',NumOfPumps:3);
216   if (I div 20)*20=I then
217   begin
218     writeln('Press any key to continue.');
219     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
220     writeln(Head);
221   end;
222 end; {Procedure VideoOutput}
223
224 begin {# main program}
225   FileName:='';
226   repeat
227     Ch:=' ';
228     ClrScr;
229     TextColor(Green);
230     Gotoxy(5,0);
231     writeln('            DRAWDOWN');
232     writeln('Calculation of drawdowns in various aquifer
    configurations.');
233     writeln;
```

```
234
235     {#----- Entry of information from keyboard -----#}
236     {#----- Entry of answers to multiple choice questions ----#}
237     write('What type of aquifer?  ');
238     Long:='Confined, Unconfined, Leaky? ';
239     AqType:=TypeOfAquifer(CapOptions(Long)-1);
240
241     writeln('Extent of aquifer; ');
242     if AqType=Confined
243       then Long:='Infinite, One boundary, Strip, sEmi bounded, seMi
    strip? '
244       else Long:='Infinite, One boundary, Strip? ';
245     AqExtent:=ExtentOfAquifer(CapOptions(Long)-1);
246
247     if AqExtent=Bounded then
248     begin
249       Write('Type of boundary; '); Long:='Discharge, or Recharge? ';
250       BounType:=TypeOfBoundary(CapOptions(Long)-1);
251     end;
252
253     write('Do you want calculations for ');
254     Long:='One time, or a Series of times?';
255     TypeCalc:=Calc(CapOptions(Long)-1);
256
257     write('What time unit, ');
258     TimeUnit:=TimeUnits(CapOptions('Minute, or Day?')-1);
259     {#----- End of multiple choice questions -----#}
260
261     {#----- Entry of numerical data -----#}
262     writeln;
263     write('Distance of piezometer from discharging well? (m) ');
264     Distance:=ReadReal(1);
265     write('Transmissivity? (m*m/day) ');
266     Trans:=ReadReal(1);
267     if (AqExtent=SemiBounded) or (AqExtent=SemiStrip)
268     then begin
269       write('What transmissivity for the area beyond the boundary? ');
270       Trans2:=ReadReal(1);
271     end;
272     write('Storage coefficient? '); Storage:=ReadReal(1);
273     if TypeCalc=Single then
274       begin
275         write('Discharge rate? (m^3/day) ');
276         OneRate:=ReadReal(1);
277       end;
278     Case TypeCalc of
279       Single: begin
280         write('Time from beginning of discharge ');
281         if TimeUnit=Minute then write('(minutes)? ') else write('(days)?
    ');
282         OneTime:=ReadReal(1);
283         if TimeUnit=Minute then OneTime:=OneTime/1440;
284       end; {Case 1}
285       Series: begin
286         write('Number of steps ');
287         NumSteps:=ReadInt(1);
288         writeln('All times must be measured from the beginning '
289           ,'of the test.');
290         write('Finishing time for step 1, ');
```

```
291         if TimeUnit=Minute then write('(minutes)? ') else write('(days)?
    ');
292         Finish[1]:=ReadReal(1);
293         if TimeUnit=Minute then Finish[1]:=Finish[1]/1440;
294         write('Discharge rate for step 1, (m^3/day) ');
295         StepRate[1]:=ReadReal(1);
296         RateChng[1]:=StepRate[1];
297         if NumSteps>1 then for I:=2 to NumSteps do
298         begin
299           repeat
300             repeat
301               TimeError:=false;
302               write('Finishing time for step ',I,', ');
303               if TimeUnit=Minute then write('(minutes)? ')
304                 else write('(days)? ');
305               Finish[I]:=ReadReal(1);
306               if TimeUnit=Minute then Finish[I]:=Finish[I]/1440;
307               if Finish[I]<=Finish[I-1] then
308               begin
309                 writeln('Error: This finishing time <= previous!');
310                 TimeError:=true;
311               end;
312             until not TimeError;
313             write('Discharge rate for step ',I,', (m^3/day) ');
314             StepRate[I]:=ReadReal(1);
315             RateChng[I]:=StepRate[I]-StepRate[I-1];
316           until Finish[I-1]<Finish[I];
317         end; {Entry of steps}
318       end; {Case 2}
319     end; {Cases}
320
321     case AqType of
322       Leaky: begin
323         write('Vertical hydraulic conductivity of aquitard (m/day) ');
324         VertCond:=ReadReal(1);
325         write('Thickness of aquitard (m) ');
326         ThickTard:=ReadReal(1);
327       end; {Leaky aquifer case}
328       Unconfined: begin
329         write('Thickness of saturated part of aquifer (m) ');
330         ThickAq:=ReadReal(1);
331       end; {Unconfined case}
332     end; {cases}
333
334     if (AqExtent=Bounded) or (AqExtent=SemiBounded)
335     then begin
336       write('Distance from discharging well to boundary (m) ');
337       PumpToBoun:=ReadReal(1);
338       write('Distance from piezometer to boundary (m) ');
339       PiezoToBoun:=ReadReal(1);
340       while abs(PumpToBoun-PiezoToBoun)>Distance do {Error control}
341       begin
342         writeln('Impossible geometry of discharge well, piezometer, ',
343         'and boundary!');
344         write('Distance from piezometer to boundary (m) ');
345         PiezoToBoun:=ReadReal(1);
346       end; {Error control}
347     end; {Single boundary}
348
```

```
    if (AqExtent=Strip) or (AqExtent=SemiStrip)
    then begin
      write('Distance from discharging well to boundary (m) ');
      PumpToBoun:=ReadReal(1);
      write('Distance from piezometer to the same boundary (m) ');
      PiezoToBoun:=ReadReal(1);
      while abs(PumpToBoun-PiezoToBoun)>Distance do {Error control}
      begin
        writeln('Impossible geometry of discharge well, piezometer, ',
        'and boundary!');
        write('Distance from piezometer to boundary (m) ');
        PiezoToBoun:=ReadReal(1);
      end; {Error control}
      write('Width of strip (m) ');
      StripWidth:=ReadReal(1);
      while (PumpToBoun>StripWidth) or (PiezoToBoun>StripWidth) do
      begin
        writeln('Strip is impossibly narrow!');
        write('Width of strip (m) ');
        StripWidth:=ReadReal(1);
      end; {Error control}
    end; {Strip aquifer}
    {#----- End entry of numerical data -----#}
    {#----- End of entry of information from keyboard -----#}

    {#----- Start of solution section -----#}
    Case TypeCalc of
      Single: {One time calculation of drawdown}
      begin
        OneDd:=CalcDrawdown(OneTime,OneRate);
        if AqType=Unconfined
        then begin
          writeln('Calculated confined drawdown is ',OneDd:7:3,' m');
          writeln('Converting to unconfined using the inverse Jacob ',
          'correction:');
          OneDd:=UnconfinedDd(OneDd);
        end;
        writeln('Final calculated drawdown is ',OneDd:7:3,' m');
      end; {Single calculation case}
      Series: {Stepped rate discharge test simulation}
      begin
        RecNum:=1;
        for I:=1 to NumSteps do
        begin
          TimeInc:=1/1440;
          if I=1 then OneTime:=1/1440 else OneTime:=Finish[I-1];
          while OneTime<=Finish[I] do
          begin
            OneDd:=0;
            PumpCount:=I;
            if I>1 then if OneTime=Finish[I-1] then PumpCount:=I-1;
            For J:=1 to PumpCount do
            begin
              OneRate:=RateChng[J];
              if J=1 then TimeAtRate:=OneTime
                else TimeAtRate:=OneTime-Finish[J-1];
              if TimeAtRate>0 then
                OneDd:=OneDd+CalcDrawdown(TimeAtRate, OneRate);
            end; {For J}
```

```
408             if AqType=Unconfined
409             then begin
410               OneDd:=UnconfinedDd(OneDd);
411               if OneDd>ThickAq then writeln('Error: Aquifer
    dewatered!');
412             end;
413             TimeVec[RecNum]:=OneTime; DdVec[RecNum]:=OneDd;
414             RateVec[RecNum]:=StepRate[I];
415             if I>1 then if OneTime=Finish[I-1]
416               then RateVec[RecNum]:=StepRate[I-1];
417             VideoOutput(RecNum,J,OneTime,OneDd,RateVec[RecNum]);
418             RecNum:=RecNum+1;
419             OneTime:=OneTime+TimeInc; TimeInc:=TimeInc*Factor;
420             if (OneTime>Finish[NumSteps]) and
421             (OneTime<Finish[NumSteps]+TimeInc/Factor)
422               then OneTime:=Finish[NumSteps];
423           end; {while}
424         end; {For I}
425         NumData:=RecNum-1;
426         Long:='Save data to disk, or Finish?';
427         I:=CapOptions(Long); if I=1 then
428         begin
429           for I:=1 to NumData do
430             TimeVec[I]:=TimeVec[I]*1440; {Convert to minutes}
431           TestType:=Simulation;
432           WellType:=Observation;
433           SaveData(TimeVec, DdVec, RateVec,
434           TestType, WellType, Distance, 1, NumData);
435         end;
436       end; {Series case}
437     end; {cases}
438     writeln; write('Go again? '); Short:='Y/N';
439     Again:=CapOptions(Short); TextColor(green);
440   until Again=2;
441   ChainTo('GWMENU.CHN',IOCode);
442   if IOCode<>0 then writeln('Unable to chain to program GWMENU.CHN!');
443 end. {# of Program DRAWDOWN.PAS}
```

6.2. Include file LEAKFUN2.FUN, listing

```
 1 Function LeakFunc {Functions for solution of leaky artesian well
    function.
 2   Not for use with LeakyFit}
 3 (u,RB: real): real;
 4 var
 5   Test: byte;
 6   TestUm, TestB, TestU, F1, RBsqrd: real;
 7
 8 Function Bessel1 {Bessel function of the first kind and zero order}
 9 (RB:real): real;
10 const
11   Factor : array [1..15] of real=(0.39894228,
12   0.01328592,2.25319E-3,-1.57565E-3,9.16281E-3,-0.02057706,
13   0.02635537,-0.01647633,3.92377E-3,3.5156229,3.0899424,
14   1.2067492,0.2659732,0.0360768,4.5813E-3);
15 var
16   RB3, RB3sqrd, RB3cubed, RB3fourth: real;
17 begin
18   RB3:=RB/3.75; RB3sqrd:=RB3*RB3;
19   RB3fourth:=RB3sqrd*RB3sqrd;
```

```
 20   if RB<=3.75
 21     then
 22       Bessel1:=1+
 23         Factor[10]*RB3sqrd+
 24         Factor[11]*RB3fourth+
 25         Factor[12]*RB3sqrd*RB3fourth+
 26         Factor[13]*RB3fourth*RB3fourth+
 27         Factor[14]*RB3fourth*RB3fourth*RB3sqrd+
 28         Factor[15]*RB3fourth*RB3fourth*RB3fourth
 29     else
 30       begin
 31         RB3cubed:=RB3sqrd*RB3;
 32         Bessel1:=1/sqrt(RB)*exp(RB)*(
 33           Factor[1]+
 34           Factor[2]*(1/RB3)+
 35           Factor[3]*(1/RB3sqrd)+
 36           Factor[4]*(1/RB3cubed)+
 37           Factor[5]*(1/RB3fourth)+
 38           Factor[6]*(1/(RB3fourth*RB3))+
 39           Factor[7]*(1/(RB3fourth*RB3sqrd))+
 40           Factor[8]*(1/(RB3fourth*RB3cubed))+
 41           Factor[9]*(1/(RB3fourth*RB3fourth)));
 42       end;
 43 end; {Bessel function of the first kind}
 44
 45 Function Bessel2 {Bessel function of the second kind and zero order}
 46 (RB: real): real;
 47 const
 48   Factor: array[1..14] of real=(-0.57721566,0.4227842,
 49   0.23069756,0.0348859,0.00262698,1.075E-4,7.4E-6,1.25331414,
 50   -0.07832358,0.02189568,-0.01062446,5.87872E-3,-2.5154E-3,
 51   5.3208E-4);
 52 var
 53   RB2, RB2sqrd, RB2fourth, TwoRB, TwoRBsqrd, TwoRBcubed: real;
 54   TwoRBfourth: real;
 55 begin
 56 if RB<=2
 57   then
 58     begin
 59       RB2:=RB/2; RB2sqrd:=RB2*RB2;
 60       RB2fourth:=RB2sqrd*RB2sqrd;
 61       Bessel2:=- ln(RB2)*Bessel1(RB)+
 62        Factor[1]+
 63        Factor[2]*RB2sqrd+
 64        Factor[3]*RB2fourth+
 65        Factor[4]*RB2sqrd*RB2fourth+
 66        Factor[5]*RB2fourth*RB2fourth+
 67        Factor[6]*RB2sqrd*RB2fourth*RB2fourth+
 68        Factor[7]*RB2fourth*RB2fourth*RB2fourth
 69     end
 70   else
 71     begin
 72       TwoRB:=2/RB; TwoRBsqrd:=TwoRB*TwoRB;
 73       TwoRBcubed:=TwoRB*TwoRBsqrd;
 74       TwoRBfourth:=TwoRBsqrd*TwoRBsqrd;
 75       Bessel2:=1/sqrt(RB)*exp(-RB)*(
 76        Factor[8]+
 77        Factor[9]*TwoRB+
 78        Factor[10]*TwoRBsqrd+
```

```
79        Factor[11]*TwoRBcubed+
80        Factor[12]*TwoRBfourth+
81        Factor[13]*TwoRBfourth*TwoRB+
82        Factor[14]*TwoRBfourth*TwoRBsqrd)
83      end;
84 end; {Bessel Function of the second kind}
85
86 Function IntFact {Solution for integral, small factorials}
87 (I: integer): integer;
88 var
89   J: integer;
90 begin
91   if I>7 then
92   begin
93     writeln;
94     writeln('Integer overflow in Factorial function');
95   end;
96   if I<0 then
97   begin
98     writeln('Negative value in Factorial function');
99   end;
100  J:=1;
101  while I>1 do
102  begin J:=J*I; I:=I-1; end;
103    if I=0 then J:=0;
104  IntFact:=J;
105 end; {Function IntFact}
106
107 Function SSURB {Solution of an indefinite integral}
108 (u,RB: real): real;
109 const
110   Term: array[1..15] of real=(0.093307812017,
111    0.492691740302,1.215595412071,2.269949526204,
112    3.667622721751,5.42533662741,7.565916226613,
113    10.120228568019,13.130282482176,16.654407708330,
114    20.776478899449,25.623894226729,31.407519169754,
115    38.530683306486,48.026085572686);
116   Factor: array[1..15] of real=(0.239578170311,
117    0.560100842793,0.887008262919,1.22366440215,
118    1.57444872163,1.94475197653,2.34150205664,
119    2.77404192683,3.25564334640,3.80631171423,
120    4.45847775384,5.27001778443,6.35956346973,
121    8.03178763212,11.5277721009);
122 var
123   I: integer;
124   A, B, F, Wu, Um: real;
125 begin  {main part of SSURB}
126   B:=RB*RB; Wu:=0; I:=1;
127   repeat
128     if u>=1
129     then begin
130       A:=1/(u+Term[I]);
131       F:=A*EXP(-(u+B*A*0.25+Term[I]));
132     end;
133     if u<1 then begin
134       Um:=0.25*B*(1/U);
135       A:=1/(Um+Term[I]);
136       if Um>35 then F:=0 else F:=A*EXP(-(Um+B*A*0.25+Term[I]));
137     end;
```

```
138     if F>0 then
139     begin Wu:=Wu+F*Factor[I]; I:=I+1; end;
140   until (I=16) or (F<=0);
141   SSURB:=Wu
142 end; {Function SSURB}
143
144 Function INTPOWER {Raise a real, A to an integral power, B.}
145 (A: real; B: integer): real;
146 var
147   I: integer; Temp: real;
148 begin
149   Temp:=1;
150   for I:=1 to B do
151     Temp:=Temp*A;
152   Intpower:=Temp;
153 end; {Function INTPOWER}
154
155 Function SUMURB(u,RB: real): real;
156 const
157   Limit=5; SmallNum=1E-8;
158 var
159   Temp, HI, HG: real;
160   M, N, H6, H9, BF, AF: integer;
161 begin
162   Temp:=0; HI:=RB*RB*0.25; N:=1; M:=1;
163   repeat
164     repeat
165       H6:=N-M+1; H9:=N+2; BF:=IntFact(H9); AF:=IntFact(H6);
166          HG:=Intpower(-1,N+M)*(AF/(BF*BF))*Intpower(HI,M)*Intpower(u,N-M);
167       Temp:=Temp+HG;
168       if HG<SmallNum then
169       begin
170         M:=N; N:=Limit;
171         end;
172       M:=M+1;
173     until M=N+1;
174     N:=N+1;
175   until N=Limit+1;
176   Sumurb:=Temp;
177 end; {Function SUMURB}
178
179 begin {# main part of Leakfunc}
180   if u>80 then Leakfunc:=0
181 else
182   RBsqrd:=RB*RB;
183   If u>=1 then Test:=1
184     else if RBsqrd>u then Test:=2
185     else Test:=3;
186   Case Test of
187     1: LeakFunc:=Ssurb(u,RB);
188     2: LeakFunc:=2*Bessel2(RB)-Ssurb(u,RB);
189     3: begin
190       F1:=RBsqrd*0.25/u;
191       LeakFunc:=2*Bessel2(RB)-Bessel1(RB)*Wellfunc(F1)+
192         exp(-F1)*(0.5772+ln(u)+WellFunc(u)-u+
193         u*((Bessel1(RB)-1)/(RBsqrd*0.25))-u*u*Sumurb(u,RB));
194     end {Case 3}
195   end; {All cases}
196 end; {Function Leakfunc}
```

6.3. REFERENCES

Bear, J., 1972. Dynamics of Fluids in Porous media. Amer. Elsevier Publ. Co., Inc., New York.

Bouwer, H., 1978. Groundwater Hydrology. McGraw-Hill Kogakusha Ltd. 480pp.

Clarke, D.K., 1987. Microcomputer Programs in Groundwater. Elsevier, Amsterdam, Oxford, New York, Toronto.

Cobb, P.M., McElwee, C.D., and Butt, M.A., 1982. An automated numerical evaluation of leaky aquifer pumping test data etc. Groundwater series 6, Kansas Geo. Surv.

Cooper, H.H., and Jacob, C.E., 1946. A generalised graphical method for evaluating formation constants and summarizing well field history. Trans. Am. Geoph. Union 27: pp. 526-534.

Freeze, R.A., and Cherry, J.A., 1979. Groundwater. Prentice-Hall Inc., New Jersey, USA. pp604.

Hantush, M.S. 1956. Analysis of data from pumping tests in leaky aquifers. Trans. Amer. Geophys. Union, 37, ppl 702-714.

Hantush, M.S. 1960. Modification of the theory of leaky aquifers. J. Geophys. Res., 65, pp. 3713-3725.

Hantush, M.S. 1964. Hydraulics of wells. Advances in Hydroscience, vol. 1, V.T. Chow (ed.), Academic Press, New York, 281-432.

Huntoon, P.W., 1980. Computationally efficient polynomial approximations used to program the Theis equation. Groundwater, Vol. 18, No. 2, March-April.

Hantush, M.S., and Jacob, C.E., 1955. Non-steady radial flow in an infinite leaky aquifer. Am. Geophys. Un. Trans. 36: pp95-100.

Kinzelbach, W., 1986. Groundwater Modelling, An Introduction with Sample Programs in Basic. Elsevier, Amsterdam-Oxford-New York-Tokyo. 333pp.

Marino, M.A., and Luthin, J.N., 1982. Seepage and Groundwater. Developments in water Science. Elsevier, Amsterdam-Oxford-New York-Tokyo. 487pp.

Muskat, M., 1937. The Flow of Homogeneous Fluids through Porous Media. McGraw Hill Book Co., New York.

Neuman, S.P., 1975. Analysis of pumping test data from anisotropic unconfined aquifers considering delayed gravity response. Water Resources Res., 11, pp. 329-342.

Theis, C.V., 1935. The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using groundwater storage. Trans. Am. Geophs. Un 16:pp519-524.

Walton, W.C., 1984. Handbook of Analytical Groundwater Models. International Groundwater Modeling Centre, Butler Uni., Indiana, USA.

Chapter 3

The previous chapter dealt with a program which simulated drawdown in a piezometer at some given distance from a discharging well in various types of aquifers. The simpler program, SIM7, explained in this chapter, simulates the drawdown in the discharging well itself. It does so by evaluation of the well equation, Equation 1.4. For convenience this equation is presented again below:

$$s=AQ + BQ \text{ Log } t + CQ^n \qquad (3.1)$$

where s is the drawdown in the discharging well,

t is the time from the commencement of discharge,

Q is the discharge rate,

A and B are coefficients dependent upon characteristics of the aquifer and well,

C is the coefficient quantifying the 'well loss',

and n has a value of between 1.5 and 3. (See Jacob, 1947; Rorabaugh, 1953; Lennox, 1966; and/or Bouwer, 1987.)

1. THE USE OF UNITS IN SIM7

In all the programs described in this book, the basic units are the metre and the day. However, as it is common practice to measure discharge test times in minutes, and consequently to calculate the first coefficient of the well equation, A, for minute units; the user may enter this coefficient based on either the minute, or the day. If the user elects to use minutes as his or her time unit, then not only must A be based on the minute, but the finishing times for the discharge stages must also be entered in minutes. The units for the discharge rate will be cubic metres per day no matter what the unit chosen above may be.

The time unit used in calculation of the drawdowns is the day. If the user stated that he wished to use minutes, then his value for A will be used to calculate the first term, but then this result will be converted to suit the day unit at about line 138 of the program. This will not be apparent to the user because, in this case, term values displayed as the calculations progress are converted to suit the minute unit.

While times associated with the drawdowns are calculated in days, they are recorded in minutes in preparation for storage to disk file, if that is required.

2. A DEMONSTRATION RUN

If the steps below are followed then a simulation the same as that used to produce the data for Figure 1.1 will be produced.

1/ Run the program by calling it up from the primary menu.

2/ Answer the question 'What time units' by pressing m for minutes.

3/ Give A the value 0.003, give B 0.005, and C 0.0001.

4/ The exponent for this exercise is 2.

5/ There are four pumping steps, as below:

Step No.	Discharge rate	Finishing time
1	200	50 minutes
2	280	100
3	360	150
4	450	200

6/ Enter the effective radius of the well as 0.1 (m). (If you save the simulated data to disk then this will be recorded, but it takes no part in the calculations.)

The computer will now calculate the discharge rates for the 'pumps' that will be used to produce the simulation. (The multiple pump method for simulation of a multi stage discharge test was explained in Chapter 2.) The computer will display the following:

"The 'pumps' involved are discharging at the rates below:

No. 1, 200.00

No. 2, 80.00

No. 3, 80.00

No. 4, 90.00"

(The model is made up of the sums of drawdowns from four pumps. The first pump starts at 0 minutes and continues until the end of the test, it pumps at 200 kilolitres per day; the second pump starts at 50 minutes and pumps at 80 kilolitres per day making a total of 280kl/day, etc., etc.)

As soon as you respond to the request to "Press any key to continue", the calculation of the simulation will commence. The first several lines of the display are given below:

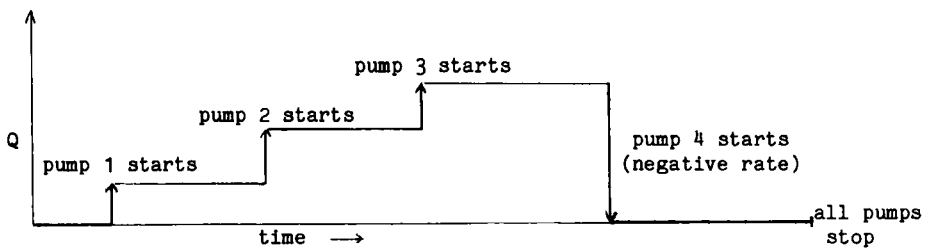
"No.	min.	days	A	B	C	Drawdown	Rate	Pumps
1	1	0.0007	0.6000	-0.0000	0.4000	1.000	200.00	1
2	2	0.0016	0.6000	0.3541	0.4000	1.354	200.00	1"

The "No." of the display is the record, or reading number. The values in the "min." column are the times in minutes, from the beginning of discharge, of the simulated drawdown reading. The "days" column displays the

same time, but converted to days. The "A", "B", and "C" columns give the values of the first, second, and third terms of the Well Equation (Equation 3.1). The total of the three terms is the calculated drawdown. The 'Rate' column holds the current discharge rate. The 'Pumps' column records the number of pumps currently in use for the model; this will increase by one for each change in discharge rate, whether that change is an increase or a decrease. The data of this simulation is shown in graphic form in Figure 1.1 on page 42.

The multiple pump model can also handle decreases in discharge rate by use of adding a 'pump' having a negative discharge. For example, figure 3.1 shows how a discharge test consisting of three steps of increasing rate, followed by a period of recovery would be simulated.

Figure 3.1



A schematic representation of a multistage discharge test with discharge rate plotted against time.

3. PROCEDURES AND FUNCTIONS

Following the convention adopted in the previous chapters, the following part of this chapter will explain the main sections of the program itself.

CalcDrawdown procedure Line 121

Purpose: to calculate the time, drawdown, and discharge rate for each simulated reading.

Notice that the entered values for the coefficients of the well equation are stored in the variables A, B, and C; while the values of the terms calculated for each of these coefficients are stored in the variables AA, BB, and CC.

Line 126 sets the values of both TimeStep and AccumTime to equal the constant, StartTime (which is set in line 22). TimeStep holds the value to be added to the current time to produce the time of the next 'reading', it is increased exponentially by multiplication with the constant ProportIncrem in

line 156 so long as the current time is not equal to, or greater than, the end of the current discharge step. The constant ProportIncrem is set to approximately the cube root of two in line 21. (There is no real need to use a number such as the cube root of two, any number would do.) AccumTime is used to hold the total accumulated time from the beginning of the first discharge stage, while TimeInStep receives the time calculated as having elapsed from the time that the particular 'pump' under consideration commenced.

The outer loop from line 128 to line 162 counts the number of 'pumps' that are used to simulate the sum of the effects of all the 'pumps'; the first stage requires one 'pump', the second stage two, etc.

Lines 132 to 133 set the value for TimeInStep. If the first pump is being considered (J=1), then TimeInStep equals the total accumulated time, but in other cases TimeInStep is the difference between the accumulated time and the end of the previous step. Line 134 is not acted upon unless an error occurs.

The value for the second term of the well equation is calculated in line 135 by summing the values for each individual 'pump'.

Line 137 calculates the value for the first term of the well equation correctly for the case when A has been entered in days. If the value of A was entered for the time unit minutes, then the value of the second term is adjusted in line 138. The constant MinDayLog (set in line 23) is the logarithm to the base 10 of the number of minutes in a day.

Lines 139 and 140 calculate the value for the third term. As there is no single operator in Pascal which can be used to raise a number to a power, (unlike Basic or Fortran) line 140 uses the logarithm (ln) and its inverse (exp) to fulfil this purpose.

In line 141 the value in AccumTime which has the unit days is converted to minutes and passed to the vector TimeVec ready for possible later storage to disk file. Later in the line the three terms are summed to produce the calculated drawdown at the current time, which is then stored in vector DdVec. The following line places a value in RateVec, completing the recording of the simulation for the current reading or record.

Lines 156 and 157 increment the current time in readiness for calculation of the next simulated drawdown.

For the simulation to be complete it is necessary to have a reading exactly at the end of each step. As the time of each successive reading is calculated from the preceding one, this is not likely to happen unless special steps are taken. Most of the remainder of this procedure ensures a

reading at the end of each step. The first lines in this section to be acted upon (apart from 156 and 157, mentioned above) will almost certainly be 158 and 159. Line 159 forces the value of AccumTime to the time at the end of the current step. After AccumTime has been set, the main loop will again be executed to produce a drawdown for this time. On the next pass through the later section of the procedure, the test in line 148 will be true, so the following few lines will be acted upon. Line 150 sets AccumTime to one minute (StartTime) past the end of the last discharge stage, and the following line resets TimeStep back to one minute. After incrementing the number of 'pumps' in line 151 we are ready to start calculating the drawdowns for the new discharge stage. The Boolean variable EndStep merely ensures that the section of code from line 155 to 160 is not executed at this time.

Called by: the main part of SIM7.

Calls: function Log, and procedure VideoOutput.

EnterData procedure Line 36

Purpose: to obtain data for the simulation from the operator.

Line 47 is a Turbo Pascal type conversion which is not available in standard Pascal. Function CapOptions returns with an integer value which is then converted to the type TimeUnits.

The loop from line 56 to 61 rejects any attempt to enter a number of steps greater than 20, as StepVec is dimensioned to 20 in line 10. (The user may alter this limit to suit his own purposes.) Similarly, for each step, the loop from line 73 to 84 ensures that time marking the end of the current step is greater than that marking the end of the previous step.

Called by: the main part of SIM7.

Calls: Functions ReadReal and CapOptions of file First.Seg.

Log function Line 28

Purpose: to calculate the logarithm, to the base ten, of a given number.

Called by: procedure CalcDrawdown

VideoOutput procedure Line 97

Purpose: to display the time/drawdown/discharge rate data as they are calculated.

As the equation terms are calculated for day units no matter which unit the user chose for data entry, if the user chose minute units then the values of the first and second terms are converted to what they would have been had

the calculations been done in minutes. These calculations are done in lines 108 and 109.)

Called by: procedure CalcDrawdown.

4. KEY LINES

```
  7 {#}{$I FIRST.SEG}
 26 {#}{$I SAVE.PRC}
 28 Function Log {Calculate the log to the base ten}
 36 Procedure EnterData; {Get the simulation description from the user}
 97 Procedure VideoOutput {Display data as it is calculated}
121 Procedure   CalcDrawdown; {Calculate all drawdown values}
168 begin {# main part of program SIM7}
199 end. {# of program SIM7.PAS}
```

5. PROGRAM SIM7.PAS, LISTING

```
  1 Program SIM7_PAS;
  2 {To simulate multistage drawdown in a discharging well given the
    coefficients
  3 of the well equation.}
  4
  5 {$R+}
  6
  7 {#}{$I FIRST.SEG}
  8
  9 type       {Special variables for program SIM7.PAS}
 10   StepVec=array[1..20] of real;
 11   TimeUnits=(Minutes, Days);
 12 var
 13   TimeError: boolean;
 14   NumData, NumOfSteps, NumOfPumps: integer;
 15   Distance, AccumTime, A, B, C, Exponent, TimeInStep, TimeStep: real;
 16   PumpRateVec, StepRateVec, StepTime: StepVec;
 17   TestType: Test; WellType: Well;
 18   TimeVec, DdVec, RateVec: MainVec;
 19   TimeUnit: TimeUnits;
 20 const
 21   ProportIncrem=1.2599211;
 22   StartTime=6.94444e-4;
 23   MinDayLog=3.1583625;
 24   Heading='No.   Time (days)   Drawdown    Pumps   Discharge ';
 25
 26 {#}{$I SAVE.PRC}
 27
 28 Function Log {Calculate the log to the base ten}
 29 (Num: real): real;
 30 const
 31   LnTen=2.302585093;
 32 begin
 33   Log:=Ln(Num)/LnTen;
 34 end; {Function Log}
 35
 36 Procedure EnterData; {Get the simulation description from the user}
 37 var Ch: char;
 38 begin
 39   writeln;
 40   writeln('Please note that there is a maximum of twenty pumping
    steps.');
```

```
 41  writeln;
 42  writeln('   The unit for discharge rates will be expected to be
    m^3/day,',
 43    ' but for');
 44  writeln('the coefficient A, and the length of steps, you may use',
 45    ' either days or');
 46  writeln('minutes.');
 47  write('What time units will you use, ');
 48  TimeUnit:=TimeUnits(CapOptions('Minutes or Days?')-1);
 49  writeln;
 50  writeln('Please enter the equation coefficients as prompted below.');
 51  write('What value for A? '); A:=ReadReal(2);
 52  write('B = ? ');
 53  B:=ReadReal(2);
 54  write('C = ? '); C:=ReadReal(2);
 55  write('What value for the exponent (often assumed to be 2)? ');
 56  exponent:=ReadReal(2);
 57  repeat
 58    write('How many pumping steps, including any steps of zero discharge?
    ');
 59    NumOfSteps:=ReadInt(1);
 60    if (NumOfSteps>20) or (NumOfSteps<1) then
 61      writeln('Invalid!  No. of steps must be from 1 to 20.');
 62  until (NumOfSteps>0) and (NumOfSteps<21);
 63  for I:=1 to NumOfSteps do
 64  begin
 65    writeln; writeln('Step No.',I:3);
 66    repeat
 67      write('What discharge rate? (m^3/day) ');
    StepRateVec[I]:=ReadReal(1);
 68      if StepRateVec[I]<0 then
 69        writeln('Sorry, negative discharge is not allowed.');
 70    until StepRateVec[I]>=0;
 71    if I=1
 72      then PumpRateVec[1]:=StepRateVec[1]
 73      else PumpRateVec[I]:=StepRateVec[I]-StepRateVec[I-1];
 74      repeat
 75        TimeError:=false;
 76        write('Finishing time ');
 77        if TimeUnit=Minutes then write('(minutes)? ') else write('(days)?
    ');
 78        if TimeUnit=Days then StepTime[I]:=ReadReal(1)
 79          else StepTime[I]:=ReadReal(1)/1440;
 80        if I>1 then if StepTime[I]<=StepTime[I-1] then
 81        begin
 82          writeln('Error: time <= to the previous time!');
 83          TimeError:=true;
 84        end;
 85      until not TimeError;
 86  end; {for I}
 87  writeln;
 88  write('What is the effective radius of the well? (m) ');
 89  Distance:=ReadReal(2);
 90  writeln('The ''Pumps'' involved are discharging at the rates below:');
 91  for I:=1 to NumOfSteps do
 92    writeln('No. ',I:3,'  ',PumpRateVec[I]:9:2);
 93  writeln('Press any key to continue.');
 94  Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
 95 end; {Procedure EnterData}
```

```
 96
 97 Procedure VideoOutput {Display data as it is calculated}
 98 (I, J: integer; AA, BB, CC: real);
 99 var
100   Ch: char;
101 const
102   Head1='No.       min.     days      A         B         C
    Drawdown';
103   Head2='   Rate   Pumps';
104 begin
105   if I=1 then begin write(Head1); writeln(Head2) end;
106   if TimeUnit=Days
107     then write(I:3,' ',TimeVec[I]:8:0,' ',TimeVec[I]/1440:9:4,
108       ' ',AA:9:4,' ',BB:9:4,' ')
109             else  write(I:3,'  ',TimeVec[I]:8:0,'  ',TimeVec[I]/1440:9:4,
110       ' ',AA-B*MinDayLog*StepRateVec[NumOfPumps]:9:4,' ',
111       BB+B*MinDayLog*StepRateVec[NumOfPumps]:9:4,' ');
112   writeln(CC:9:4,' ',DdVec[I]:9:3,' ',StepRateVec[J]:9:2,'
    ',NumOfPumps:3);
113   if (I div 20)*20=I then
114   begin
115     writeln('Press any key to continue.');
116     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
117     write(Head1); writeln(Head2);
118   end;
119 end; {Procedure VideoOutput}
120
121 Procedure CalcDrawdown; {Calculate all drawdown values}
122 var
123   EndStep, Finished: boolean;
124   I: integer;
125   AA, BB, CC: real;
126 begin
127   Finished:=false;
128   I:=1; TimeStep:=StartTime; AccumTime:=StartTime;
129   NumOfPumps:=1; {Start at one minute}
130   repeat
131     BB:=0; EndStep:=false;
132     for J:=1 to NumOfPumps do
133     begin
134       if J=1 then TimeInStep:=AccumTime
135         else TimeInStep:=AccumTime-StepTime[J-1];       {Current time}
136       if TimeInStep<=0 then writeln('TimeInStep =',TimeInStep:8);
137       BB:=BB+B*Log(TimeInStep)*PumpRateVec[J];
138     end; {for J}
139     AA:=A*StepRateVec[NumOfPumps];
140     if TimeUnit=Minutes then AA:=AA+B*MinDayLog*StepRateVec[NumOfPumps];
141     if StepRateVec[NumOfPumps]=0 then CC:=0 else
142       CC:=C*exp(exponent*ln(StepRateVec[NumOfPumps]));
143     TimeVec[I]:=AccumTime*1440; DdVec[I]:=AA+BB+CC;
144     RateVec[I]:=StepRateVec[NumOfPumps];
145     VideoOutput(I,NumOfPumps,AA,BB,CC);
146     I:=I+1;
147     if AccumTime=StepTime[NumOfSteps] then Finished:=true;
148     if not Finished then
149     begin
150       if AccumTime=StepTime[NumOfPumps] then
151       begin
152         AccumTime:=StepTime[NumOfPumps]+StartTime;
```

```
153         TimeStep:=StartTime; NumOfPumps:=NumOfPumps+1;
154         EndStep:=true;
155       end; {if end of step}
156       if not EndStep then
157       begin
158         TimeStep:=TimeStep*ProportIncrem;
159         AccumTime:=AccumTime+TimeStep;
160         if AccumTime>StepTime[NumOfPumps]
161           then AccumTime:=StepTime[NumOfPumps]; {Get last reading for
    step}
162       end; {if not EndStep}
163     end; {if not Finished}
164   until Finished;
165   NumData:=I-1;
166 end; {Procedure CalcDrawdown}
167
168 begin {# main part of program SIM7}
169   ClrScr; TextColor(green); FileName:='';
170   writeln('   Simulation of multi-stage discharge test, with water
    levels');
171   writeln('being measured in the discharging well.  The Well Equation,');
172   writeln('rather than the Well Function, is used.');
173   EnterData;
174   CalcDrawdown;
175   write('Do you want the data saved in a disc file? ');
176   if Response('YN')='Y' then
177   begin
178     writeln('Do you want initial conditions included in the file?');
179     write('ie. drawdown and discharge rate at time zero ');
180     if Response('YN')='Y' then
181     begin
182       for I:=NumData downto 1 do
183       begin
184         TimeVec[I+1]:=TimeVec[I]; DdVec[I+1]:=DdVec[I];
185         RateVec[I+1]:=RateVec[I];
186       end; {for}
187       TimeVec[1]:=0; DdVec[1]:=0; RateVec[1]:=0; NumData:=NumData+1;
188     end;
189     writeln('Please note that times in the file are in minutes.');
190     TestType:=Simulation; WellType:=Pumped;
191     SaveData(TimeVec, DdVec, RateVec, TestType, WellType, Distance,
192       1, NumData);
193   end;
194   {The code below should only be used when this program is used as a
195    chain file.}
196   ChainTo('GWMENU.CHN',IOCode);
197   if IOCode<>0 then
198   writeln('Unable to chain to program GWMENU.CHN!');
199 end. {# of program SIM7.PAS}
```

6. REFERENCES

Bouwer, H., 1978. Groundwater Hydrology. McGraw-Hill Kogakusha Ltd. 480pp.

Jacob, C.E., 1947. Drawdown test to detirmine effective radius of artesian well. Trans. Am. Soc. Civ. Eng. 112: pp 1047-1070.

Lennox, D.H., 1966. Analysis and application of step-drawdown test. J. Hydraul. Div., Proc. Am. Soc. Civ. Eng. 92(HY6): pp 25-48.

Rorabaugh, M.I., 1953. Graphical and theoretical anal. of step-drawdown tests of artesian wells. Proc. Am. Soc. Civ. Eng. 79, separate No. 362, 23 pp.

Chapter 4

This chapter describes a program which simulates drawdown in an infinite, homogeneous, unconfined aquifer having a hydraulic conductivity in the vertical direction which is not necessarily the same as that in the horizontal direction. The program described is a computer application of Neuman's Unconfined Well Function, hence the name, NEUMAN. Program NEUMAN produces a series of time/drawdown/discharge rate data using the method and tables published in Marino and Luthin pp262-265 (1982). The method was originally published by Neuman (1972, 1973, 1975).

1. NEUMAN'S UNCONFINED WELL FUNCTION

According to Neuman's unconfined well function, an unconfined aquifer behaves at very early times like a confined aquifer, with water being released to a discharging well by compression of the aquifer and by expansion of the water itself. At later times vertical drainage becomes significant and consequentially the drawdown curve flattens out. Finally, if discharge persists long enough, the drawdown curve follows a Theis curve corresponding to instantaneous release of water from the unconfined aquifer. (Presumably the water is no more instantaneously released at later times than it is as earlier times, but as the duration of discharge becomes longer, so the delay in release of water becomes progressively less significant by comparison.) It seems that Neuman's method takes no account of the decline in transmissivity due to the thinning of the saturated part of the unconfined aquifer following partial dewatering; this effect has been included in the present program by use of the inverse Cooper-Jacob correction (Cooper and Jacob, 1946).

The simplified form of the equation representing Neuman's solution is:

$$s = \frac{Q}{4\ Pi\ T}\ W(u_A,\ u_B,\ Psi) \qquad (4.1)$$

where $W(u_A,\ u_B,\ Psi)$ is known as Neuman's unconfined well function,

Q is the discharge rate,

T is transmissivity,

$$u_A = \frac{R^2\ S}{4\ T\ t}\ , \qquad (4.2)$$

$$u_B = \frac{R^2 S_y}{4 T t}, \qquad (4.3)$$

$$Psi = \frac{K_v R^2}{K_h b^2}, \qquad (4.4)$$

where K_v is the vertical hydraulic conductivity of the aquifer,

K_h is horizontal hydraulic conductivity,

R is the distance from the discharging well to the piezometer,

S is the confined storage coefficient,

S_y is the unconfined specific yield,

t is the time from the commencement of discharge,

and b is the initial thickness of the saturated part of the aquifer.

It should be pointed out that at any one time drawdown is not a function of both u_A and u_B, but is a function of the former at early times, and of the latter at later times. For very late times Neuman's function approaches the Theis well function and, in fact, this program makes a transition from the u_B function to Theis for calculation of drawdown at very late times.

In practice Neuman's well function is evaluated by consulting tabulated data, program NEUMAN interpolates the tables which are themselves written into the program. The tables will not be repeated here, please consult the program listing, or Marino and Luthin (1982) if you wish to study them. (It would not be difficult to make a modified form of the program which could produce a printed copy of the tables.)

2. USING THE PROGRAM

Program NEUMAN may be used to produce a simulation of a period of constant rate discharge from an unconfined aquifer showing delayed yield. The simulation would synthesize drawdowns as could be expected from a piezometer at some distance from the discharging well. Such a simulation would be possible by manual calculation methods, but so time consuming as to be prohibitive. Also, if Neuman's well function is manually evaluated for intermediate times, then the decision on whether to use the u_A or u_B form might not be an easy one to make. It is hoped that with this program Neuman's unconfined well function will become much more accessible to the practising hydrogeologist, allowing production of type curves simply and in a short time.

Broadly, the program is very similar in use to program DRAWDOWN which was the subject of Chapter 2; a series of questions are asked by the

computer, and answered by the user to quantify the aquifer parameters. This is followed by calculation of a series of time/drawdown/discharge rate data which may eventually be saved to a disk file for later display, analysis, modification, listing, or plotting on paper or screen. While in the most general terms the operation is similar to that of DRAWDOWN, in fact a totally different method is used to achieve the desired ends.

I will attempt to explain the operation of the program as the user would see it in the following section.

2.1. An example run

The user will become familiar with the program most quickly by running it and examining the results. Also, an explanation can best be given in conjunction with a trial run. The program is called from the primary menu of the GW group of programs in the same way as are the other major programs explained previously in this book.

Once the program is running, enter data at the prompts as given below. Enter only the numerals, do not enter the unit names.

```
confined storage coefficient; enter 0.0001
unconfined specific yield, 0.1
discharge rate, (m^3/day), 500
horizontal hydraulic conductivity, (m/day), 10
vertical hydraulic conductivity, (m/day), 0.1
saturated thickness of the aquifer, (m), 20
distance between discharge well and piezometer, (m), 30
time to terminate the simulation, (days), 400
```

Entry of the zero before the decimal point is not necessary, but is printed above for clarity.

After the final entry, that of the 400 days, calculations of drawdowns will commence. The first item to be displayed is the value for Psi for this simulation. Psi is not dependent upon time, it's value remains the same for the whole simulation. The data displayed should be the same as that below. (The exponential format - under the MinDif column - will be slightly different if a maths co-processor is not used, and there may be very minor differences in some of the other figures.)

Time in days is shown in the first column. The second column shows the drawdown calculated by Neuman's method for early times, and the third column is the drawdown by Neuman's method for late times. The forth column is the

drawdown calculated by the Theis method assuming instantaneous release of water from the unconfined aquifer, and using the specific yield rather than the storage coefficient. This column is also corrected for the effect on transmissivity due to the partial dewatering of the aquifer by use of the inverse Cooper-Jacob correction. (The Cooper-Jacob correction is explained in Chapter 2 on page 141, and is given in Equation 2.5. The inverse is given in Equation 2.6.) The final column of the display gives the minimum difference between the second and third columns up to that time; the purpose of this will be explained later.

Time	Dd A	Dd B	Dd Theis-Jacob	MinDif
0.0007	0.241	0.546	0.000	3.0E-001
0.0010	0.287	0.546	0.000	2.6E-001

At early times it is the drawdown in the Dd A column that is used as the calculated drawdown. Note that the drawdown in this column is already greater than 0.2 metres at one minute; there has been quick drawdown due to confined conditions. The difference between the figures in column Dd A and Dd B is declining. Continuing with the display:

0.0014	0.333	0.546	0.000	2.1E-001
0.0020	0.378	0.546	0.000	1.7E-001
0.0028	0.419	0.546	0.000	1.3E-001
0.0040	0.453	0.546	0.000	9.3E-002
0.0056	0.482	0.546	0.000	6.4E-002

At 0.005 days the rate of increase in the drawdown of column two is noticeably decreasing as the effects of vertical flow begin to be felt.

0.0079	0.504	0.546	0.000	4.1E-002
0.0112	0.522	0.546	0.000	2.4E-002
0.0158	0.534	0.546	0.000	1.2E-002
0.0224	0.542	0.546	0.000	4.0E-003
0.0317	0.547	0.547	0.001	1.3E-004

A very important point has now been reached; the drawdowns calculated by both the early and late time parts of Neuman's function are close to identical (the difference, as shown in the MinDif column, is 0.00013m). This time is therefore the time for transition to the u_B function.

Notice that while the drawdown in column two has not quite become static, the static phase in column three has ended. This means that there will be no absolutely static phase in the final data. If the storage coefficient was any closer to the specific yield than it is for this

simulation (three orders of magnitude), then the static phase would be even less in evidence.

Also notice that for the first time the drawdown calculated by the Theis method is large enough to appear on the display.

Time	Dd A	Dd B	Dd Theis-Jacob	MinDif	
0.0448	0.548	0.548	0.005	1.3E-004	
0.0634	0.547	0.548	0.013	"	
0.0896	0.546	0.549	0.029	"	
0.1267	0.547	0.550	0.053	"	
0.1792	0.547	0.553	0.086	1.3E-004	

The static stage has been reached in column two, but this will have no effect on the final output as that is now given by column three.

0.2534	0.547	0.557	0.127	1.3E-004	
0.3584	0.546	0.562	0.174	"	
0.5069	0.547	0.567	0.228	"	
0.7168	0.547	0.575	0.286	"	
1.0137	0.547	0.588	0.347	"	
1.4336	0.547	0.604	0.411	"	
2.0274	0.000	0.626	0.477	"	uA not tabled

The value of uA has become too small to allow reference to Neuman's early time table (uA is inversely proportional to time), so no further uA values will be calculated. This is of no concern as this part of the algorithm has played it's part.

2.8672	0.000	0.653	0.545	1.3E-004	uA not tabled
4.0548	0.000	0.687	0.613	"	"
5.7344	0.000	0.728	0.683	"	"
8.1097	0.000	0.776	0.753	"	"
11.4688	0.000	0.829	0.824	"	"
16.2193	0.000	0.887	0.896	"	"

Now the drawdown in the Theis-Jacob column has exceeded that in the Dd B column, because the former makes some allowance for the slight reduction in transmissivity due to the aquifer being partially dewatered. (Remember that the aquifer thickness was 20 metres, and this now has been effectively reduced to about 19.1 metres, reducing the transmissivity by around 4% in the vicinity of the piezometer.)

22.9376	0.000	0.949	0.967	1.3E-004	uA not tabled	
32.4387	0.000	1.014	1.040	"	"	
45.8752	0.000	0.000	1.112	"	"	uB not tabled

Times are now sufficiently large to cause the value of uB to move off the table, so no further drawdowns can be displayed in this column. Again this is no problem, because the drawdowns calculated by the Theis-Jacob method can now be used.

Time	Dd A	Dd B	Dd Theis-Jacob	MinDif		
64.8774	0.000	0.000	1.185	1.3E-004	uA not tabled	uB not tabled
91.7505	0.000	0.000	1.259	"	"	"
129.7548	0.000	0.000	1.332	"	"	"
183.5010	0.000	0.000	1.406	"	"	"
259.5096	0.000	0.000	1.418	"	"	"
376.0020	0.000	0.000	1.555	"	"	"
400.0000	0.000	0.000	1.574	1.3E-004	uA not tabled	uB not tabled

The given maximum time, 400 days, has now been reached so this part of the calculation is finished.

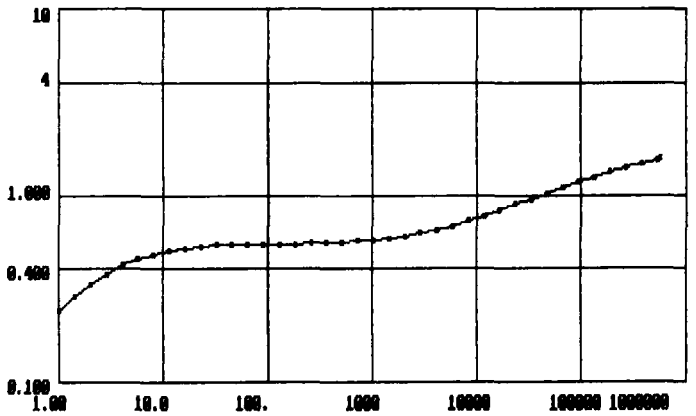
The program now selects the appropriate parts of the above data, joins each part by use of Lagrangian interpolation (to obtain a smooth transition), and displays the final form of the data. In this case the final display is as given in Table 4.1 below.

Table 4.1

Record #	Time	Drawdown	Record #	Time	Drawdown
1	0.0007	0.241	2	0.0010	0.287
3	0.0014	0.333	4	0.0020	0.378
5	0.0028	0.419	6	0.0040	0.453
7	0.0056	0.482	8	0.0079	0.504
9	0.0112	0.522	10	0.0158	0.534
11	0.0224	0.541	12	0.0317	0.545
13	0.0448	0.547	14	0.0634	0.548
15	0.0896	0.549	16	0.1267	0.550
17	0.1792	0.553	18	0.2534	0.557
19	0.3584	0.561	20	0.5069	0.567
21	0.7168	0.575	22	1.0137	0.588
23	1.4336	0.604	24	2.0274	0.626
25	2.8672	0.653	26	4.0548	0.687
27	5.7344	0.728	28	8.1097	0.776
29	11.4688	0.829	30	16.2193	0.887
31	22.9376	0.949	32	32.4387	1.014
33	45.8752	1.088	34	64.8774	1.166
35	91.7505	1.246	36	129.7548	1.327
37	183.5010	1.406	38	259.5096	1.481
39	367.0020	1.555	40	400.0000	1.574

The data of Table 4.1 is plotted on a log-log graph in Figure 4.1.

Figure 4.1



A log-log plot of the example run of program NEUMAN; see the text.

2.2. Joining the three segments of the simulation

The minimum difference between columns two and three occurred at a time of 0.0317 days. The program uses the drawdowns on column two at 0.0112 and 0.0158 days, and those of column three at 0.0634 and 0.0896 days, and uses them to interpolate three new readings for 0.0224, 0.0317, and 0.0448 days. The three interpolated 'readings' are 0.541, 0.545, and 0.547 metres; which are not, in this case, very different from either the drawdowns of column two or three.

The joining of the data from column three, for late times, to that of column four, for very late times, is similar. Here it is the last two readings of column three (readings numbers 31 and 32) which, with readings numbers 37 and 38 are interpolated to obtain slightly modified values for drawdowns number 33 to 36.

2.3. Results from NEUMAN compared to those from program DRAWDOWN

Readers may have noticed that the conceptual model for a situation in which program NEUMAN can be used has strong similarities to one previously described in Chapter 2, (section 3.4, page 133). Figures 4.2a and 4.2b show the two conceptual models.

The model described in Chapter two uses the leaky artesian well function at early to later times, and follows this with an unconfined (Theis with the Cooper-Jacob correction) for very late times. In this case dewatering in the overlying unconfined aquifer is at first neglected. When drawdown calculated

for the piezometer, given a value of S equal to the specific yield, begins to exceed the drawdown calculated for the leaky confined aquifer, the aquifer is treated as unconfined.

In the case corresponding to Figure 4.2b, the aquifer is treated as unconfined from start to finish.

Figure 4.2a

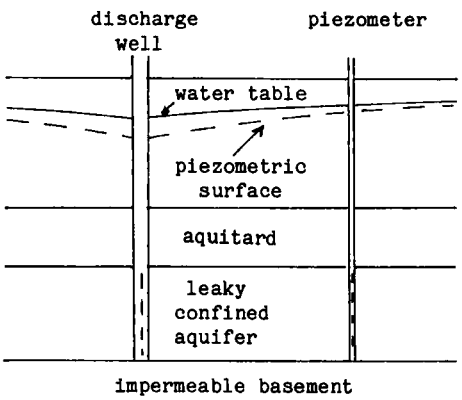


Figure 4.2b

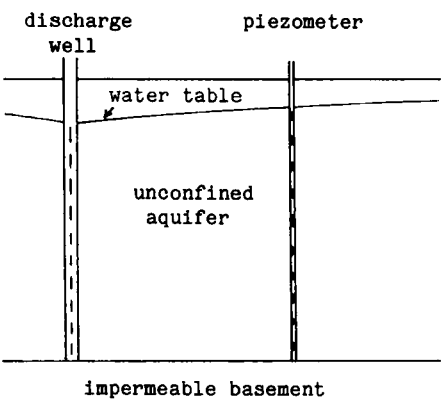
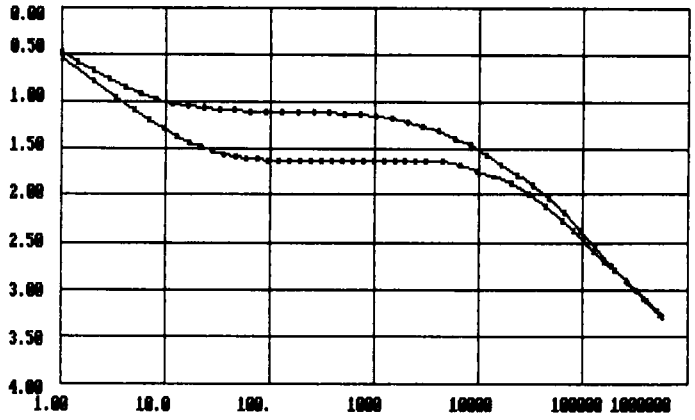


Figure 4.2a shows the conceptual model for the leaky confined aquifer model, modified (extended in time) to allow for dewatering of the overlying unconfined aquifer at late times. Figure 4.2b is the conceptual model for an unconfined aquifer as dealt with in program NEUMAN.

Figure 4.3 shows curves produced by the two simulations. The lower curve is the leaky artesian aquifer model of program DRAWDOWN extended by data from the simple unconfined aquifer model of the same program. After being saved to disk files, the two models (or segments of models) are joined using program JOINWTD. The upper curve was produced by program NEUMAN. Data entries for the three components were as shown in Table 4.2.

The two conceptual models are different, but not greatly different given that the aquitard of the first model is replaced by the low vertical hydraulic conductivity of the second model. The two curves are also similar, but one tends to wonder about the significance of the difference. Is the difference in the curves entirely due to the differences in the conceptual models, or is it perhaps partly or wholly due to inaccuracies in the fit between the theory and the reality? If the earth had been formed with a set of aquifers of all the various types, all perfectly homogeneous and isotropic, we could test our theories using the scientific method. But no aquifer anywhere exactly matches any model!

Figure 4.3



A comparison of output from program NEUMAN with a similar model created by use of programs DRAWDOWN and JOINWTD (see the text).

Table 4.2

	Leaky confined	simple unconfined	Neuman
S	0.0001	n. a.	0.0001
Sy	n. a.	0.1	0.1
Q	1000	1000	1000
K_h	n. a.	n. a.	10
K_v	0.1	n. a.	0.1
b	n. a.	20	20
b'	20	n. a.	n. a.
T	200 (10 • 20)	200	n. a.
R	30	30	30
t	400	400	400

Note that b above is the thickness of the unconfined aquifer, and in the former model is considered to be the effective thickness of the combined aquifers when, at late times, they behave as a single unconfined aquifer. The variable b' is the thickness of the semiconfining layer. Values for K_h and b are not required for the leaky simulation, because transmissivity is entered instead. Similarly, K_h is not entered for the simple unconfined simulation, but could be calculated as T/b. Resistance to vertical flow in the two models is similar because the product of K_v and b' of the first model is equal to the product of K_v and b of Neuman's model.

3. THE LIMITS OF THE TABLED DATA

The greatest restriction in basing a computer program on tabled data is that the tables rarely go far enough to cover every likely situation. It may be possible to get around this problem to a very limited extent by extrapolation, but the risks of producing unacceptable errors are quite high. For that reason, extrapolation should be used only for very short distances from the tabled data, and then only with care.

This program extrapolates for Psi only, and if this is necessary, the user is warned. Extrapolation in this dimension will probably be found necessary only when there is a very great difference between hydraulic conductivity in the vertical and horizontal directions, or when either the distance from the discharging well, or the thickness of the aquifer, are very large or very small. (Note the definition of Psi, Equation 4.4.)

It is quite possible to go beyond the limits of the tables in dimension of u_A and u_B too. These parameters are time dependent. Generally when $1/u_A$ is very small there would be little or no error in taking drawdown to be zero. Very small values of $1/u_B$ will not usually cause trouble because transition from the first function will be into some part of the second function above the lowest values. Similarly, high values in $1/u_A$ will probably not occur before transition to the second function.

High values in $1/u_B$ are a special case, as they will occur at large times. At Psi values of greater than 0.1 it is possible to move straight from Neuman's second function to the Theis function with negligible error, but when Psi is less than this then interpolation will often be required between the limit of the second function and the Theis curve. This will be done as required so long as there are sufficient data calculated for late times by use of the Theis function to make it possible.

3.1. Cautionary notes

The limits to the tabled data (ie. the boundaries of the tables themselves) will obviously lead to limitations in the applicability of this program. For example, given the entered data below, the simulation will produce unreasonable results because the boundary of the u_B table will be reached significantly before the drawdown calculated by the Theis equation is comparable. (S=0.0001, Sy=0.1, Q=1000, K_h=10, K_v=0.1, b=100, R=20, and t=100.)

It seems that Neuman's tables do not take into account the reduction in transmissivity caused by the dewatering of the unconfined aquifer. For this reason drawdowns calculated from the u_B table that are large relative to

aquifer thickness will be less than those calculated for the same times by the Theis equation with the Cooper-Jacob correction. In practice this should not be a significant problem as the limitations enforced by the boundaries of the tables make calculations for a piezometer close to the discharging well impossible, so great drawdowns at the piezometer cannot happen before dewatering of the aquifer at the discharging well.

Please remember that this program does not calculate, or make any allowance for, the drawdown at the discharging well. The program will quite happily calculate a piezometer drawdown when the discharge rate is so high that the drawdown at the discharging well would be greater than the aquifer thickness.

Having written about the problems associated with the limitations of tabled data tends to make one appreciate how much easier formulae are to deal with in computer applications. But is there a greater danger here? At least the author of a method that relies for it's solution on a table, that he supplies, can be reasonably confident that users will not use his method outside of it's range of validity. The author of a method that relies on the use of formulae has no such control; it is quite possible to push his formulae way beyond any limits that he might have thought reasonable!

4. DESCRIPTION BY PROCEDURE AND FUNCTION

The following section of this chapter will describe the operation of program NEUMAN by consideration of it's component parts, one at a time. The main, or controlling part of the program is described first; this is followed by procedures and functions in alphabetical order.

Main part of program Neuman Line 411

The more interesting part of this section starts about line 417. At this point the user enters his/her estimates of the aquifer parameters, duration of discharge etc., and the preliminary time/drawdown vectors are calculated. In line 418 a test is made to discover whether the duration of the simulated discharge was great enough to allow the joining of Neuman's early function data to that of his late function. An error flag is set at this point if insufficient data are present for interpolation of the join. If the error condition did not arise, then the program goes on to produce the final vector, drawdown vector number 4 (DdVec4), by the call to CalcDdVecNo4 in line 421.

Calls: procedures GetNatLogs, GetInput, CalcDdVectors, CalcDdVecNo4, SimplePlot, SaveData, and ChainTo; and function Response.

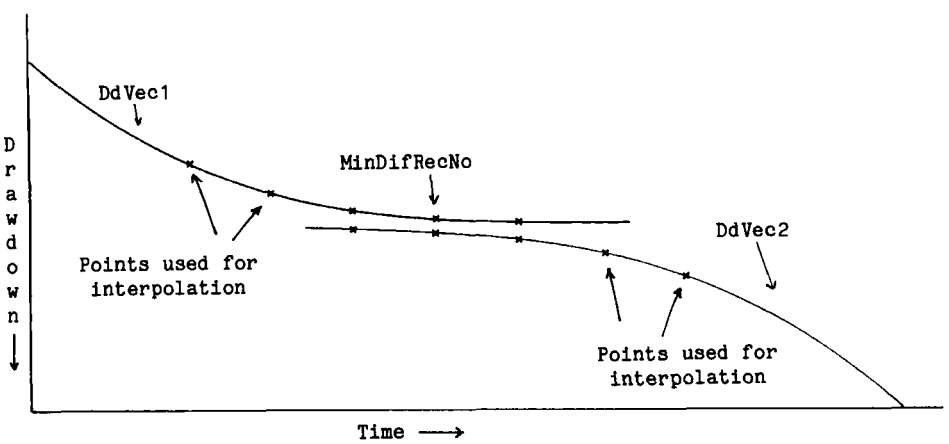
CalcDdVecNo4 procedure Line 342

Purpose: to select drawdown values from the three preliminary vectors, and supervise the connection of these segments into one whole, drawdown vector number four.

The first data to be placed into DdVec4 is the earlier part of DdVec1 (line 346). Variable MinDifRecNo is the record number at the point where the difference between DdVec1 and DdVec2 was at a minimum. It has been set previously, in procedure CalcDdVectors, and it marks the point about which the data from DdVec1 will be joined to that from DdVec2.

Figure 4.4 shows graphically how the data points are selected for joining the chosen parts of the first two vectors. You will see from the figure that the record pointed at by variable MinDifRecNo is the central point of the section to be interpolated. The values chosen to use in the interpolation are the second and third records both before and after this central record.

Figure 4.4



The selection of the points used to interpolate the data joining Neuman's early times function to his later times function.

Program lines 346 to 350 convert the chosen times to their natural logarithms, place them in the small vector IndepVar, and place the chosen drawdowns in the vector DepVar. It is these independent variables and associated dependent variables that will be interpolated. Notice that two of the drawdowns are taken from DdVec1, and two from DdVec2. Now lines 351 and 352 can call up the Lagrangian interpolation function and obtain a four point interpolation for drawdown at a given time. The drawdowns that are

interpolated are for the times to either side of the time pointed to by MinDifRecNo, as well as for that time. ie. There are three drawdowns interpolated, one for the time of the record preceding the time when the drawdown calculated for DdVec1 was closest to that for DdVec2, one for the time when they were at their closest, and one for the time of the record following that closest point.

The next step is to include the records following the interpolated section from DdVec2 into DdVec4. Records are taken from DdVec2 until drawdown values of zero are reached, indicating that the boundary of Neuman's table was encountered at this time (lines 354 to 357). Lines 359 to 361 now add drawdowns from DdVec3 (Theis-Jacob) up to the end of the given duration of discharge; at this stage the join between the chosen data of DdVec2 and DdVec3 has not been interpolated.

In the tests that were done during program development it was found that with large values of Psi interpolation of this last join was unnecessary, hence the test in line 362. Users may, of course, alter the 0.1 value on this line if they wish. It may prove that interpolation of this join is worth while in all cases.

If Psi is less than 0.1 then the two segments of data are joined by an interpolated section on the call to function InterpLastJoin. As the point from which the interpolation commences is fixed at the end of the calculated data of DdVec2 (record number LastuB), successful interpolation of this join will depend partly upon there being sufficient data after this record in vector DdVec3. If interpolation is not possible due to too early a termination of the simulation line 366 causes the later, erroneous data, to be truncated.

The remaining section of this procedure displays the data in it's final form, that is, the data of vector DdVec4.

Called by: the main part of program NEUMAN.

Calls: function Lagrange and procedure InterpLastJoin.

CalcDdVectors procedure Line 267

Purpose: to control the calculation of the three preliminary drawdown vectors.

The three vectors contain drawdowns calculated from: Neuman's unconfined well function for early times, DdVec1; Neuman's unconfined well function for late times, DdVec2; and the Theis well function modified by the inverse Cooper-Jacob correction, DdVec3.

Transmissivity has not been entered as such, so is calculated as the product of horizontal hydraulic conductivity (K_h, represented in the program as Kr) and aquifer thickness (b, represented in the program as AqThick). Also in line 269, the first time for the simulation is set to approximately one minute, or 0.0007 days. This value may be modified to suit the user, of course. The variable FirstTime will be treated as a constant, while the variable Time will hold the current time as that is incremented through the simulation. Psi is evaluated by use of Equation 4.4 in line 270; and a preliminary, extremely large, value is given to MinDif in readiness for finding the minimum value for this figure. Line 272 causes a warning to be displayed if Psi is beyond the limits covered by Neuman's tables. A small extrapolation may be acceptable, but data produced by a large extrapolation should be treated with extreme scepticism. Figures produced by the Lagrangian interpolation routine in extrapolation generally tend to be rather wild at moderate distances from the known data.

Production of the drawdowns to fill the three vectors begins in the loop that starts at line 277. Line 279 calculates the value of uA by use of Equation 4.2, and then line 280 tests whether this value is within Neuman's table. If the value is tabled, then a drawdown is produced from Neuman's function for early times in line 282. Equation 4.1 is used here, as well as a call to function InterpWuA which produces a value from Neuman's table by four point Lagrangian interpolation. Lines 288 to 296 produce a drawdown figure for the second vector (DdVec2) by very similar means, but using u_B in place of u_A, and Neuman's second table rather than his first. (Please refer to the notes on the functions InterpWuA and InterpWuB for details on the interpolation of the tables.)

The third drawdown figure, the Theis-Jacob drawdown remains. This is evaluated by a call to function UnconfinedDd after the calculation of the value of u (lines 297 and 298). (The equations used are given in the notes on function UnconfinedDd.) Please note that for this particular function specific yield, S_y, takes the place of storage coefficient, S, in Equation 4.6, because we are dealing with an unconfined aquifer.

Lines 300 to 303 search the data produced in vectors DdVec1 and DdVec2 for the minimum difference in calculated drawdown. Each time a smaller difference is calculated than that already recorded, it's record number is noted in variable MinDifRecNo, and the difference itself is stored in variable MinDif.

The section from line 304 to 308 writes the calculated data to the screen as the calculations proceed. Lines 309 to 314 delay the program

periodically to give the user a chance to study the data being produced.

In line 315, the record counter, I, is incremented as each set of drawdowns are produced, and the current time is exponentially increased.

Called by: the main part of program NEUMAN.

Calls: functions InterpWuA, InterpWuB, WellFunc, and UnconfinedDd.

GetInput procedure Line 251

Purpose: to obtain values for the parameters required for running the simulation from the user.

Called by: the main part of program NEUMAN.

Calls: function ReadReal (described in the notes on file First.seg, in the Preliminary section of this book).

GetNatLogs procedure Line 242

Purpose: to fill reference vectors with the natural logs of the tabled arguments of the functions W(u_A,Psi) and W(u_B,Psi).

These vectors will be referred to by functions InterpWuA and InterpWuB when it is time to interpolate Neuman's tabled functions.

Called by: the main part of program NEUMAN.

InterpLastJoin procedure Line 321

Purpose: to produce a smooth transition to the join between the data from Neuman's late time function [W(u_B,Psi), DdVec2], and that from the Theis-Jacob vector (DdVec3).

This interpolation is very similar to that explained under procedure CalcDdVecNo4, so a full explanation will not be given here. While the point about which the interpolation was made in procedure CalcDdVecNo4 was the closest approach between DdVec1 and DdVec2, here it is the record number of the last value calculated for DdVec2. This point is discovered by lines 326 and 327, and stored in variable LastuB. (It may be preferable to make this join around the point of closest approach between vectors two and three. This could be achieved by a search for the minimum difference such as that used for the join between vectors one and two. In this present application it was decided to remain with Neuman's function as far as possible.)

In some simulations, as noted earlier in this chapter, the drawdowns calculated by function W(u_B,Psi) will not continue sufficiently long to closely approach those calculated by Theis-Jacob, due to the limits of the tabled data. In this case a poor join might result, and the simulated data would then be particularly questionable. (All simulated data is questionable

to some extent; a sceptical nature should be cultivated.) Lack of a close approach results in a warning being displayed by lines 329 and 330.

Called by: procedure CalcDdVecNo4

Calls: function Lagrange

InterpWuA function Line 186

Purpose: to obtain a value for Neuman's early time drawdown function, $W(u_A,Psi)$, by use of Lagrangian interpolation of tabulated data.

As this function is not particularly simple an illustration of the way in which it works may be useful. Suppose the program has called on this function to interpolate the table for the value of W(0.2, 0.025). The first operation would then be to select the section of Neuman's table that brackets these values. This section of the table is shown below.

		ln(Psi)			
		-5.5215	-4.6052	-2.8134	-2.3026
	0.87547	0.648	0.633	0.57	0.54
$\ln(1/u_A)$	1.3862	0.992	0.963	0.849	0.792
	2.0794	1.52	1.46	1.23	1.12
	2.6391	1.97	1.88	1.51	1.34

You will notice that Psi has been replaced by it's natural logarithm, and u_A has been replaced by the natural logarithm of the inverse of u_A. The reason for using the inverse of u_A is that that is what is used on Neuman's tables. The reason for the use of natural logarithms is only a little more involved. The tabled values of Psi (for this section of table) were 0.004, 0.01, 0.06, and 0.1, and those of $1/u_A$ were 2.4, 4, 8, and 14. These series are much closer to being exponential than they are to arithmetic series. Interpolation produces more accurate results when the indices to a table are a linear series, so natural logarithms have been used to produce a rough linearization of the indices.

Now to interpolate for $1/u_A$=5 and Psi=0.025 we first interpolate each row for the given Psi. Interpolating the top row gives 0.60786, the second row gives 0.91830, the third row gives 1.36766, and the last row gives 1.72966. The next and final step is to interpolate these four values for $1/u_A$=5. This gives a result of W(0.2,0.025) being approximately equal to 1.06093. (If the reader is interested in exploring this further, there are several programs on Lagrangian interpolation in Clarke, 1987. The working of Lagrangian interpolation is given in the notes on function Lagrange.)

Having explained what is done, I will now attempt to explain how the program does it. Program line 195 increments the value of integer variable I

until it points to a value in PsiVec that is greater than the Psi for which an interpolation is required. This is the way in which the part of the table to be interpolated is chosen. The table is two dimensional, so line 197 sets the variable J to point at the correct value of InvuAVec (inverse of u_A vector) along the other index of the table.

The double loop from line 199 to 208 produces the first four interpolations mentioned above, in effect producing a new section of the table for the given value of Psi. This temporary table is placed in the vector TempuAVec, and a little later, in line 206 is interpolated for the given value of the inverse of u_A.

Called by: procedure CalcDdVectors

Calls: function Lagrange

InterpWuB function Line 216

Purpose: to obtain a value for Neuman's late time drawdown function, $W(u_B,Psi)$, by use of Lagrangian interpolation of tabulated data.

This function is identical in operation to InterpWuA above, the difference only being in the table that is interpolated (in the previous case the tabled data is held in matrix WuAMat, and in this case it is held in matrix WuBMat).

Called by: procedure CalcDdVectors

Calls: function Lagrange

Lagrange function Line 148

Lagrangian interpolation (Hildebrand, 1968) was chosen for use in this program because it may be used when the intervals between the tabled independent variables are unequal. Some other methods of interpolation demand equal intervals between independent variables; a constraint that disallows their use in this type of application.

Lagrangian interpolation uses the equation:

$$F(x) = \frac{(x-x_2)(x-x_3) \ . \ . \ . \ (x-x_n)}{(x_1-x_2)(x_1-x_3) \ . \ . \ . \ (x_1-x_n)} F(x_1) +$$

$$\frac{(x-x_1)(x-x_3) \ . \ . \ . \ (x-x_n)}{(x_2-x_1)(x_2-x_3) \ . \ . \ . \ (x_2-x_n)} F(x_2) + \ . \ . \ .$$

$$+\frac{(x-x_1)(x-x_2)\ \ldots\ (x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\ \ldots\ (x_n-x_{n-1})}\ F(x_n) \qquad (4.5)$$

where

1/ F(x) is the value of the function at x, ie. the unknown value.

2/ x is the value for which the solution is required, ie. the independent variable for which the corresponding dependent variable is required. Call it the input of the function.

3/ x_1, x_2, etc. are the independent variables.

4/ $F(x_1)$, $F(x_2)$, etc. are the values of the function at x_1, x_2, etc., the dependent variables. These particular dependent variables are know from the table.

5/ n is the number of known function values that are to be considered in the interpolation.

This method of interpolation allows any number of independent/dependent variable pairs to be used.

While the equation is large, it is not particularly complex as a number of general rules may be deduced from it.

1/ The number of terms is equal to the number of known function values to be used in the interpolation and the number of factors in both the denominator and numerator of each term is one less than this number.

2/ The minuend (that which is diminished in a subtraction operation) of the numerator for all factors and in all terms is the input.

3/ The subtrahend (that which is deducted from the minuend) in the Ith factor in each term is always equal to x_i when the number of the term is greater than i, in all other cases it is equal to x_{i+1}. eg. The first subtrahend in the second term is x_1 because 1 is less than 2, but the second subtrahend in the same term is x_3 because 2 is not less than 2. This rule applies to both numerator and denominator.

4/ All minuends in all denominators in the Ith term are equal to x_i.

The program calculates the value of each term in the outer loop from line 157 to 182, with variable I holding the value of the current term.

The byte variable, Flag, is used to handle the tricky rule number 3 above; it always has the value zero if Factor is less than I, but is given the value 1 when Factor is increased sufficiently to equal I. Once given a value of 1 Flag will retain it until it is time to evaluate the next term.

(See lines 159 and 164.) This incrementing of the value of Flag ensures that the value of each element of the vector Subtrahend is given the correct value in line 166. Calculation of the minuend for the denominator (variable Minuend) is handled by lines 164 and 168. (It may well be that this could be simplified.)

If there are only two points to be interpolated then there is only one factor in each numerator and denominator, but if there are more than two points then the section from lines 171 to 179 is used to evaluate the additional factors. (In this program four point interpolation is used, but this function may be used for interpolation from any number of points. Two point interpolation is, of course, linear interpolation and can be achieved by a subroutine much simpler than this.)

Finally the sum of all terms is accumulated in line 181.

Called by: functions InterpWuA, InterpWuB, and procedures InterpLastJoin and CalcDdVecNo4.

SimplePlot procedure Line 396

Purpose: to produce a very simple plot of the final time-drawdown data so that the user may visualise the simulation.

The plot is a semilogarithmic graph, hence the calculation of TimePlotFac in line 398 uses the natural log of the first time and the maximum time; the 639 in this line relates to the width of the high resolution screen, 640 pixels. TimePlotFac and TimePlotConst are calculated to allow calculation of the x coordinate of the graph by a linear function (line 404). The calculation of the y coordinate is also by linear function (line 405), but no constant is required because zero drawdown is plotted at y=0. In fact zero drawdowns will probably be rare in this simulation.

This plot is not meant to be at all detailed, as a detailed plot may be obtained from program PLOTWTD.

Called by: the main part of program NEUMAN.

UnconfinedDd function Line 137

Purpose: the calculate the drawdown assuming an unconfined aquifer with negligible delayed yield, but allowing for the effects of reduction in transmissivity due to reduction in saturated thickness.

Drawdown is first calculated by use of the Theis equations which apply to a confined aquifer, but here specific yield takes the place of storage coefficient. (This is valid so long as drainage is instantaneous, and drawdown is negligible relative to aquifer thickness. In fact neither of

these conditions are met, but as the equation is used only at very late times the delay for drainage is assumed to be negligible in relation to the duration of discharge, and the inverse Cooper-Jacob correction will be applied to allow for aquifer dewatering.)

The Theis equation is given below.

$$s = \frac{Q\ W(u)}{4\ Pi\ T} \qquad (4.6)$$

where

$$u = \frac{R^2\ S_y}{4\ T\ t}, \qquad (4.7)$$

$$W(u) = -0.577216 - \ln u + u - \frac{u^2}{2 * 2!} + \frac{u^3}{3 * 3!} \ldots, \qquad (4.8)$$

and the other variables are defined on the first two pages of this chapter.

The calculation of u is carried out on line 297 in procedure CalcDd-Vectors, s is evaluated in line 298 of the same procedure, and the calculation of W(u) is performed by Function WellFunc (lines 118 to 135); but all equations are placed together here for convenience.

The above equations are taken from Bouwer, (1978, p73) and are also in Marino and Luthin, (1982, p250) and in part in Freeze and Cherry, (1979, p317). The original reference is Theis, 1935.

The equation for the inverse Cooper-Jacob correction (Cooper and Jacob, 1946) is given in this work on page 141 as Equation 2.6. This equation is applied in line 145 to convert the calculated confined drawdown to what might be expected given the reduction in transmissivity due to partial dewatering.

The characteristics of the Cooper-Jacob correction are such that given a calculated confined drawdown of half the aquifer thickness, then the unconfined drawdown will be equal to the aquifer thickness. This being so, the error catching routine of lines 140 to 144 is necessary to avoid program termination due to an attempted extraction of the square root of a negative number.

Called by: procedure CalcDdVectors

WellFunc function Line 118

Purpose: to evaluate the Theis well function.

This function is identical to that by the same name in program DRAWDOWN. The equation solved is Equation 4.8 above. Please refer to page 142 for further information.

Called by: procedure CalcDdVectors

5. KEY LINES OF PROGRAM NEUMAN

```
  6 {#}{$I FIRST.SEG}
 36 {#}WuAMat: array[1..19,1..13] of real=(
 76 {#}WuBMat: array[1..25,1..13] of real=(
116 {#}{$I SAVE.PRC}
118 Function WellFunc {The well function of u}
137 Function UnconfinedDd {Convert confined to unconfined, inverse Jacobs}
148 Function Lagrange {Lagrangian interpolation}
186 Function InterpWuA {Interpolate matrix InvuA}
216 Function InterpWuB {Interpolate matrix InvuB}
242 Procedure GetNatLogs; {Fill the natural log vectors}
251 Procedure GetInput; {Allow user to enter data on the aquifer}
267 Procedure CalcDdVectors; {Calculate the drawdowns, by all methods}
321 Procedure InterpLastJoin; {Interpolate section joining DdVec2 and 3}
342 Procedure CalcDdVecNo4; {Fill this vector with the final drawdowns
    values}
396 Procedure SimplePlot; {Produce a simple graph of the time-drawdown data}
411 {#}begin {Main part of program Neuman}
```

6. LISTING OF PROGRAM NEUMAN

```
  1 Program NEUMAN_PAS;
  2 {Produces a simulation of drawdown using Neumans Unconfined Well
    Function}
  3
  4 {$R+}
  5
  6 {#}{$I FIRST.SEG}
  7
  8 type
  9   ShortVec=array[1..9] of real;
 10 var
 11   Again, uAout, uBout: boolean;
 12   NumOfPoints: byte;
 13   Ch: char;
 14   LastuB, NumData, MinDifRecNo, Factor: integer;
 15   MaxDd, MaxTime, u, Wu, R, T, S, Sy, Kr, Kv, Time, Q, ThickAq: real;
 16   TimePlotConst, TimePlotFac, DdPlotFac: real;
 17   FirstTime, MinDif, Psi, InputX, uA, uB, WuA, WuB, Drawdown: real;
 18   IndepVar, DepVar: ShortVec;
```

```
188   Simulation (3)

 19   LnPsiVec: array[1..13] of real;
 20   LnInvuAVec: array[1..19] of real;
 21   LnInvuBVec: array[1..25] of real;
 22   TimeVec, DdVec1, DdVec2, DdVec3, DdVec4, RateVec: MainVec;
 23   TestType: Test; WellType: Well;
 24 const
 25   TimeInc=1.4142136;
 26   NumOfColumns=13; NumOfARows=19; NumOfBRows=25;
 27   MinuA=7.14e-5; MaxuA=2.5; MinuB=0.0025; MaxuB=2500;
 28   MinPsi=0.001; MaxPsi=7;
 29   PsiVec: array[1..13] of real=(0.001,0.004,0.01,0.06,0.1,0.6,1,2,3
     ,4,5,6,7);
 30   InvuAVec: array[1..19] of real=(0.4,0.8,1.4,2.4,4,8,14,24,40,80
     ,140,240,
 31     400,800,1400,2400,4000,8000,14000);
 32   InvuBVec: array[1..25] of real=(0.0004,0.0008,0.0014,0.0024,0.004
     ,0.008,
 33     0.014,0.024,0.04,0.08,0.14,0.24,0.4,0.8,1.4,2.4,4,8,14,24,40,80,140,
 34     240,400);
 35
 36   {#}WuAMat: array[1..19,1..13] of real=(
 37   (0.0248,0.0243,0.0241,0.023,0.0224,0.0188,0.017,0.0138,0.0113,0.00933,
 38     0.00772,0.00639,0.0053),
 39   (0.145,0.142,0.14,0.131,0.127,0.0988,0.0849,0.0603,0.0435,0.0317,
     0.0234,
 40     0.0174,0.0131),
 41   (0.358,0.352,0.345,0.318,0.304,0.217,0.175,0.107,0.0678,0.0445,0.0302,
 42     0.021,0.0151),
 43   (0.662,0.648,0.633,0.57,0.54,0.343,0.256,0.133,0.0767,0.0476,0.0313,
     0.0214,
 44     0.0152),
 45   (1.02,0.992,0.963,0.849,0.792,0.438,0.3,0.14,0.0779,0.0478,0.0313,
     0.0215,
 46     0.0152),
 47   (1.57,1.52,1.46,1.23,1.12,0.497,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 48     0.0152),
 49   (2.05,1.97,1.88,1.51,1.34,0.507,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 50     0.0152),
 51   (2.52,2.41,2.27,1.73,1.47,0.507,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 52     0.0152),
 53   (2.97,2.8,2.61,1.85,1.53,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 54     0.0152),
 55   (3.56,3.3,3,1.92,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 56     0.0152),
 57   (4.01,3.65,3.23,1.93,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 58     0.0152),
 59   (4.42,3.93,3.37,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 60     0.0152),
 61   (4.77,4.12,3.43,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 62     0.0152),
 63   (5.16,4.26,3.45,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,
     0.0215,
 64     0.0152),
```

```
 65   (5.4,4.29,3.46,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 66     0.0152),
 67   (5.54,4.3,3.46,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 68     0.0152),
 69   (5.59,4.3,3.46,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 70     0.0152),
 71   (5.62,4.3,3.46,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 72     0.0152),
 73   (5.62,4.3,3.46,1.94,1.55,0.507,0.317,0.141,0.0779,0.0478,0.0313,0.0215,
 74     0.0152));
 75
 76   {#}WuBMat: array[1..25,1..13] of real=(
 77   (5.62,4.3,3.46,1.94,1.56,0.508,0.318,0.142,0.078,0.0479,0.0314,0.0215,
 78     0.0153),
 79   (5.62,4.3,3.46,1.94,1.56,0.508,0.318,0.142,0.0781,0.048,0.0315,0.0215,
 80     0.0153),
 81   (5.62,4.3,3.46,1.94,1.56,0.508,0.318,0.142,0.0783,0.0481,0.0316,0.0217,
 82     0.0154),
 83   (5.62,4.3,3.46,1.94,1.56,0.508,0.318,0.142,0.0785,0.0484,0.0318,0.0219,
 84     0.0156),
 85   (5.62,4.3,3.46,1.94,1.56,0.508,0.318,0.142,0.0789,0.0487,0.0321,0.0221,
 86     0.0158),
 87   (5.62,4.3,3.46,1.94,1.56,0.509,0.319,0.143,0.0799,0.0496,0.0329,0.0228,
 88     0.0164),
 89   (5.62,4.3,3.46,1.94,1.56,0.51,0.321,0.145,0.0814,0.0509,0.0341,0.0239,
 90     0.0173),
 91   (5.62,4.3,3.46,1.94,1.56,0.512,0.323,0.147,0.0838,0.0532,0.0361,0.0257,
 92     0.0189),
 93   (5.62,4.3,3.46,1.94,1.56,0.516,0.327,0.152,0.0879,0.0568,0.0393,0.0286,
 94     0.0215),
 95   (5.62,4.3,3.46,1.94,1.56,0.524,0.337,0.162,0.098,0.0661,0.0478,0.0362,
 96     0.0284),
 97   (5.62,4.3,3.46,1.94,1.56,0.537,0.35,0.178,0.113,0.0806,0.0612,0.0486,
 98     0.0398),
 99   (5.62,4.3,3.46,1.95,1.57,0.557,0.374,0.205,0.14,0.106,0.0853,0.0714,
100     0.0614),
101   (5.62,4.3,3.46,1.96,1.58,0.589,0.412,0.248,0.184,0.149,0.128,0.113,
    0.102),
102   (5.62,4.3,3.46,1.98,1.61,0.667,0.506,0.357,0.298,0.266,0.245,0.231,
    0.22),
103   (5.63,4.31,3.47,2.01,1.66,0.78,0.642,0.517,0.47,0.445,0.43,0.419,
    0.411),
104   (5.63,4.31,3.49,2.06,1.73,0.954,0.85,0.763,0.733,0.718,0.709,0.703,
    0.699),
105   (5.63,4.32,3.51,2.13,1.83,1.2,1.13,1.08,1.07,1.06,1.06,1.05,1.05),
106   (5.64,4.35,3.56,2.31,2.07,1.68,1.65,1.63,1.63,1.63,1.63,1.63,1.63),
107   (5.65,4.38,3.63,2.55,2.37,2.15,2.14,2.14,2.14,2.14,2.14,2.14,2.14),
108   (5.67,4.44,3.74,2.86,2.75,2.65,2.65,2.64,2.64,2.64,2.64,2.64,2.64),
109   (5.7,4.52,3.9,3.24,3.18,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14),
110   (5.76,4.71,4.22,3.85,3.83,3.82,3.82,3.82,3.82,3.82,3.82,3.82,3.82),
111   (5.85,4.94,4.58,4.38,4.38,4.37,4.37,4.37,4.37,4.37,4.37,4.37,4.37),
112   (5.99,5.23,5,4.91,4.91,4.91,4.91,4.91,4.91,4.91,4.91,4.91,4.91),
113   (6.16,5.59,5.46,5.42,5.42,5.42,5.42,5.42,5.42,5.42,5.42,5.42,5.42));
114   {End of constants}
115
116 {#}{$I SAVE.PRC}
117
118 Function WellFunc {The well function of u}
119 (u:real): Real;
```

```
190   Simulation (3)

120 const
121   C0=-0.57721566; C1=0.99999193; C2=-0.24991055; C3=0.05519968;
122   C4=-0.00976004; C5=0.00107857; C6=0.250621;    C7=2.334733;
123   C8=1.681534;    C9=3.330657;
124 var
125   u2,u3:real;
126 begin
127   if u<=0 then u:=1e-35;
128   if u>80 then WellFunc:=0
129   else begin
130     U2:=U*U; U3:=U2*U;
131   if u<1
132     then WellFunc:=-ln(u)+C0+C1*U+C2*U2+C3*U3+C4*U2*U2+C5*U2*U3
133     else WellFunc:=1/(U*EXP(U))*(C6+C7*U+U2)/(C8+C9*U+U2)
134   end {else WellFunc=}
135 end; {Function WellFunc}
136
137 Function UnconfinedDd {Convert confined to unconfined, inverse Jacobs}
138 (ConfinedDd: real):real;
139 begin
140   if ConfinedDd>=0.5*ThickAq
141   then begin
142     writeln('ERROR: Unconfined aquifer dewatered.');
143     UnconfinedDd:=ThickAq;
144   end
145   else UnconfinedDd:=ThickAq*(1-sqrt(1-(2*ConfinedDd)/ThickAq));
146 end; {Function UnconfinedDd}
147
148 Function Lagrange {Lagrangian interpolation}
149 (IndepVar, DepVar: ShortVec; NumOfPoints: byte; InputX: real):real;
150 var
151   Flag: byte;
152   I, Factor: integer;
153   Subtrahend: ShortVec;
154   Numerator, Denominator, Minuend, SumOfTerms: real;
155 begin
156   SumOfTerms:=0;
157   for I:=1 to NumOfPoints do     {For each term}
158   begin
159     Flag:=0;
160     for Factor:=1 to NumOfPoints-1 do
161     begin
162       if Factor=I
163       then begin
164         Minuend:=IndepVar[I]; Flag:=1;
165       end;
166       Subtrahend[Factor]:=IndepVar[Flag+Factor];
167     end;
168     if I=NumOfPoints then Minuend:=IndepVar[NumOfPoints];
169     Numerator:=InputX-Subtrahend[1];
170     Denominator:=Minuend-Subtrahend[1];
171     if NumOfPoints>2
172     then begin
173       for Factor:=2 to NumOfPoints-1 do
174       begin
175         Numerator:=Numerator*(InputX-Subtrahend[Factor]);
176         {Product of numerator factors up to and including factor No.
    "Factor"}
177         Denominator:=Denominator*(Minuend-Subtrahend[Factor]);
```

```
178       end;
179     end; {If NumOfPoints > 2}
180     {writeln('InputX=',InputX:8:3,'  Num=',Numerator:8:3,'  Den=',
    Denominator:8:3);}
181     SumOfTerms:=SumOfTerms+Numerator/Denominator*DepVar[I]
182   end; {for I}
183   Lagrange:=SumOfTerms;
184 end; {Function Lagrange}
185
186 Function InterpWuA {Interpolate matrix InvuA}
187 (InvuA, Psi: real): real;
188 var
189   I, J, K, L: integer;
190   TempReal: real;
191   TempuAVec: ShortVec;
192 begin
193   I:=3; J:=3;
194   {writeln('Interpolating for ',Psi:7,'   ',InvuA:7);}
195   while (PsiVec[I]<Psi) and (I+1<NumOfColumns) do I:=I+1;
196   {I is pointing at the third of four Psi values to be interpolated}
197   while (InvuAVec[J]<InvuA) and (J+1<NumOfARows) do J:=J+1;
198   {J is pointing at the third of four InvuA values to be interpolated}
199   for L:=1 to 4 do
200   begin
201     for K:=1 to 4 do
202     begin
203       IndepVar[K]:=LnPsiVec[K+I-3]; DepVar[K]:=WuAMat[L+J-3,K+I-3];
204       {write(IndepVar[K]:7:4,'  ',DepVar[K]:7:4,'; ');}
205     end;
206     TempuAVec[L]:=Lagrange(IndepVar, DepVar, 4, Ln(Psi));
207     {writeln; writeln(TempuAVec[L]:7:4);}
208   end;
209   for K:=1 to 4 do IndepVar[K]:=LnInvuAVec[J+K-3];
210   {writeln; for K:=1 to 4 do write(InvuAVec[J+K-3]:10:4); writeln;}
211   TempReal:=Lagrange(IndepVar, TempuAVec, 4, Ln(InvuA));
212   InterpWuA:=TempReal;
213   {writeln('Interpolation for WuA=',TempReal:7);}
214 end; {Function InterpWuA}
215
216 Function InterpWuB {Interpolate matrix InvuB}
217 (InvuB, Psi: real): real;
218 var
219   I, J, K, L: integer;
220   TempuBVec: ShortVec;
221 begin
222   I:=3; J:=3;
223   while (PsiVec[I]<Psi) and (I+1<NumOfColumns) do I:=I+1;
224   {I is pointing at the third of four Psi values to be interpolated}
225   while (InvuBVec[J]<InvuB) and (J+1<NumOfBRows) do J:=J+1;
226   {J is pointing at the third of four InvuB values to be interpolated}
227   for L:=1 to 4 do
228   begin
229     for K:=1 to 4 do
230     begin
231       IndepVar[K]:=LnPsiVec[K+I-3]; DepVar[K]:=WuBMat[L+J-3,K+I-3];
232       {write(IndepVar[K]:7:4,'  ',DepVar[K]:7:4,'; ');}
233     end;
234     TempuBVec[L]:=Lagrange(IndepVar, DepVar, 4, Ln(Psi));
235     {writeln; writeln(TempuBVec[L]:7:4);}
```

```
236   end;
237   for K:=1 to 4 do IndepVar[K]:=LnInvuBVec[J+K-3];
238   {writeln; for K:=1 to 4 do write(InvuBVec[J+K-3]:10:4); writeln;}
239   InterpWuB:=Lagrange(IndepVar, TempuBVec, 4, Ln(InvuB));
240 end; {Function InterpWuB}
241
242 Procedure GetNatLogs; {Fill the natural log vectors}
243 var
244   I, J: integer;
245 begin
246   for I:=1 to NumOfColumns do LnPsiVec[I]:=Ln(PsiVec[I]);
247   for I:=1 to NumOfARows do LnInvuAVec[I]:=Ln(InvuAVec[I]);
248   for I:=1 to NumOfBRows do LnInvuBVec[I]:=Ln(InvuBVec[I]);
249 end; {Procedure GetNatLogs}
250
251 Procedure GetInput; {Allow user to enter data on the aquifer}
252 begin
253   write('Enter the confined storage coefficient '); S:=ReadReal(1);
254   write('Enter the unconfined specific yield '); Sy:=ReadReal(1);
255   write('Discharge rate? (m^3/day) '); Q:=ReadReal(1);
256   write('Horizontal hydraulic conductivity? (m/day) '); Kr:=ReadReal(1);
257   write('Vertical hydraulic conductivity? (m/day) '); Kv:=ReadReal(1);
258   write('Enter the saturated thickness of the aquifer (m) ');
259   ThickAq:=ReadReal(1);
260   write('Distance between discharge well and piezometer? (m) ');
261   R:=ReadReal(1);
262   writeln('   The simulation will start at one minute (1/1440 days).');
263   write('What time to terminate the simulation? (days) ');
264   MaxTime:=ReadReal(2);
265 end; {Procedure GetInput}
266
267 Procedure CalcDdVectors; {Calculate the drawdowns, by all methods}
268 begin
269   T:=Kr*ThickAq; FirstTime:=0.0007; Time:=FirstTime;  I:=1;
270   Psi:=(Kv/Kr)*(sqr(R)/Sqr(ThickAq)); MinDif:=1e30;
271   writeln('Psi = ',Psi:7:5);
272   if (Psi<MinPsi) or (Psi>MaxPsi) then
273     writeln('WARNING: extrapolating for Psi=',Psi:8);
274   writeln('   Below are calculated values for each method used.');
275   writeln('   Time       Dd A    Dd B  Dd Theis-Jacob  MinDif');
276   while Time<=MaxTime do
277   begin
278     TimeVec[I]:=Time; uAOut:=false; uBOut:=false;
279     uA:=sqr(R)*S/(4*T*Time);
280     if (uA>MinuA) and (uA<MaxuA)
281     then begin
282       WuA:=InterpWuA(1/uA,Psi); DdVec1[I]:=Q*WuA/(4*Pi*T)
283     end
284     else begin
285       uAOut:=true;
286       WuA:=0; DdVec1[I]:=0
287     end;
288     uB:=sqr(R)*Sy/(4*T*Time);
289     if (uB>MinUb) and (uB<MaxuB)
290     then begin
291       WuB:=InterpWuB(1/uB,Psi); DdVec2[I]:=Q*WuB/(4*Pi*T)
292     end
293     else begin
294       uBOut:=true;
```

```
295       WuB:=0; DdVec2[I]:=0
296     end;
297     u:=sqr(R)*Sy/(4*T*Time); Wu:=WellFunc(u);
298     DdVec3[I]:=UnconfinedDd(Q*Wu/(4*Pi*T));
299     RateVec[I]:=Q;
300     if (DdVec1[I]<>0) and (DdVec2[I]<>0) and
    (abs(DdVec1[I]-DdVec2[I])<MinDif)
301     then begin
302       MinDif:=abs(DdVec1[I]-DdVec2[I]); MinDifRecNo:=I
303     end;
304     write(Time:9:4,' ',DdVec1[I]:9:3,' ',DdVec2[I]:9:3,' ',DdVec3[I]:9:3,
305       '  ',MinDif:8);
306     if uAOut=true then write('  uA not tabled');
307     if uBOut=true then write('  uB not tabled');
308     writeln;
309     if (I mod 17 = 0) and (Time<MaxTime) then
310     begin
311       writeln('Press any key to procede.');
312       Ch:='!'; repeat read(kbd,Ch) until Ch<>'!';
313       writeln('   Time       Dd A     Dd B  Dd Theis-Jacob  MinDif');
314     end;
315     I:=Succ(I); Time:=Time*TimeInc;
316     if (Time>MaxTime) and (Time<MaxTime*TimeInc) then Time:=MaxTime;
317   end; {while time}
318   NumData:=I-1;
319 end; {Procedure CalcDdVectors}
320
321 Procedure InterpLastJoin; {Interpolate section joining DdVec2 and 3}
322 var
323   I: integer;
324 begin
325   I:=MinDifRecNo;
326   while (I<=NumData) and (DdVec2[I+1]<>0) do I:=I+1;
327   LastuB:=I;
328   if DdVec2[LastuB]>DdVec3[LastuB]*1.1 then
329     writeln('CAUTION; last uB drawdown significantly greater than
    Theis ',
330     'at same time!');
331   if LastuB+6<=NumData then
332   begin
333     IndepVar[1]:=Ln(TimeVec[LastuB-1]); DepVar[1]:=DdVec2[LastuB-1];
334     IndepVar[2]:=Ln(TimeVec[LastuB]);   DepVar[2]:=DdVec2[LastuB];
335     IndepVar[3]:=Ln(TimeVec[LastuB+5]); DepVar[3]:=DdVec3[LastuB+5];
336     IndepVar[4]:=Ln(TimeVec[LastuB+6]); DepVar[4]:=DdVec3[LastuB+6];
337     for I:=LastuB+1 to LastuB+4 do
338       DdVec4[I]:=Lagrange(IndepVar, DepVar, 4, Ln(TimeVec[I]));
339   end; {if LastuB}
340 end; {Procedure InterpLastJoin}
341
342 Procedure CalcDdVecNo4; {Fill this vector with the final drawdowns
    values}
343 var
344   J: integer;
345 begin
346   for I:=1 to MinDifRecNo-2 do DdVec4[I]:=DdVec1[I];
347   IndepVar[1]:=Ln(TimeVec[MinDifRecNo-3]);
    DepVar[1]:=DdVec1[MinDifRecNo-3];
348   IndepVar[2]:=Ln(TimeVec[MinDifRecNo-2]);
    DepVar[2]:=DdVec1[MinDifRecNo-2];
```

```
194   Simulation (3)

349   IndepVar[3]:=Ln(TimeVec[MinDifRecNo+2]);
     DepVar[3]:=DdVec2[MinDifRecNo+2];
350   IndepVar[4]:=Ln(TimeVec[MinDifRecNo+3]);
     DepVar[4]:=DdVec2[MinDifRecNo+3];
351   for I:=MinDifRecNo-1 to MinDifRecNo+1 do
352     DdVec4[I]:=Lagrange(IndepVar, DepVar, 4, Ln(TimeVec[I]));
353   I:=MinDifRecNo+2;
354   while (DdVec2[I]<>0) and (I<=NumData) do
355   begin
356     DdVec4[I]:=DdVec2[I]; I:=Succ(I);
357   end;
358   if I<=NumData then
359   repeat
360     DdVec4[I]:=DdVec3[I]; I:=Succ(I);
361   until I>NumData;
362   if Psi<0.1 then InterpLastJoin;
363   MaxDd:=DdVec4[NumData];
364   writeln('Press any key to procede.');
365   Ch:='!'; repeat read(kbd,Ch) until Ch<>'!';
366   if LastuB+6>NumData then
367   begin
368     writeln('File truncated because insufficient duration for joining
     of',
369       ' Theis data.');
370     NumData:=LastuB;
371   end;
372   writeln('Below are the final values.');
373   writeln('Rec.    Time     Drawdown        Rec.   Time    Drawdown');
374   for I:=1 to (NumData+1) div 2 do
375   begin
376     J:=I*2;
377     if J-1<=NumData then
378     begin
379       write(J-1:4,' ',TimeVec[J-1]:9:4,' ',DdVec4[J-1]:9:3,'       ');
380       if J<=NumData
381         then writeln(J:4,' ',TimeVec[J]:9:4,' ',DdVec4[J]:9:3)
382         else writeln;
383     end;
384     if (I mod 20=0) and (J<NumData) then
385     begin
386       writeln('Press any key to continue');
387       Ch:='!'; repeat read(kbd,Ch) until Ch<>'!';
388       writeln('       Time   Drawdown')
389     end;
390   end;
391   writeln('End of data.  A simple semilog plot will follow.');
392   writeln('Press any key to continue.');
393   Ch:='!'; repeat read(kbd,Ch) until Ch<>'!';
394 end; {Procedure CalcDdVecNo4}
395
396 Procedure SimplePlot; {Produce a simple graph of the time-drawdown data}
397 begin
398   TimePlotFac:=639/(Ln(MaxTime)-Ln(FirstTime));
399   TimePlotConst:=-(Ln(FirstTime)*TimePlotFac);
400   DdPlotFac:=199/MaxDd;
401   HiRes;
402   for I:=1 to NumData do
403   begin
404     plot(round(Ln(TimeVec[I])*TimePlotFac+TimePlotConst),
```

```
405     round(DdVec4[I]*DdPlotFac),1);
406   end;
407   Ch:='!'; repeat read(kbd,Ch) until Ch<>'!';
408   TextMode;
409 end; {Procedure SimplePlot}
410
411 {#}begin {Main part of program Neuman}
412   textcolor(green); FileName:='';
413   TestType:=Simulation; WellType:=Observation;
414   GetNatLogs;
415   repeat
416     ClrScr; writeln('Neuman''s Unconfined Well Function');
417     writeln; GetInput; CalcDdVectors;
418     if MinDifRecNo+3>NumData then Error:=true else Error:=false;
419     if Error<>true
420     then begin
421       CalcDdVecNo4; SimplePlot; textcolor(green);
422       write('Do you want to save the data? ');
423       if Response('YN')='Y' then
424       begin
425         for I:=1 to NumData do TimeVec[I]:=TimeVec[I]*1440;
426         SaveData(TimeVec, DdVec4, RateVec, TestType,
427           WellType, R, 1, NumData);
428       end;
429     end {if Error<>true}
430     else begin
431       writeln;
432       writeln('The simulation terminated too early to interpolate
    correctly.');
433       writeln('Try a longer maximum time.');
434     end; {else Error}
435     write('Do you want another run? ');
436     if Response('YN')='Y' then Again:=true else Again:=false;
437   until Again=false;
438   {The code below should only be used when this program is used as a
439    chain file.}
440   ChainTo('GWMENU.CHN',IOCode);
441   if IOCode<>0 then
442   writeln('Unable to chain to GWMENU.CHN!');
443 {#}end.
```

7. REFERENCES

Bouwer, H., 1978. Groundwater Hydrology. McGraw-Hill Kogakusha Ltd. 480pp.

Clarke, D.K., 1987. Microcomputer Programs for Groundwater Studies. Developments in Groundwater Science, 30. Elsevier, Amsterdam/Oxford/New York/Tokyo. 268pp.

Cooper and Jacob, 1946. A generalised graphical method for evaluating formation constants and summarizing well field history. Trans. Am. Geoph. Union 27: pp. 526-534.

Freeze, R.A., and Cherry, J.A., 1979. Groundwater. Prentice-Hall Inc., New Jersey, USA. pp604.

Hildebrand, H.I., 1968. Interpolation and Extrapolation. Encyclopaedia Britannica, Vol. 12, p445.

Marino, M.A., and Luthin, J.N., 1982. Seepage and Groundwater. Developments in water Science. 489 pp. Elsevier.

Neuman, S.P., 1972. Theory of flow in unconfined aquifers considering delayed response of the water table. Water Resour. Res., 8:1031-1045.

Neuman, S.P., 1973. Supplementary comments on theory of flow in unconfined aquifers considering delayed response of the water table. Water Resour. Res., 9:1102-1103.

Neuman, S.P., 1975. Analysis of pumping test data from anisotropic unconfined aquifers considering delayed gravity response. Water Resour. Res., 10:303-312.

Theis, C.V., 1935. The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using groundwater storage. Trans. Am. Geophs. Un 16: pp519-524.

Chapter 5

Some aquifers behave as if they are confined or leaky confined in the short to medium term, but unconfined in the very long term; due to the pumped confined aquifer being supplied from an overlying unconfined aquifer through an aquitard. I will call this case one. A very similar drawdown curve will be produced by discharge from an unconfined aquifer showing delayed yield. I will call this case two. (These two situations are shown diagrammatically in Figures 4.2a and 4.2b in the previous chapter.) It is not within the scope of this book, nor within the ability of the author, to explain how one may decide from discharge test data whether the former or latter case is being dealt with in a particular instance. Perhaps the only possible basis for a decision would be on stratigraphic grounds as indicated by drilling samples or geophysical logs.

Chapter two described how program DRAWDOWN could be used to produce a simulation of drawdown in a piezometer in a leaky confined aquifer, and also in an unconfined aquifer assuming instantaneous release of water from storage. At very late times during a discharge test the delay caused by leakage through an aquitard becomes negligible; the whole system, aquifers and aquitards, behaving as a single unconfined aquifer. Therefore these simulations could be combined to cover case one. Chapter four described program NEUMAN which used Neuman's unconfined well function to simulate drawdown in an unconfined aquifer with delayed yield (case two above).

The program described in this chapter, JOINWTD, may be used to combine a leaky confined aquifer simulation with an unconfined aquifer simulation, so modelling discharge from an aquifer of the type of case one above. Before running this program one must therefore have produced disk files of data from a leaky confined model of early to later time response, and an unconfined model of very late time response, using program DRAWDOWN. The lower drawdown curve in Figure 4.3 was produced by this method.

1. AN EXAMPLE OF THE USE OF PROGRAM JOINWTD

For this example run it will be necessary to first store on disk file the data for two simulations which are to be joined. Both simulations must be produced by using program DRAWDOWN. The first should be the leaky confined aquifer simulation, the entries for which are given in Table 4.2. Save the output data from this simulation under the name of Leaky (either as a 'fast', or 'human readable' file). The second should be an unconfined

simulation, again based on the data of Table 4.2. This set of output data should be placed in a file named Unconf.

Terminate program DRAWDOWN, return to the primary menu, and select program JOINWTD. The program will now ask you for the file name of "The leaky confined aquifer simulation". Reply by typing 'Leaky'. This file will be loaded, and you will be asked if you want to view the data. Whether or not you choose to do so will have no effect on the remainder of this exercise. You will next be asked to give the name of "The unconfined aquifer simulation", reply with 'Unconf'. Again the file will be loaded, and again you will have an opportunity to view the data.

JOINWTD will now carry out a test on the data, although you will not see anything of this because the test will be successful. If the curves of the two sets of data had not been found to cross then you have been informed, and the operation would have been aborted. The program expects the unconfined aquifer drawdowns to be less than the leaky confined drawdowns at early times, but to be greater at later times. If this is not so, then further progress is impossible.

You will now be asked a question which requires some explanation. The program will cause the following message to be displayed.

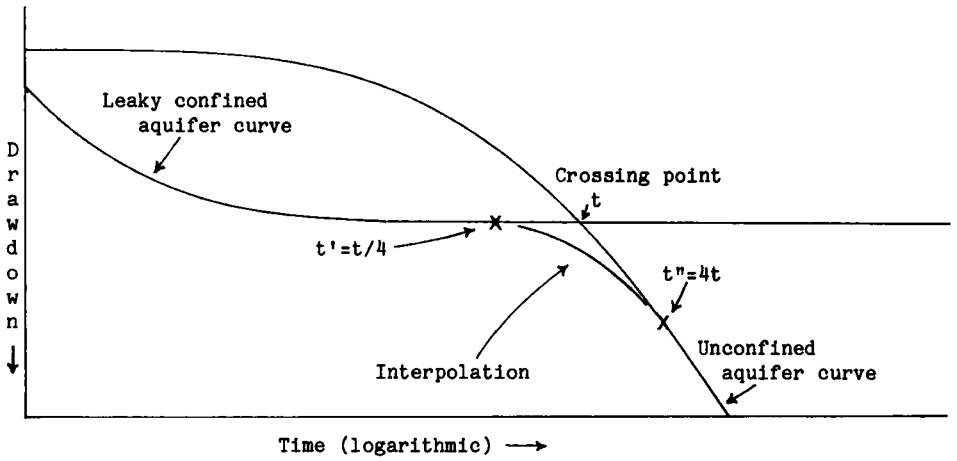
"The default times for the beginning and ending of the interpolated section are a factor of 4 from the crossing time. Do you want to use the default?"

This can best be explained by reference to Figure 5.1. The program calculates the crossing time, t, as the time corresponding to the point on the graph where the two curves intersect. A point, t', is chosen on the leaky confined aquifer curve at the time t/4. A second point, t", is chosen on the unconfined aquifer curve at the time 4t. The aim now is to calculate data for the time period t' to t" by interpolation of the data of the curves immediately outside of this segment. This is the application of the 'time factor' requested above, and if the default of 4 is not suitable, then you are free to enter some other values. (If you choose to enter values manually, then you will be asked for a division factor and a multiplication factor for calculation of t' and t" respectively.)

If the default is chosen then the program sets pointers to the record numbers immediately outside of the section marked on the graph. The values of the pointers are displayed as soon as they are calculated. In the case of the example data the two pointers will be given the values 30 and 43. This means that the interpolated data will be calculated from the time and drawdown values of records 29, 30, 43, and 44. These times and drawdowns are

displayed by the program, and are given below.

Figure 5.1



Location of points used for producing an interpolated section to join a leaky confined drawdown curve (early to later times) to an unconfined drawdown curve (very late times).

Record No.	Time (min.)	Drawdown (m)
29	2479	1.615
30	3124	1.615
43	63032	2.273
44	79416	2.377

The times are marked as independent variables, and the drawdowns as dependent variables by the program as drawdown is a function of time.

Had you entered smaller numbers as the 'time factors', then the interpolated section would have been smaller; larger factors would lead to a larger interpolated section. The default of 4 was chosen because it gives reasonable results in most cases. It is worth noticing that the interpolation would be impossible if the simulation had finished much before 70 000 minutes. In the case of the example data the end is at 400 days or 576 000 minutes, so there is plenty of lee-way.

Eight times are calculated, evenly spaced on a logarithmic scale, within the section to be interpolated. The interpolated data are then calculated, and displayed. The values calculated for the example program run are given below.

Time (min.)	Drawdown (m)
4362	1.629
6091	1.657
8505	1.7
11875	1.757
16582	1.829
23153	1.917
32329	2.019
45142	2.138

The three sections of the data: the early leaky confined section, the interpolation section, and the late unconfined section, are now joined together, and the final time/drawdown values are displayed. Lastly, you will be given the opportunity of saving the data to a disk file.

2. DESCRIPTION OF PROGRAM JOINWTD BY PROCEDURES AND FUNCTIONS

The following section of this chapter will describe the operation of program JOINWTD by it's component parts. The main, or controlling part, will be first, followed by the procedures and functions in alphabetical order.

Main part of program JOINWTD Line 122

This program uses three time/drawdown/discharge rate matrices: one for the leaky confined simulation, one for the unconfined simulation, and one for the final joined data. As 500 records are reserved as the standard for all such matrices, memory requirements are quite high. Hence line 123 creates space in the Heap area for the three matrices. (Please refer to the Turbo Pascal manual for information on the Heap.)

Lines 132 and 133 call procedure ReadTestDataFile (this procedure is within include file ReadSave.prc) to load the data of the leaky confined simulation. If the user is unable to give this procedure a valid file name, or path and file name, then he may enter 'x' instead of a file name and leave ReadTestDataFile without loading a file. In this case NumData[1] will have the value of zero, and line 134 will redirect program execution to procedure Exit. This procedure, in turn, will cause termination of JOINWTD and return to the primary menu. From the primary menu the user may call for a directory and find out what file names are available.

Procedure ViewAlterData, called from line 135, and also in include file ReadSave.prc, allows the viewing of the file and the alteration of any

specified datum. The file containing the unconfined aquifer simulation is read by the second call to ReadTestDataFile in line 138.

If the distance between the discharging well and the piezometer in the two simulation is dissimilar, then plainly the combined simulation will be invalid; so this is tested in line 143.

Line 148 calls procedure GetCrossPoint to find the record numbers of the two files at the point at which the drawdown of the unconfined simulation first exceeds that of the leaky simulation. If no such crossing point is found the the Boolean variable, Error, is given a value of true and the problem is reported in lines 149 to 153.

The time factors for selection of the extent of the interpolated section was discussed earlier in this chapter. If the user does not want to use the default he may enter some other values in lines 164 and 166.

The time at which the two simulations crossed is calculated by the call to FindCrossTime. Note the difference between this and the call to GetCrossPoint above. GetCrossPoint finds record numbers, this procedure calculates a crossing time.

Some further testing for errors is carried out from line 177 to 195, before the preparations for interpolation begin at line 199. At this point the four independent variables and the four dependent variables which will be used for the Lagrangian interpolation are placed in two vectors to be accessed by function Lagrange.

The final data will be placed in the third set of vectors, VRP[3] (VRP is the acronym for 'vector record pointer'). The early data, that up to the beginning of the section to be interpolated, are placed into the three vectors in lines 217 to 222. The later time data will be placed into the third set of vectors as it is calculated.

In line 223, the variable Factor is given a value that, when multiplied eight times by the time pointed to by Pointer1, will produce a series of times that may be used to fill in the gap between Pointer1 and Pointer2. This can be elucidated by giving an example. From the data produced by the demonstration run we have the variable Pointer1 pointing at the time 3124 minutes (app.), and Pointer2 is pointing at the time 63032 minutes. The natural log of the quotient of 63032 divided by 3124 is 3.0045, so the variable Factor gets the value 1.3963. Multiplying this with 3124 eight times gives the series:

4362,	6091,	8505,	11875,
16582,	23153,	32329,	45142.

(And once again gives 63032, getting back to the time of Pointer2.) We can now use Factor to produce an exponential series of times to use in the interpolation of drawdowns. Note that if more interpolated time/drawdown values were required this could be achieved very simply. For example, if we wanted ten interpolated values rather than 8, we would change the 9 in lines 223, 237, and 241 to 11, and change the 8 in line 228 to 10.

Lines 228 to 236 use the method given above to produce times, which are then placed in VRP3. In line 231 variable X is given the current time value, and passes it to the Lagrangian interpolation routine, which returns a drawdown in line 232. The current time, and associated drawdown are displayed as they are calculated by lines 234 and 235.

The last part of VRP3 is filled in by copying data from the latter part of VRP2 in lines 237 to 246. Finally the data of VRP3 is displayed on the screen, and an opportunity is given to save it to disk file. If you wish to view your finished example run data as a graph, then save it and go to PLOTWTD.

The main section calls: procedures ReadTestDataFile, ViewAlterData, GetCrossPoint, FindCrossTime, and functions Response, ReadReal (both of file First.seg) and Lagrange.

Exit procedure Line 105

Purpose: to exit from JOINWTD and return to the primary menu.

There is no way back from this procedure to the main part of JOINWTD, so the memory used by the three sets of vectors VRP[1] to VRP[3] is freed by the statements of line 106.

If NumData[1] or NumData[2] equalled zero it would indicate that either the user opted out of the file loading section, probably because no data file could be found, or the two input simulations where found to not cross. In this case the message of line 110 will be displayed.

If either program GWMENU.CHN can not be found by the operating system, or if this program is compiled and run in memory, IOCode will have a non zero value and the section of the procedure from line 115 to 117 will take over. In the former case this will lead to a return to the operating system, in the latter case the return will be to the Turbo Pascal main menu.

Called by: the main part of program JOINWTD.

Calls: procedure ChainTo of include file First.seg.

FindCrossTime procedure Line 54

Purpose: to produce a close approximation of the time at which the drawdown of the unconfined model first exceeded that of the leaky artesian model.

The method used for producing the time of intersection is as follows. The times and drawdowns of the records immediately before and after the crossing time have been marked by a preceding procedure, GetCrossPoint. These are stored in the variables given in table 5.1.

Table 5.1

	Leaky model	Unconfined model
Preceding time	LTime1	UTime1
Following time	LTime2	UTime2
Preceding drawdown	LDd1	UDd1
Following drawdown	LDd2	UDd2

The assumption is made that the drawdown curve between the relevant points is linear, so the linear equations for the two curves are calculated and expressed in the slope - intercept form by program lines 58 to 61. (Divergence from linearity over such a small section of data will be negligible for these purposes.) Line 63 then calculates the crossing time by simultaneous solution of the two equations, and line 63 is then able to calculate the drawdown at that time.

Called by: the main part of program JOINWTD.

GetCrossPoint procedure Line 32

Purpose: To locate and flag the records, of both simulations, that are to either side of the time at which the drawdown of the unconfined simulation begins to exceed that of the leaky artesian simulation.

Line 35 moves the second pointer, Pointer2, up to the first record in the unconfined data having a non zero drawdown. (Pointer1 relates to the leaky artesian simulation, and Pointer2 to the unconfined simulation.) Line 36 now moves Pointer1 up to approximately the same time in the leaky simulation data.

The important section of this procedure is the loop from line 38 to 47. The aim here is to make Pointer1 and Pointer2 point at the last records, in their respective sets of vectors, where leaky confined drawdown is less than unconfined drawdown. To achieve this it is necessary to increment both pointers, as nearly as possible, together. Line 40 increments Pointer2, and lines 41 and 42 ensure that Pointer1 keeps up. Line 43 checks that the end of the data has not been reached, and sets the error flag if this is not so.

Finally, the times and drawdowns for the records to either side of the crossing point are stored in the variables designated for that purpose, in lines 48 to 51. These variables, and their functions, are given in Table 5.1 above.

Called by: the main part of program JOINWTD.

Lagrange function Line 66

Purpose: to interpolate the drawdowns for the section of simulation which provides a smooth transition from the leaky confined simulation to the unconfined simulation.

This routine is identical to that of the same name in program NEUMAN. As it was discussed at some length in the previous chapter it will not be explained again here.

Called by: the main part of program JOINWTD.

3. KEY LINES OF PROGRAM JOINWTD

```
  5 {#}{$I First.seg}
 19 {#}{$I ReadSave.prc}
 32 Procedure GetCrossPoint; {Get the record numbers at the crossing point}
 54 Procedure FindCrossTime; {Find the crossing time by linear
     interpolation}
 66 Function Lagrange {Lagrangian interpolation}
105 Procedure Exit; {End the program and return to the primary menu}
122 begin {# Main part of program}
265 {#}end.
```

4. LISTING OF PROGRAM JOINWTD

```
  1 Program JOINWTD_PAS; {For joining two discharge test drawdown simulated
  2  data files to produce a three part drawdown curve}
  3 {$R+}
  4
  5 {#}{$I First.seg}
  6
  7 type
  8   VecRec=record
  9     TimeVec, LogTimeVec, RootTimeVec, DdVec,
 10     LogDdVec, RateVec: MainVec;
 11   end;
 12
 13 var
 14   NumData: array[1..3] of integer;
 15   Distance: array[1..3] of real;
 16   VRP: array[1..3] of ^VecRec;
 17   TestType: array[1..3] of Test; WellType: array[1..3] of Well;
 18
 19 {#}{$I ReadSave.prc}
 20
 21 Type
 22   ShortArray=array[1..4] of real;
 23 var
 24   Pointer1, Pointer2: integer;
 25   NumOfPoints, Result, X1: integer;
```

```
26   DivFac, MultFac, X: real;
27   Factor, CrossTime, CrossDd, LTime1, LTime2, UTime1, UTime2: real;
28   LDd1, LDd2, UDd1, UDd2: real;
29   Ch: Char;
30   IndepVar, DepVar: ShortArray;
31
32 Procedure GetCrossPoint; {Get the record numbers at the crossing point}
33 begin
34   Pointer1:=1; Pointer2:=1;
35   while VRP[2]^.DdVec[Pointer2]=0 do Pointer2:=Pointer2+1;
36   while VRP[1]^.TimeVec[Pointer1]<VRP[2]^.TimeVec[Pointer2]
37     do pointer1:=Pointer1+1;
38   while VRP[2]^.DdVec[Pointer2]<VRP[1]^.DdVec[Pointer1] do
39   begin
40     Pointer2:=Pointer2+1;
41     while (VRP[1]^.TimeVec[Pointer1+1]<=VRP[2]^.TimeVec[Pointer2]) and
42       and (Pointer1<=NumData[1]+1) do Pointer1:=Pointer1+1;
43     if (Pointer1>NumData[1]) or (Pointer2>NumData[2])
44     then begin
45       VRP[2]^.DdVec[Pointer2]:=1e6; Error:=true;
46     end;
47   end;
48   LTime1:=VRP[1]^.TimeVec[Pointer1-1]; LTime2:=VRP[1]^.TimeVec[Pointer1];
49   UTime1:=VRP[2]^.TimeVec[Pointer2-1]; UTime2:=VRP[2]^.TimeVec[Pointer2];
50   LDd1:=VRP[1]^.DdVec[Pointer1-1]; LDd2:=VRP[1]^.DdVec[Pointer1];
51   UDd1:=VRP[2]^.DdVec[Pointer2-1]; UDd2:=VRP[2]^.DdVec[Pointer2];
52 end; {Procedure GetCrossPoint}
53
54 Procedure FindCrossTime; {Find the crossing time by linear
    interpolation}
55 var
56   Slope1, Slope2, Yint1, Yint2: real;
57 begin
58   Slope1:=(LDd2-LDd1)/(LTime2-LTime1);
59   Slope2:=(UDd2-UDd1)/(UTime2-UTime1);
60   YInt1:=-LTime1*Slope1+LDd1;
61   YInt2:=-UTime1*Slope2+UDd1;
62   CrossTime:=(YInt2-YInt1)/(Slope1-Slope2);
63   CrossDd:=YInt2+Slope2*CrossTime;
64 end; {Procedure FindCrossTime}
65
66 Function Lagrange {Lagrangian interpolation}
67 (IndepVar, DepVar: ShortArray; NumOfPoints: byte; InputX: real):real;
68 var
69   Factor: integer;
70   Subtrahend: ShortArray;
71   Numerator, Denominator, Minuend, SumOfTerms: real;
72   Flag, I: byte;
73 begin
74   SumOfTerms:=0;
75   for I:=1 to NumOfPoints do     {For each term}
76   begin
77     Flag:=0;
78     for Factor:=1 to NumOfPoints-1 do
79     begin
80       if Factor=I
81       then begin
82         Minuend:=IndepVar[I]; Flag:=1;
83       end;
```

```
 84       Subtrahend[Factor]:=IndepVar[Flag+Factor];
 85     end;
 86     if I=NumOfPoints then Minuend:=IndepVar[NumOfPoints];
 87     Numerator:=InputX-Subtrahend[1];
 88     Denominator:=Minuend-Subtrahend[1];
 89     if NumOfPoints>2
 90     then begin
 91       for Factor:=2 to NumOfPoints-1 do
 92       begin
 93         Numerator:=Numerator*(InputX-Subtrahend[Factor]);
 94         {Product of numerator factors up to and including factor No.
    "Factor"}
 95         Denominator:=Denominator*(Minuend-Subtrahend[Factor]);
 96       end;
 97     end; {If NumOfPoints > 2}
 98     {writeln('InputX=',InputX:8:3,'  Num=',Numerator:8:3,'  Den='
    ,Denominator:8:3);}
 99     SumOfTerms:=SumOfTerms+Numerator/Denominator*DepVar[I]
100   end; {for I}
101   Lagrange:=SumOfTerms;
102 end; {Function Lagrange}
103
104 Procedure Exit; {End the program and return to the primary menu}
105 begin
106   dispose(VRP[1]); dispose(VRP[2]); dispose(VRP[3]);
107   if (NumData[1]=0) or (NumData[2]=0) then
108   begin
109     writeln;
110     writeln('Exiting from Join ------');
111     delay(1000)
112   end;
113   ChainTo('GWMENU.CHN',IOCode);
114   if IOCode<>0 then
115   begin
116     writeln('Unable to chain to program GWMENU.CHN');
117     halt;
118   end;
119 end; {Procedure Exit}
120 {----- End of procedures and functions -----}
121
122 begin {# Main part of program}
123   new(VRP[1]); new(VRP[2]); new(VRP[3]);
124   clrscr; TextColor(green); Error:=false;
125   writeln('  This is a program to join two *.WTD (or *.FTD) files, the
    first');
126   writeln('being a simulation of drawdown in a leaky confined aquifer,
    and the');
127   writeln('second being a simulation of drawdown in an unconfined
    aquifer.');
128   writeln('The result will be a file showing all the stages from
    confined type');
129   writeln('drawdown, to fully unconfined.');
130   writeln;
131   writeln('The leaky confined aquifer simulation should be loaded
    first.');
132   ReadTestDataFile(VRP[1]^.TimeVec, VRP[1]^.DdVec, VRP[1]^.RateVec,
133     TestType[1], WellType[1], Distance[1], NumData[1]);
134   if NumData[1]=0 then Exit;
135   ViewAlterData(VRP[1]^.TimeVec, VRP[1]^.DdVec, VRP[1]^.RateVec,
```

```
136     TestType[1], WellType[1], Distance[1], NumData[1]);
137   writeln('The unconfined aquifer simulation should be loaded next.');
138   ReadTestDataFile(VRP[2]^.TimeVec, VRP[2]^.DdVec, VRP[2]^.RateVec,
139     TestType[2], WellType[2], Distance[2], NumData[2]);
140   if NumData[2]=0 then Exit;
141   ViewAlterData(VRP[2]^.TimeVec, VRP[2]^.DdVec, VRP[2]^.RateVec,
142     TestType[2], WellType[2], Distance[2], NumData[2]);
143   if Distance[1]<>Distance[2]
144   then begin
145     writeln('WARNING: The R value for the two simulations are
    dissimilar!');
146     delay(2000);
147   end;
148   GetCrossPoint;
149   if Error=true
150   then begin
151     writeln('No crossing point detected.  Data invalid?');
152     delay(4000)
153   end {of then}
154   else begin
155     writeln('   The default times for the beginning and ending of the ',
156       'interpolated');
157     writeln('section are a factor of 4 from the crossing time.  Do ',
158       'you want');
159     write('to use the default?');
160     Answer:=Response('YN');
161     if Answer='N'
162     then begin
163       write('Enter the factor for multiplication ');
164       MultFac:=ReadReal(1);
165       write('Enter the factor for division ');
166       DivFac:=ReadReal(1);
167     end {then}
168     else begin
169       MultFac:=4; DivFac:=4;
170     end; {if then else}
171     FindCrossTime; {Calculate exact time and drawdown of crossing}
172     writeln;
173     writeln('Time and drawdown at which unconfined drawdown exceded ',
174       'leaky drawdown');
175     writeln('CrossTime=',CrossTime:8:3,'  CrossDd=',CrossDd:8:3);
176     while (VRP[1]^.TimeVec[Pointer1]>CrossTime/DivFac) and (Error=false)
    do
177     begin
178       Pointer1:=Pointer1-1;
179       if Pointer1<2 then Error:=true;
180     end;
181     if Error=true
182     then begin
183       writeln('Leaky data does not start early enough.');
184       delay(4000);
185     end;
186     while (VRP[2]^.TimeVec[Pointer2]<CrossTime*MultFac) and
    (Error=false) do
187     begin
188       Pointer2:=Pointer2+1;
189       if Pointer2>NumData[2]-1 then Error:=true;
190     end;
191     if Error=true
```

```
192     then begin
193       writeln('Unconfined data does not go long enough.');
194       delay(4000)
195     end;
196   end; {if no error}
197   writeln('Pointers 1, 2 =',Pointer1,'  ',Pointer2); writeln;
198   if Error=false
199   then begin      {Set up for 4 point lagragian interpolation}
200     IndepVar[1]:=VRP[1]^.TimeVec[Pointer1-1];
201     IndepVar[2]:=VRP[1]^.TimeVec[Pointer1];
202     IndepVar[3]:=VRP[2]^.TimeVec[Pointer2];
203     IndepVar[4]:=VRP[2]^.TimeVec[Pointer2+1];
204     DepVar[1]:=VRP[1]^.DdVec[Pointer1-1];
205     DepVar[2]:=VRP[1]^.DdVec[Pointer1];
206     DepVar[3]:=VRP[2]^.DdVec[Pointer2];
207     DepVar[4]:=VRP[2]^.DdVec[Pointer2+1];
208     writeln('Data to be used for interpolation is:-');
209     for I:=1 to 4 do
210     begin
211       writeln('Indep=',IndepVar[I]:10:1,'  Dep=',DepVar[i]:8:3);
212       IndepVar[I]:=Ln(IndepVar[I]); DepVar[I]:=Ln(DepVar[I]);
213     end;
214     writeln;
215     {First part of output vectors.  The data up to Pointer1 remains as
216     it was in the leaky aquifer simulation.}
217     for I:=1 to Pointer1 do
218     begin
219       VRP[3]^.TimeVec[I]:=VRP[1]^.TimeVec[I];
220       VRP[3]^.DdVec[I]:=VRP[1]^.DdVec[I];
221       VRP[3]^.RateVec[I]:=VRP[1]^.RateVec[I];
222     end;
223     Factor:=exp((1/9)*Ln(VRP[2]^.TimeVec[Pointer2]/
224       VRP[1]^.TimeVec[Pointer1]));
225     TempVal:=Factor;
226       {Interpolation section.  Calculate the times for the interpolated
227          drawdowns, and call Lagrange to interpolate for drawdown.}
228     for I:=Pointer1+1 to Pointer1+8 do
229     begin
230       VRP[3]^.TimeVec[I]:=VRP[1]^.TimeVec[Pointer1]*TempVal;
231       X:=Ln(VRP[3]^.TimeVec[I]); TempVal:=TempVal*Factor;
232       VRP[3]^.DdVec[I]:=Exp(Lagrange(IndepVar, DepVar, 4, X));
233       VRP[3]^.RateVec[I]:=VRP[1]^.RateVec[I];
234       write('At time =',VRP[3]^.TimeVec[I]:10:1);
235       writeln('  Interp. Dd =',VRP[3]^.DdVec[I]:8:3);
236     end;
237     J:=0; NumData[3]:=Pointer1+9+(NumData[2]-Pointer2);
238     Delay(3000);
239      {The last part of the output data remains the same as it was in the
240       unconfined aquifer simulation, only the reading numbers are
    changed.}
241     for I:=Pointer1+9 to NumData[3] do
242     begin
243       VRP[3]^.TimeVec[I]:=VRP[2]^.TimeVec[Pointer2+J];
244       VRP[3]^.DdVec[I]:=VRP[2]^.DdVec[Pointer2+J];
245       VRP[3]^.RateVec[I]:=VRP[2]^.RateVec[Pointer2+J]; J:=J+1;
246     end;
247     writeln; writeln('Final data is:-');
248     for I:=1 to NumData[3] do
249     begin
```

```
250       write('I=',i:3,'  Time =',VRP[3]^.TimeVec[I]:10:1);
251       writeln('  Drawdown =',VRP[3]^.DdVec[I]:8:3,
252         '  Rate =',VRP[3]^.RateVec[I]:8:2);
253     end;
254     write('Do you want to save the data? ');
255     Short:='YN';
256     if Response(Short)='Y'
257     then begin
258       TestType[3]:=TestType[2]; WellType[3]:=WellType[2];
259       Distance[3]:=(Distance[1]+Distance[2])/2;
260       SaveData(VRP[3]^.TimeVec, VRP[3]^.DdVec, VRP[3]^.RateVec,
261       TestType[3], WellType[3], Distance[3], 1, NumData[3]);
262     end;
263   end; {if no error}
264   Exit;
265 {#}end.
```

Chapter 6

The IBM PC compatible computer has graphical capabilities which, while not being ideal, are quite good enough for displaying discharge test data curves. Screen graphics have the advantage of being quick to produce and being alterable, but the disadvantages of having a relatively low resolution, limited text options, and they are not easily transportable. Plotter graphics have high resolution, are easily transportable, but are slow to produce and cannot easily be altered. It is highly desirable, therefore, to have access to both screen graphics, and plotter graphics. Then one may experiment with screen graphics while examining data, or until the desired result is achieved; and use the plotter when a more permanent record, or a more detailed or aesthetic record is required. For these reasons graphical capabilities are provided to the GW set of programs by program PLOTWTD, the subject of this chapter. (More precisely, program PLOTWTD can be used for screen and plotter graphics with a Roland plotter or compatible, while program PLOTWTD2 can be used for screen graphics and with a Hewlett-Packard plotter or compatible.

1. HARDWARE REQUIREMENTS FOR GRAPHIC OUTPUT

For screen graphics an IBM compatible colour graphics adaptor will be needed with a colour or black and white monitor. (The subject of the next chapter, program ANALYZE, requires a colour monitor, although a black and white monitor may be found to suffice.)

Plotter graphics may be obtained on either a Roland DXY-880 compatible plotter (one using the DXY set of commands), or something similar to the Hewlett-Packard 7440A 'ColorPro' plotter. Two variants of program PLOTWTD are provided, one for each plotter type.

2. USING THE PROGRAM

As with all the other programs in this book, PLOTWTD is called from the primary menu of program GWMENU, which itself is reached by typing GW from the DOS level. Please read the section 'Getting started' in the Introduction if you are unsure how to do this.

If you have a Hewlett-Packard plotter there are a few simple steps that you will need to make before producing a plotter graph from the GW set of programs.

2.1. Altering the program names for a Hewlett-Packard plotter

The plotting program is called from GWMENU and that program expects to find a program named PLOTWTD.CHN, but the H-P version of PLOTWTD is named PLOTWTD2.PAS so will compile into PLOTWTD2.CHN. If you wish to use a Hewlett-Packard or compatible, then I suggest that you first rename PLOTWTD.PAS as PLOTWTD1.PAS, and then rename PLOTWTD2.PAS as PLOTWTD.PAS. When you compile your H-P plotting program as a chain file it will now be named by Turbo Pascal as PLOTWTD.CHN, as expected by GWMENU. Alternatively, if you order the programs on disk then please state the type of your plotter and request that the program be compiled for your plotter.

For those unfamiliar with the MSDOS operating system, the steps involved in renaming the files are given below. (The symbol <Enter> will be used to indicate pressing the Enter key on your computer. On some computers, this key may be labelled only with a bent arrow.)

1/ Assuming that you have the programs on disk, and both programs are on drive 'A:', type the following command. (Do not type the quotes.) "rename a:plotwtd.pas plotwtd1.pas", <Enter> (this renames file 'PLOTWTD.PAS' as 'PLOTWTD1.PAS', the file itself remains unchanged.)

2/ Then type "rename a:plotwtd2.pas plotwtd.pas" <Enter>.

The same process would be used to change the names of the chain files if they have been produced from the program source code as named and listed in this book.

Alternatively you could type the program listed here as PLOTWTD2.PAS into the Turbo Pascal editor under the name of PLOTWTD.PAS.

2.2. Running the program

One proceeds through the program mainly by answering multiple choice questions. The first of these asks for the device that is to be used to receive the graph; plotter, screen, or disk file. (The H-P version of the program produces direct output only to screen or disk file for reasons explained in the section 'Notes Specific to the H-P ColorPro Plotter'.)

2.3. Graphing devices

A graph may be sent, by the program, to any one of three devices, although only two of these will cause immediate production of a graph.

2.3.1. Screen graph

This is the medium you will probably use most often. It is fastest, and does not result in using reams of paper, most of which will shortly find it's way to the rubbish bin. By first checking you graphs on the screen you can make sure that they will show exactly what you want them to show when you do put them on paper.

2.3.2. Plotter graph

This option outputs instructions to a Roland plotter through the parallel port. As it is easy to produce graphs, you will probably find that you produce more than you need, and throw most away.

2.3.3. Disk file graph

Here the plotter instructions for producing a graph are sent to a disk file rather than being sent directly to the plotter. A graph can be produced at any later time by sending the disk file of instructions to the plotter. This is the easiest way of producing a number of identical graphs. Please refer to the section 'Sending disk file data to a plotter' later in this chapter for more information.

2.4. An example screen graph

For this example, please use program DRAWDOWN to produce two simulations from the following data. Name the first 'conf1' and the second 'conf2'.

Simulation one, 'conf1'.

Aquifer type, confined, infinite
A series of time/drawdown data
Time unit: minute
Distance from pumped well to piezometer: 15m
Transmissivity: 15 m^3/day
Storage coefficient: 0.0003
A single step
Finishing time: 480 min.

Simulation two, 'conf2'; identical to 'conf1' but with storage coefficient equal to 0.0004.

Now call PLOTWTD from the primary menu and you will be asked whether you want output to go to a plotter graph, screen graph, or to a disk file.

Reply by pressing S. You will now be asked for a file name, reply with 'Conf1'. This data file will be loaded, and you will be asked if you wish to see the data. Reply to this question by pressing 'n'. Reply to the "Another set of data?" question with 'y', and give the file name 'conf2'. When asked again whether you want to load another data file, reply 'n'. (For a screen graph the maximum is three files, the program will not let you go beyond that. This is because any more files on one screen are visually confusing.)

All the data to be graphed are now in memory, and you will be asked to choose the type of graph that you want from the list "Linear, Semi-log, lOg-log, Root-time". Make your decision known by pressing the appropriate key. Take care to press 'o' rather than '1' if you want a Log-Log graph. Whichever choice you make here the resultant graph can be one of those of Figures 6.1 to 6.4, so long as you follow the next two steps.

You will be informed that "Drawdown ranges from 0.021 to 2.715m" and "Time ranges from 1.000 to 480.000min.", and that "The graph will be based on these limits unless you enter other limits manually.". The question "Do you want to alter any drawdown or time limits?" will then be displayed. If you were to answer this question with a 'y' you would be able to produce a graph of some selected part of your data, or you could make the field of the graph much larger than that required to contain your data. Answering 'n', as is required for this example, makes the program base the scaling of the graph on the limits of your data.

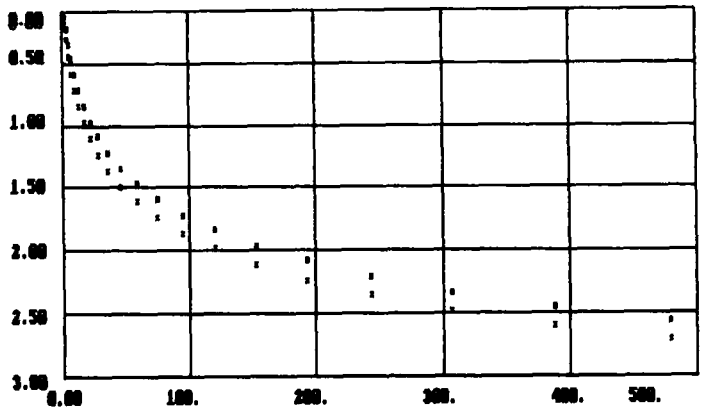
The final question before the graph is produced is "Join data points?". For the example answer this with 'n'. If answered with 'y', then a line will be drawn between consecutive data points, unless there is a change of discharge rate between the particular pair of points concerned.

The screen will now go into high density graphics mode, the first drawdown curve will be drawn, and the message "Press space bar to proceed" will be displayed at the bottom of your screen. A touch of the space bar will cause the second curve to be drawn, and another touch will finish the graph by the addition of the reference lines. Pressing any key will return you to the text screen and the question "Another graph?".

2.5. Graph types available on the screen

The linear graph is the simplest, on it both the drawdown and time scales will be linear. Drawdown increases toward the bottom of the screen; time, toward the right. See Figure 6.1.

Figure 6.1



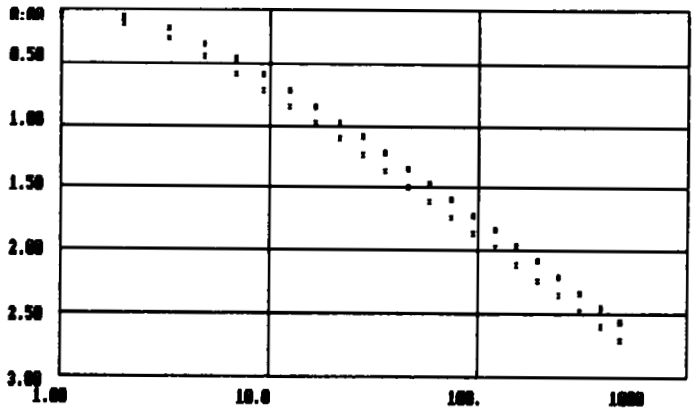
A double linear graph of the data produced by the two simulations, 'conf1', and 'conf2'. The lower curve is conf1 (S=0.0003) and the upper is conf2 (S=0.004). The plotting symbols are allocated in order of graphing, the first curve has small x shapes, the second solid rectangles, and if there was a third it would have diamond shapes.

Figures 6.1 to 6.4, and similar figures elsewhere in this book, were produced by use of a commercial program designed to copy graphics from the screen to a dot matrix printer. The result is not as good as that from a plotter as the limitations of the screen graph are transposed to the graph on paper. For example, the 500 at the right bottom of the graph cannot be placed much further to the right without causing the screen to scroll with disastrous effects to the graph. The slight distortion at the top of the graph is due to errors in the transposing operation, not in the program that placed the graph on the screen.

In my experience a double linear graph applied to discharge test data is most useful when used to observe periodic noise which may show up in a long discharge test; for example barometric or tidal effects.

In the semilogarithmic graph the drawdown also increases linearly toward the bottom of the screen, but the time increases logarithmically toward the right. See Figure 6.2. The semilogarithmic graph is very useful because of the straight line that forms in a homogeneous aquifer at later times, the slope being inversely proportional to the transmissivity. This graph can show you what part of the data may validly be used for the calculation of transmissivity.

Figure 6.2



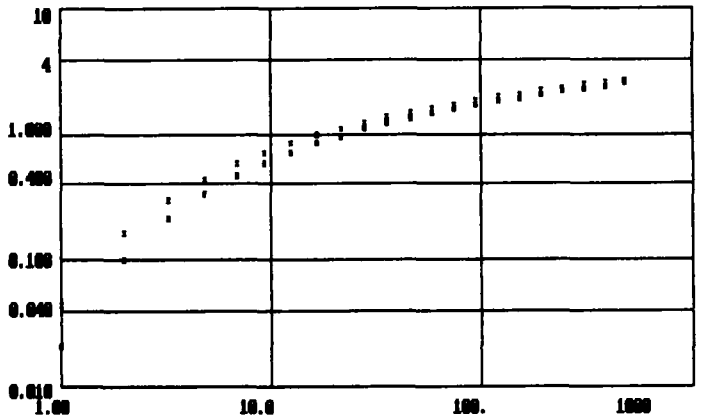
The same two data files as in Figure 6.1, but here plotted on a semi-logarithmic scale. Note the parallel straight lines from around 20 minutes onward.

The log-log graph has both axes logarithmic. Among the graphs produced by PLOTWTD, this one is exceptional in that drawdown increases toward the top. As usual, time increases toward the right. See Figure 6.3. This type of graph has traditionally been used for matching of type curves. I suspect that greater use of computers in future discharge test analysis will reduce the call for type curves.

The log-log graph can be useful in recognition of strip aquifer response as the later part of the curve will be straight and will have a slope of 0.5 so long as the strip is water tight, see Figure 6.7.

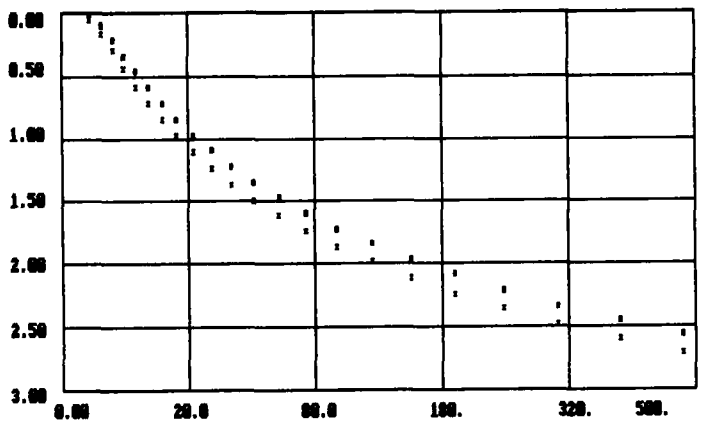
In the Root Time graph, drawdown increases linearly downward, and time increases, as the square root, toward the right. This graph would most likely be used for suspected strip aquifers where a straight line at later times would indicate a strip having negligible leakage through the walls. Any upward curvature in the latter part of the 'straight' line is a possible indication of leakage through the walls of the strip. See Figure 6.4 and also Figure 6.8.

Figure 6.3



A graph of the same data as Figures 6.1 and 6.2, but with both scales logarithmic.

Figure 6.4



The data of example files 'conf1' and 'conf2' plotted against the square root of time. Note that, except for very early times, there is a continuous upward trend to the two curves. This is to be expected for data from an unbounded aquifer on this type of graph.

2.6. Output to a plotter

Very much the same steps must be followed here as were given in the example of the production of a screen graph. There is a complication if you have a computer with two parallel ports, and you have the printer plugged into the first port, and the plotter in the second. Turbo Pascal seems to

have no facilities for sending output to the second parallel port rather than the first. There are at least two answers to this problem, neither of them ideal.

1/ Unplug the printer from the first port, and plug in the plotter whenever you want to produce a graph.

2/ Instead of producing a graph by direct output from PLOTWTD to the plotter, send the data to a disk file, exit from the GW set of programs, and get DOS to send the disk file of data to the second parallel port. You can re-enter the GW programs while DOS goes on sending the plotting instructions to your plotter.

The second alternative is the one that I would recommend, it is not as complicated as it may seem. For details see the section on 'Sending disk file data to a plotter' below.

The steps involved in producing a plotter graph on a Roland plotter connected to the first parallel port, or a disk graph, are very much the same as those in producing a screen graph. If the graph is being produced for a Roland plotter, either directly or indirectly, then it is necessary to specify whether it should be drawn on large (A3) or small (A4) paper. This is not necessary in the case of a Hewlett-Packard plotter, as only an A4 size graph may be produced.

2.6.1. Standard scale log-log graph

If a log-log graph with a scale of 76mm per log cycle is required for curve matching purposes, then it can only be done by the Roland program, and is drawn on large paper. These programs carry out semi-automatic scaling for all other graphs. If your data are not able to be fitted on one such graph, because of the limitations of the number of cycles that will fit on one sheet of A3, you will be able to specify which parts you want graphed. You will then be able to produce a second graph containing those parts that would not fit onto the first, and join the two paper graphs together.

2.7. Graph types available on a plotter

The same four basic types of graph are available on a plotter as on the screen: double linear, semilogarithmic, log-log, and square root of time. In addition, on the Roland plotter, two graph sizes are available, A3 (297mm x 420mm), and A4 (210mm x 297mm). Only an A4 size graph is available on a H-P plotter simply because only an A4 sized plotter was available for program development.

An example consisting of four graphs produced by a plotter and reproduced in monochrome is given in Figures 6.5 to 6.8 below. All these were originally A4 size. All except the log-log graph (Figure 6.7) were produced by a H-P plotter, that was produced by a Roland plotter.

The linear graph (Figure 6.5) is of pumped well data from a simulated four stage discharge test. The data are the same as those used to produce the screen graph from which Figure 1.1 was produced. (Figure 1.1 is a semilogarithmic graph, and is on page 42.)

Figure 6.5 was drawn with the option of a line between points being requested. The three gaps where no line is drawn are due to changes in discharge rate at those places, the connecting line is never drawn between two points separated by a change in discharge rate. This rule allows the production of a sound graph from a file including t/t1 recovery data with drawdown data and with the 'joined points' option on.

Figure 6.6 is a semilogarithmic graph of a set of simulations of steady rate discharge from aquifers ranging from unbounded homogeneous to a water tight strip. All aquifers are confined and homogeneous within the confines of the boundaries (where there are boundaries).

The upper curve of Figure 6.6 is for an infinite, unbounded aquifer, and uses the following data:

Distance from discharging well to piezometer, 30m
Transmissivity, 300 m^3/day/m
Storage coefficient, 0.0002
Discharge rate, 500 m^3/day
Duration of discharge, 6 days (8640 min.)

The lower curve is for a water tight strip aquifer and uses the same data with the addition of the specifications of the strip configuration:

Distance from discharging well to boundary, 100m
Distance from piezometer to boundary, 110m
Width of strip, 230m

The remaining curves are transitional between these two extremes. They are all for strip aquifers having significant transmissivity outside of the strip and good hydraulic connection across the strip boundaries. These curves were produced from the same data as the water tight strip above, except for this factor. Beginning with the second curve from the bottom and working upward the transmissivity outside the strip is: 5, 10, 20, 50, and 120m^3/day/m. Of course the lowest curve corresponds to a transmissivity

outside of the strip of 0, and in the upper curve the transmissivity is 300m^3/day/m, just as it is inside the strip.

In producing Figure 6.6 the program was instructed to use implied, rather than full, reference lines. The pairs of characters "s0", "s1", etc. across the top of the graph are names under which the data files were saved at the time the simulations were produced. These file names were given to PLOTWTD so that the data could be retrieved from file for production of this graph. The file names are printed on the original graph in the same colour as are the corresponding curves, of course this will not be apparent in the monochrome print. The original has each curve in a different colour.

The log-log graph of Figure 6.7 uses some of the same simulations as were used for Figure 6.6.

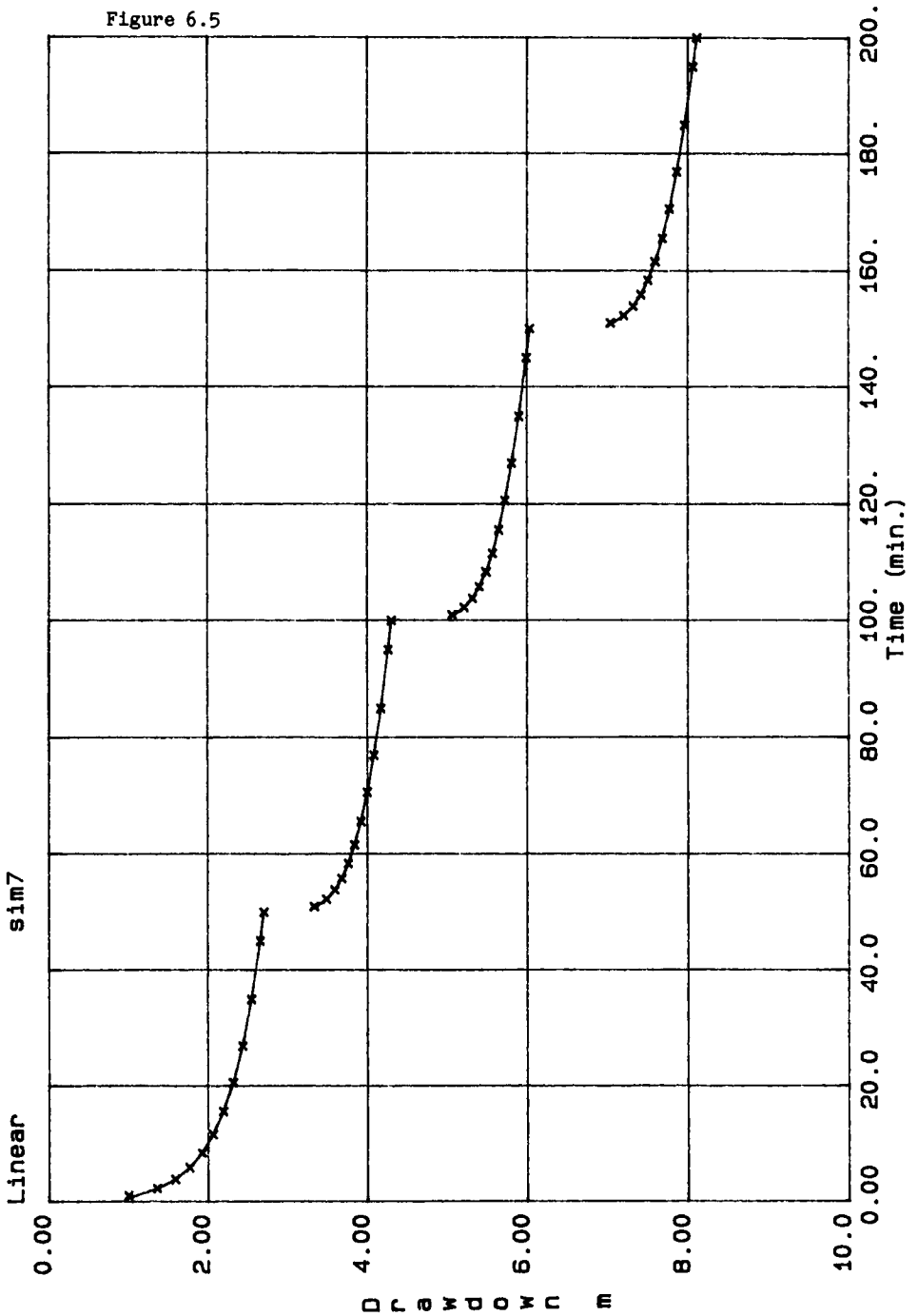
The file names are different because the simulations were produced at different times. This graph is the only one of Figures 6.5 to 6.8 which was produced by a Roland DXY-880 plotter, the others having been produced by a Hewlett-Packard ColorPro. Here the program was instructed to draw full reference lines, considerably increasing the time taken for production of the graph. The time difference between the full reference lines and the implied reference lines in the case of the H-P plotter is much less significant.

Although the maximum drawdown plotted on this graph is only about 12 metres, the top of the graph corresponds to 100 metres drawdown. The A4 graphing routines always draw a full log cycle.

The upper curve is the water tight strip aquifer simulation, the second curve is for transmissivity outside the aquifer (T_2) equal to 10m^3/day/m, the third had T_2=50, and the lower curve is the simple confined aquifer. Notice that the upper curve has a slope of 0.5 from about 100 minutes onward, as is to be expected for a water tight strip aquifer.

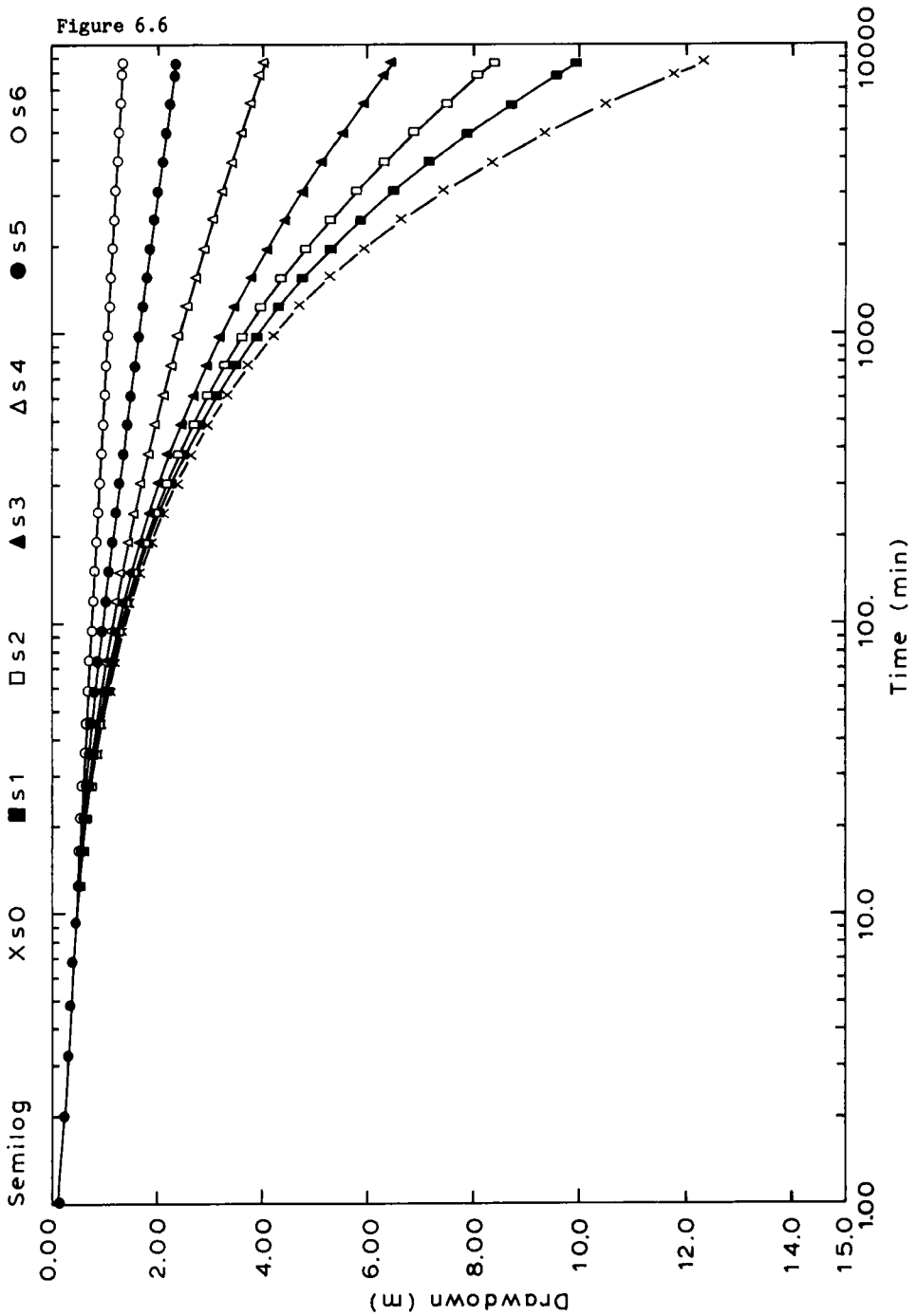
The last graph in the group is the square root of time graph, Figure 6.8. This has three curves of the same data as were used for Figure 6.6; from the bottom they are the water tight strip aquifer, the strip with leaky boundaries and T_2=5, and the upper curve is for T_2=20. Note that the latter part of the lower curve is a straight line, again to be expected for a strip aquifer with water tight boundaries.

Figure 6.5



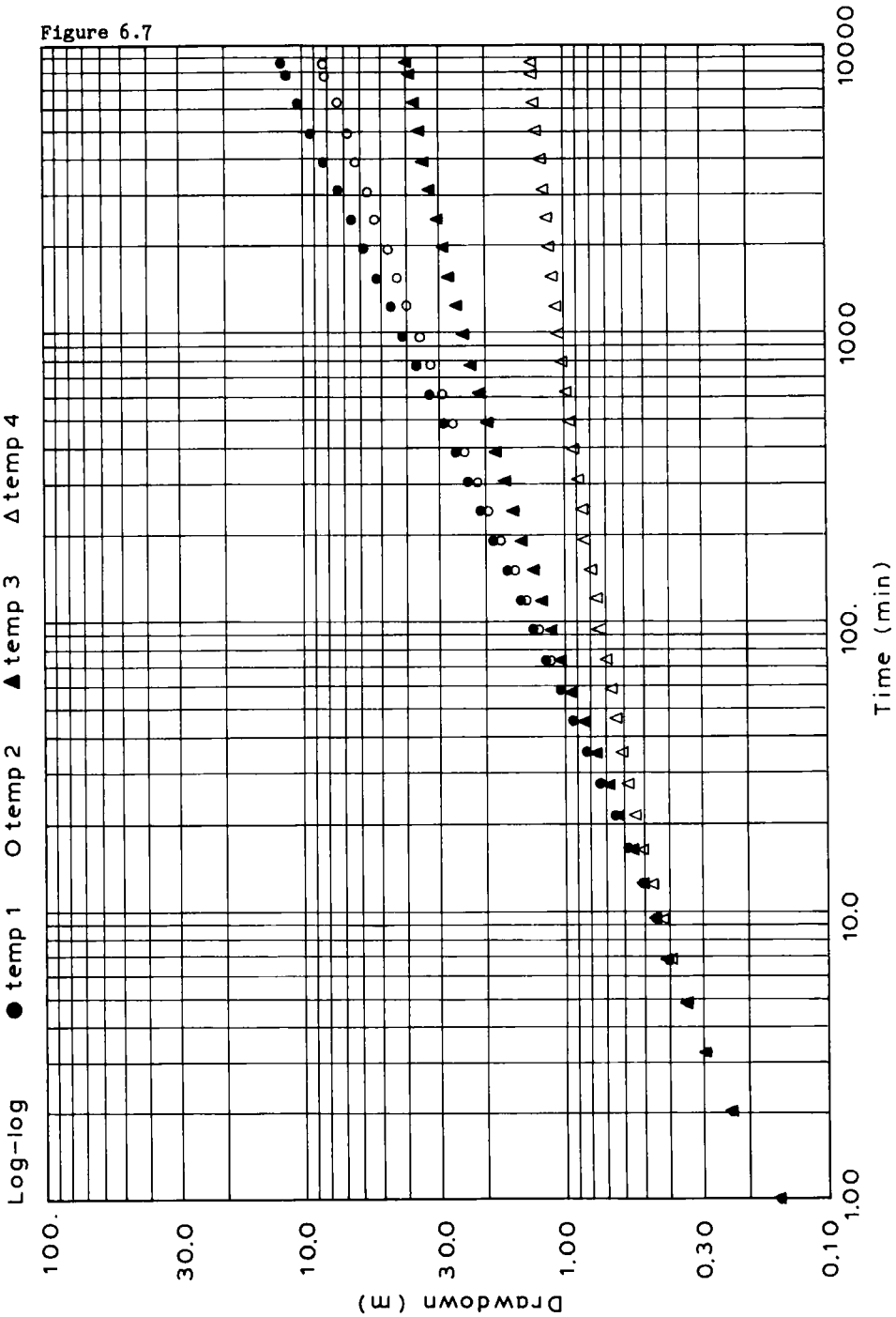
A double linear plotter graph of a four stage drawdown simulation for the discharging well, see text for details. Full reference lines and joined points were specified.

Figure 6.6



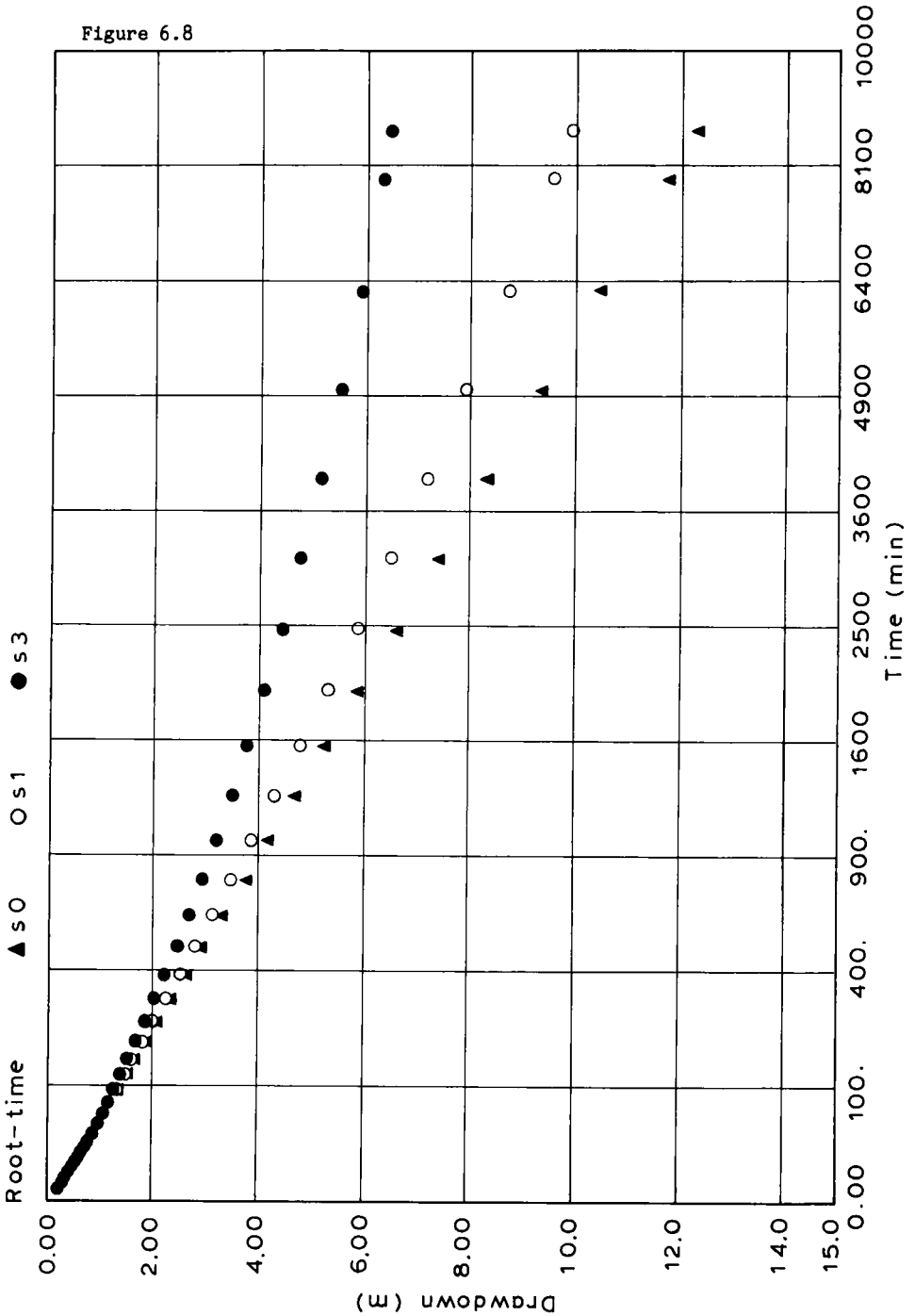
A semilogarithmic plotter graph of seven sets of simulated data, see text for details. Implied reference lines and points not joined.

Figure 6.7



A double logarithmic plotter graph of four files of simulated discharge test data, see text for details. Full lines and points not joined.

Figure 6.8



A square root of time plotter graph of three sets of simulated discharge test data, see text for details. Full lines and points not joined.

2.8. Scaling of graphs

As mentioned above, a large (A3) log-log graph will always be produced at the scale of 76mm per log cycle, other graphs do not have a fixed scale. All small (A4) graphs are designed to make the most use of the paper upon which they are printed, and to be suitable for binding into a report.

The maximum and minimum times and drawdowns on the graph may be chosen manually or automatically. As with the screen graph option, you will be informed of the extremes of the data, and asked whether you want these limits used for the graph, or enter other limits. Whether the limits of the data, or the limits that you enter, are used, round numbers will be selected for the limits of the graph. Log scales will always start and end with the start and end of a full log cycle, except in the case of the large log-log graph where this is not possible because of the set scale. All linear or square root graph maximums will be made equal to some power of ten multiplied by one of the numbers: 1, 1.2, 1.5, 2, 2.5, 3, 4, 5, 6, 7, 8, or 9. Linear scale reference lines and graph minimums will at most times be selected from the same set, but negative drawdowns may necessitate going outside of the set.

A moderate change in the size of a graph may be easily achieved by a slight modification to the program. There are some constants which fix the coordinates of the top, bottom, left, and right boundaries of the graphs. (These are listed under the section on constants in this chapter.) All plotted points, reference lines, reference values, and graph text are placed relative to these coordinates. An A4 graph (Roland version) can be made 10mm narrower, for example, by simply changing PlotBS from 960 to 1060.

2.9. Sending disk file data to a plotter

It is important to note that once the disk file of plotter instructions has been produced, none of the programs in this book are required, or used, to send those instructions to the plotter. These programs therefore have no control over the sending of the instructions to the plotter, it is entirely up to the operating system of your computer. For the first steps in the operation to send a file of plotter instructions to a H-P plotter see the section of notes specific to that plotter.

To send a disk file of plotter instructions to a Roland plotter connected to a parallel port, first leave the GW group of programs and return to the DOS ready prompt. Type 'print filename' where filename is the name that you gave program PLOTWTD when you created the file of plotter instructions. DOS uses a program file (which you received with your copy of DOS) called PRINT.EXE for this purpose, so it (DOS) must be able to find both this and

your plotter instruction file if the command is to be carried out. The most likely cause for this command to fail is if one or other of the files is on some disk drive or directory which DOS has not been instructed to search. Program PLOTWTD will place your plotter instruction file on the default drive and directory unless otherwise instructed. I would expect your copy of PRINT.EXE to be on your DOS disk in drive A, or in the root directory of your hard drive, if you have one. If the command 'print filename' fails, then you might try 'A:print filename' if you have a two floppy drive system, or '\print filename' if you have a system that includes a hard drive. If this also fails to find the files then refer to the section on 'disks files and directories' in your DOS manual.

When you successfully call program PRINT.EXE it will respond with "Name of list device [PRN]: ". Here it is asking whether you want to send the output to the default device (PRN, the device connected to the first parallel port), or to some other device. Using the Roland version of this program, the only other device you may want to output to would be that on the second parallel port. If your computer has two parallel ports then one (probably the first) may be connected to the printer, and the other to the plotter. In this case you would answer the above question with "LPT2" (meaning line printer number 2). This will cause DOS to attempt to send the plotter instructions from the file to your plotter. If your plotter is connected to port number one, then simply respond by pressing the Enter key.

After having answered the above question and pressed the enter key, DOS will respond with "Resident part of PRINT installed". At this time, if DOS is unable to find the plotter instruction file that you named, it will respond with a message to that effect and return you to the DOS ready mode.

If the file was found then "filename is currently being printed" will be displayed, and DOS will send the instructions to the device that you specified. You may immediately go on with other work, and DOS will carry on 'in the background'.

3. NOTES SPECIFIC TO THE ROLAND DXY-880 PLOTTER

The following instructions apply to an IBM PC computer or similar, connected to a Roland DXY-880 plotter, or compatible, via a Centronics type parallel port.

Set the DIP (Dual In-line Package) switches on the plotter as shown below;

Switch bank No. 1; all switches off.

Switch bank No. 2; switches 1-3 on,

switches 4-6 off,
switch 7 on,
switches 8,9 off,
and switch 10 on.

Note that the DIP switch settings should be changed while the plotter is turned off, as any change made to them while it is on will probably be ignored. (It seems that devices such as plotters and printers check the settings on their DIP switches only at the time the power is first switched on, at 'boot up'.)

Clip a piece of paper of either A3 (297mm x 420mm) or A4 (210mm x 297mm) size into upper left position of the plotter with the long dimension of the paper running left-right. The black pen should be placed in position number 1, other pens may be placed in positions of your choice so long as there is a full sequence of pens starting at position one and carrying on sufficiently far to allow each curve to be plotted in a different colour. The black pen will be used to draw the surrounds of the graph and the first curve, pens of increasing position number for each subsequent curve. Before the plotter is switched on the pen carrier must be moved to the home position; as far to the left and the bottom as it will go. Place the magnetic strip paper holders along the right and lower edges of the paper, and smooth out any parts of the paper that are not lying against the plotting surface. (This will not be necessary with the Roland DXY-980, as the static pad produces very good smoothing of the paper.)

4. NOTES SPECIFIC TO THE H-P COLOR PRO PLOTTER

Hewlett-Packard were kind enough to loan me a 'ColorPro' plotter so that I could write a version of PLOTWTD to suit the HP-GL instruction language (program PLOTWTD2). The plotter had no model number on it, nor was the manual any help in this regard, but the plotter was capable of telling the computer it's model number, which turned out to be 7440A. This model handles paper up to something slightly larger than A4 size (297mm x 210mm).

Given this maximum paper size, the A3 plotting options, in particular the log-log at 75mm log cycles, had to be dropped. There were, therefore, two major sets of changes to the program; those associated with the change from the DXY (Roland) to the HP-GL instruction set, and the removal of the A3 plotting options. Unfortunately the necessary alterations affected so many parts of the program that a simple listing of the changes themselves would be almost as long as the program; so rather than including such a list in this

book, I have reproduced all those parts of the program that differ in any way from the Roland version.

Any reader wishing to modify this program in it's production of graphs on a Hewlett-Packard plotter will require H-P programming manuals in addition to the operators manual that came with the plotter. The 'HP-GL Programmers Reference Manual' gives some information, but is not complete. The much larger 'Interfacing and Programming Manual' will be found to be essential for any substantial changes. (The Roland plotter, on the other hand, comes complete with programming manual.)

The H-P 7440A has the disadvantage of not possessing a Centronics parallel port (as is available and most commonly used on printers and plotters), but instead uses the RS232 serial port. It seems that Turbo Pascal does not have an instruction to output to a serial port; list device output defaults to the parallel port. For this reason program PLOTWTD2 must be used in a somewhat roundabout way in comparison to PLOTWTD. When it is desired to produce a plotter graph, instruct the program to output to a disk file. After producing as many disk file 'graphs' as are required it will be necessary to exit from the program back to DOS and send the instructions from the files to the plotter at that level. There are several simple steps in this operation.

4.1. Configuring your system for the H-P plotter

1/ Tell DOS what Baud rate to use by typing 'Mode com1:96' <Enter>. DOS should reply with 'COM1: 9600,e,7,1, -', which means that the first serial port has been set to operate at 9600 Baud, with Even parity, a 7 bit word, and one stop bit. (The ColorPro comes from the factory with it's DIP switch set for 9600 Baud, but it might be worth checking on this setting.)

2/ Instruct DOS to send the plotting instructions from the disk file to the plotter by use of the PRINT program that came with you copy of DOS. See the notes on 'Sending disk file data to a plotter' earlier in this chapter.

4.2. Instruction set differences

The following notes not only concern differences between the plotters, but also to a lesser extent, differences in the ways in which I have chosen to program the plotters. With both of the plotters, there is more than one way of achieving the desired result.

1/ Both plotters are capable of plotting to Cartesian coordinates. The Roland uses increments of tenths of a millimetre, while the H-P uses fortieths of a millimetre. This is the cause of the greater values of the constants PlotL, PlotR, and PlotT in program PLOTWTD2.

2/ Where the Roland uses 'm500,1000' to move the pen to coordinates x=500 (50mm from the extreme left) and y=1000 (100mm from the bottom), the H-P uses 'pa2000,4000' to achieve the same result. (The Roland instruction 'm' stands for 'move', while the H-P instruction 'pa' stands for 'plot absolute'.)

3/ Given that the instruction of point 2 above has been carried out, and that the user wished to draw a line 10mm long in the 3 o'clock direction, the Roland would now use 'd600,1000' (draw to x=600, y=1000), while the H-P would use 'pd' (pen down), 'pa2400,4000', and then probably 'pu' (pen up). Alternatively the H-P could use 'pd', 'pr400,0' (plot relative x=400, y=0).

4/ For the purpose of printing text the Roland plotter uses the instruction 'p' (for print) followed by the text string. The end of the message to be printed is marked by a line feed, which is sent by Turbo Pascal at the end of the text file line. In the HP-GL instruction set 'lb' is the instruction to print text (label), and the end of the string must be marked by character number three.

5. A DEFINITION OF SELECTED VARIABLES AND CONSTANTS

5.1. Variables

SameFiles: Boolean; indicates when the same files are to be reused for the next graph.

Join: Boolean; indicates when plotted points are to be joined with a line on either paper or screen.

NumOfCurves: byte; the number of curves to be plotted on the current graph.

PlotR, PlotB: integers: respectively, the x and y plotter addresses for the current graphs. These will have values dependent upon the size of the paper. (These are used as variables in program PLOTWTD, and as constants in program PLOTWTD2.)

MaxDd, MinDd, MaxTime, MinTime: reals; the upper and lower limits of either the data of all the current curves, or of interest to the user. Compare to MaxGraphDd (etc.) and MaxLogGDd (etc.).

LogDdRange, LogTimeRange: reals; the log of the ranges covered by a large paper log-log graph. ie. LogDdRange equals the log (base ten) of (the maximum graphable drawdown divided by the minimum graphable drawdown).

MaxGraphDd, MinGraphDd, MaxGraphTime, MinGraphTime: reals; the maximum and minimum values for graphing.

MinLogGDd, MaxLogGDd, MinLogGTime, MaxLogGTime: reals; the maximum and minimum logs of values to be graphed.

5.2. Constants

PlotLt=220, PlotRL=3620, PlotRS=2360; (Program PLOTWTD, Roland plotter) Respectively, the plotter x addresses of the left side of the graph (both large and small paper), right side for large paper, and right side for small paper.

PlotL=1000, PlotR=9500; (Program PLOTWTD2, H-P plotter.) The plotter x coordinates of the left and right edges of the graph.

PlotBL=140, PlotBS=960, PlotT=2520; (Program PLOTWTD, Roland plotter) Respectively, the plotter y addresses of the bottom of the graph for large paper, the bottom for small paper, and the top (for both large and small paper.

PlotB=500, PlotT=6900; (Program PLOTWTD2, H-P plotter) The plotter y coordinate of the bottom and top edges of the graph.

Left=48; right=639; Top=0; Bottom=185; The screen x, y, addresses of the graph boundaries.

LnTen=2.302585093; The natural logarithm of ten; used for calculating logs to the base ten.

MaxFiles=7; The maximum number of discharge test data files to be read into memory. This number could be varied by the user to suit computer memory limitations, file sizes, and the number of different colours available on his plotter. (The amount of available memory in your computer is very dependent upon whether or not you have defined part of your memory as a RAM disk; whether, and how many, memory resident programs are present; etc.)

6. A DESCRIPTION OF PLOTWTD BY PROCEDURES AND FUNCTIONS

The first part of the following description concerns the main, or controlling, part of program PLOTWTD, and the remainder deals with each procedure and function of that program in alphabetical order. Program PLOTWTD2 (for the H-P plotter) is very similar, so it is not explained separately.

Main part of the program Line 1175

The constant MaxFiles limits the number of discharge test data files

that may be loaded into memory at one time. The value given to this variable has a very great effect on the memory requirement of the program. It is assigned a value of seven, as this was found to be a reasonable limit given the amount of memory available in the computer on which the program was developed (an IBM XT clone). Factors affecting the optimal value for this constant are:

1/ The amount of RAM in the users computer.

2/ The number and size of any memory resident programs that may be in the computer at the time of program execution.

3/ The size of the simulated disk drive in RAM if one is present.

4/ The dimensioning of MainVec in line 6 of file FIRST.SEG. This has been set to accommodate a maximum of 500 time/drawdown/discharge rate records, but users may find it necessary, or desirable, to alter this value.

5/ The number of pens in the users plotter. There is no point in increasing MaxFiles beyond this.

Space is reserved in the Heap for the maximum number of files in line 1176. If memory is insufficient, Turbo will terminate program execution, and give an appropriate error message (heap/stack collision, run time error number FF) at this point.

A loop begins in line 1180 allowing production of as many graphs as may be required in one program run. If a second graph is requested in one program run then the user will be asked if he wants to graph the same files as are currently in memory. An affirmative answer to this question will result in the Boolean variable SameFiles being given a true value in line 1201.

As the number of files that may be graphed on a plotter is limited only by the value of MaxFiles and the number of files that may be graphed on the screen is limited to three, line 1185 makes sure that even if there are more than three files in memory, no attempt is made to plot greater than three on the screen at one time.

The normal exit from this program is by chaining back to program GWMENU, but if DOS is unable to find this file, or if program PLOTWTD has been compiled in memory, this will not be possible and the message of line 1217 will be given.

Calls: procedures PlotterGraph, ScreenGraph, DiskGraph, and ChainTo; and functions CapOptions and Response of file FIRST.SEG

DiskGraph procedure Line 1135

Purpose: to control the sending of a set of instructions, for the production of one graph, to a disk file.

Line 1137 arranges for the reading of a set of discharge test data files from disk if those files which are still in memory from an immediately previous graph are not to be used in the production of the next graph.

Lines 1140 to 1143 fix the value of the variables that will be used to ensure that an A3 size log-log graph on paper will be produced at the standard scale of 76mm per log cycle. (Remember that the Roland plotter plotting addresses are spaced at ten per millimetre.)

Called by: procedure PlotterGraph and the main part of program PLOTWTD.

Calls: procedures GetSetOfFiles, PrepForGraph, GraphMaxMin, OpenGraphFile, PlotRefLines, and PaperPlot.

ExpTen function Line 64

Purpose: to calculate the inverse of the logarithm to the base ten.

This function has been explained in a previous chapter.

Called by: procedures GraphMaxMin, PlotLogTimeRL, and PlotLogDdRL; and subprocedures GetDdMaxMin, GetTimeMaxMin, LogDdLines, and LogTimeLines.

FileExist function Line 46

Purpose: to check for the existence of a file by the name given.

This function is very similar to function Exist which is in files Save.Prc, Read.Prc, and ReadSave.Prc, and is explained on page twelve in the Preliminary section.

Called by: procedure OpenGraphFile.

Format2 function Line 70

Purpose: to produce a string representation of a given length from a given number.

There are some limitations in the Turbo Pascal numerical formatting routines, and this functions attempts to make up for some of these. For example, Turbo has no direct facility for producing a four character representation of any number between 0.01 and 9999 or between -0.1 and -999.

This function will round where possible, but if a number is too large to fit the requested length string, the whole integer part of the number will be retained and rounding will not take place. Some examples:

Original No.	format	output	Original No.	format	output
0.023	4	0.02	0.026	4	0.03
12345.7	4	12345	3456.7	4	3457
65.432	4	65.4	-6.45	4	-6.5

If the original number is negative, then that fact is recorded in line 75 and the sign of the number is changed so that the function will have to format only positive numbers. In this case space must be reserved for later addition of the minus sign, so the variable Leng is reduced by one.

Line 76 produces a string form of the number with a total length of 20 and Leng-2 decimal places. This causes Pascal to round the last digit. The rounding will be retained if the number is less than ten. In most cases the string form of the number at this point will contain quite a few spaces, these are removed by lines 77 and 78.

At this stage, if the absolute value of the original number was less than ten, it's string form length will now be acceptable and the test of line 79 will be failed. If the number is greater than ten and has a value which will allow it to be represented non exponentially by the given string length the test of line 80 will be passed. Line 82 calculates the number of decimal places that may be represented considering the magnitude of the number, and the number is appropriately converted to a string by Turbo Pascal in line 84. Line 86 truncates the decimal point and anything further right if the string form of the number is longer than Leng characters. Finally line 87 replaces the sign in numbers that are less than zero.

Called by: procedures PlotLinDdRL, PlotLinTimeRL, PlotLogTimeRL, and PlotRootTimeRL.

Calls: function Log.

GetAFile procedure Line 989

Purpose: to control the loading of a discharge test data file.

The section of code from line 995 to 1001 calls the directory option of batch file GW.BAT if the user exits procedure ReadTestDataFile without loading a file. If execution passes back to DOS by this means then it can only return to program PLOTWTD via program GWMENU.

If a data file is successfully loaded, then procedure ViewAlterData allows the user to check the data and alter no more than one value.

Called by: procedure GetSetOfFiles.

Calls: procedures ReadTestDataFile and ViewAlterData, as well as the batch file GW.BAT.

GetDdMaxMin subprocedure Line 269

Purpose: to calculate the maximum and minimum drawdowns best suited for the data and a standard scale (76mm per log cycle) log-log graph.

MinLogGDd has previously been set in procedure GraphMaxMin and LogDdRange has been set to the log of the range of drawdowns plotable on an A3 sized log-log graph. The sum of these two gives the log of the maximum graphable drawdown on a graph beginning with MinLogGDd. This may not cover all the significant drawdowns in the data, and if this is so then the user is informed of the situation and asked to enter a new minimum drawdown for the graph (line 286). After validation of the entry the maximum graphable drawdown is calculated from the entered value.

Called by: subprocedure LogDdLines.

Calls: procedure ReadReal of file FIRST.SEG.

GetGraphType procedure Line 643

Purpose: to find out from the user what type and size of paper graph is required.

Lines 661 and 662 set the values of PlotR and PlotB to the appropriate plotter addresses for the right and bottom boundaries (respectively) of the graph.

Called by: procedure PrepForGraph.

Calls: functions CapOptions and Response of file FIRST.SEG.

GetMaxMin procedure Line 746

Purpose: to discover and record the maximums and minimums of the data to be graphed.

Called by: procedure PrepForGraph.

GetSetOfFiles procedure Line 1093

Purpose: to control the loading into memory of a set of discharge test data files.

Called by: ScreenGraph and DiskGraph.

Calls: procedure GetAFile, and function Response of file FIRST.SEG.

GetTimeMaxMin subprocedure Line 298

Purpose: to calculate the maximum and minimum times best suited for the data and a standard scale (76mm per log cycle) log-log graph.

This procedure is very similar in operation to GetDdMaxMin described above except in that drawdown is dealt with, while here the subject is time.

Called by: subprocedure LogTimeLines.

Calls: procedure ReadReal of file FIRST.SEG.

GraphMaxMin procedure Line 90

Purpose: to calculate the values to be used as the maximums and minimums of the graph, and the intervals to be used between the reference lines.

Line 96 defines the vector GraphStep as containing 1, 2, and 5; the only values permitted in the calculation of reference lines. These may be multiplied by some power of ten. Line 97 places all legal values for graph drawdown maximums, and time maximums for non logarithmic graphs, in the vector GraphRef. Again, in use these may be multiplied by a power of ten.

The procedure is divided into two main sections; the first, to line 140, deals with the drawdown scale, and the second section deals with the time scale.

The section from line 104 to 112 is typical of several sections of this procedure. The maximum graphable drawdown is decided by testing MaxDd (the maximum recorded drawdown) against GraphRef[x]*Multiplier while this product is progressively increased. The first product is 1*0.001. If this is less than MaxDd, then 1.2*0.001 is tried, then 1.5*0.001, etc. up to 9*0.001. If MaxDd still has not been exceeded, and in most cases it will not, multiplier is changed to 0.01 and GraphRef[x] goes back to 1, giving the product 1*0.01. This process repeats as long as is necessary. It is not the most efficient possible method of achieving the desired end, but speed is unimportant here.

Lines 114 to 124 find the graph drawdown reference line step in a very similar way, but here we are looking for a number that will give something in the order of eight reference lines for the graph.

The value MaxLogGDd referred to in line 131 is the log (base 10) of the proposed upper limit of the drawdown scale for a log-log graph. (MaxLogGDd is a constriction of 'the log of the maximum graphable drawdown'.) This, and MinLogGDd of line 135, may be adjusted by procedure GetDdMaxMin later. Line 132 sets the minimum graphable drawdown for the log-log scale to one millimetre even if the minimum drawdown in the data is less. MaxLogGTime and MinLogGTime are similarly set in the section of code from line 186 to 193.

Called by: procedures PrepForGraph and DiskGraph.

Calls: function Log and ExpTen.

LinDdLines subprocedure Line 403

Purpose: to produce all the plotter instructions necessary for the drawing of drawdown reference lines or implied reference lines, and label them as appropriate; and to label the drawdown scale of the graph.

This is a subprocedure of procedure PlotRefLines which begins on line 262.

This procedure is divided into two sections, the first (lines 409 to 433) will produce full reference lines, while the second (lines 434 to 458) will produced implied reference lines. (Full reference lines are lines right across the page, while implied lines are just small markers on each side of the graph.)

It would be simplest, from the programming point of view, to have all reference lines drawn in one direction, eg. from left to right. This would be inefficient for the plotter because the pen carrier would have to travel from left to right drawing a line, and then return to the left with the pen up, in readiness for drawing the next line. It is therefore desirable that the lines are drawn in alternate directions. Of course they cannot be labelled alternately, all drawdown reference line labels must be on the left side of the graph. This has been achieved by using the variable Tog to control the alternate execution of either lines 416 and 417 or lines 420 and 421.

Function Format2 is called to produce a string form of the drawdown labels with a length of four characters.

Lines 427 to 432 make sure that the bottom line, the x axis, of the graph is drawn, and that the corresponding drawdown is written against it's left end.

The implied reference lines are produced in much the same way as the full lines except that they are drawn on one side of the graph at a time, and it is only on the left side where labels need be written. This allows the code to be a little simpler.

Finally, the drawdown scale itself is labelled by lines 459 to 461. The "q1" of line 459 tells the plotter to rotate the printing 90 degrees anticlockwise, and the "q0" tells it to return to printing horizontally.

Called by: the main part of procedure PlotRefLines.

Calls: functions Format2 and LinearYP.

LinTimeLines subprocedure Line 464

Purpose: to produce all the plotter instructions necessary for the drawing of time reference lines or implied reference lines, and label them as appropriate; and to label the time scale of the graph.

As this procedure is very similar to LinDdLines described above, it will not be explained here.

Called by: the main part of procedure PlotRefLines.

Calls: functions Format2 and LinearXP.

LinearX function Line 812

Purpose: to calculate the plotting coordinate on the linear x scale on the VDU screen for a given time.

The time is passed to the function in the real variable Num, and the equation of lines 815 and 816 convert that to the x value of the screen address (in colour graphics mode) which corresponds to the given time.

Called by: procedures PlotData, PlotLinTimeRL, and PaperPlot.

LinearXP function Line 211

Purpose: to calculate the plotting coordinate on the linear x scale on the plotter for a given time.

This function operates in very much the same way as that above, except that this one deals with plotter addresses.

Called by: procedure PaperPlot and subprocedure LinTimeLines.

LinearY function Line 819

Purpose: to calculate the plotting coordinate on the linear y scale on the VDU screen for a given drawdown.

The drawdown is passed to the function in the real variable Num, and the equation of lines 822 and 823 convert that to the y value of the screen address (in colour graphics mode) which corresponds to the given drawdown.

Called by: procedures PlotData, PlotLinDdRL, and PaperPlot.

LinearYP function Line 218

Purpose: to calculate the plotting coordinate on the linear y scale on the plotter for a given drawdown.

This function operates in very much the same way as that above, except that this one deals with plotter addresses.

Called by: procedure PaperPlot and subprocedure LinDdLines.

LoadDdLog procedure Line 779

Purpose: to load a set of vectors (VRP[x]^.LogDdVec) with the logarithms (base ten) of the drawdown values of the set of discharge test data in memory.

Called by: procedure PrepForGraph.

LoadRootTime procedure Line 788

Purpose: to load a set of vectors (VRP[x]^.RootTimeVec) with the square roots of the time values of the set of discharge test data in memory.

Called by: procedure PrepForGraph.

LoadTimeLog procedure Line 770

Purpose: to load a set of vectors (VRP[x]^.LogTimeVec) with the logarithm (base ten) of the time values of the set of discharge test data in memory.

Called by: procedure PrepForGraph.

Log function Line 58

Purpose: to calculate the logarithm to the base ten of the argument passed to the function.

An explanation of this simple function was given in an earlier chapter.

Called by: procedures GraphMaxMin, GetMaxMin, LoadTimeLog, LoadDdLog, PlotData, PlotLogTimeRL, PlotLogDdRL, and PaperPlot; and subprocedures GetDdMaxMin, GetTimeMaxMin, LogDdLines, and LogTimeLines.

LogDdLines subprocedure Line 327

Purpose: to produce and label drawdown reference lines on the y scale of a log-log graph on a plotter.

There are four possible variations; the reference lines may be either implied or full, and the graph size may be large (A3) or small (A4). The scaling to suit the large or small graph is taken care of automatically by the values given to PlotB and PlotT (the bottom and top plotter addresses to be used for the graph) and the values of MinLogGDd and MaxLogGDd (the logarithms of the minimum and maximum graphable drawdowns). If the graph is large then the scale is fixed at 76mm per log cycle, so the subprocedure GetDdMaxMin is called from line 332 to produce special values for MinLogGDd and MaxLogGDd to cause bring about this scaling.

The remainder of the code of this subprocedure is broken up into two sections depending on whether the reference lines are full or implied. The

full reference lines section comes first, beginning at line 335. The plotting of the log drawdown reference lines is very similar to the plotting of the linear drawdown reference lines described above, with two exceptions. In the case of log reference lines the values increase as in the series 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30,.... The production of this series is taken care of by the code from line 355 to 358. The second exception is, of course, that the lines are plotted in different positions on the graph in the case of a log drawdown scale. This is taken care of by the call to LogYP in line 360 rather than to LinearYP.

Called by: the main part of procedure PlotRefLines.

Calls: procedure GetDdMaxMin and functions ExpTen, Format2, and LogYP.

LogTimeLines subprocedure Line 513

Purpose: to produce and label time reference lines on the x scale of a log-log graph on a plotter.

In operation this procedure is very similar to procedures LogDdLines and LinDdLines both of which have been explained above.

Called by: the main part of procedure PlotRefLines.

Calls: procedure GetTimeMaxMin and functions ExpTen, Format2, and LogXP.

LogX function Line 798

Purpose: to calculate the plotting coordinate on the logarithmic x scale on the VDU screen for a given time.

The time is passed to the function in the real variable Num, and the equation of lines 801 and 802 convert that to the x value of the screen address (in colour graphics mode) which corresponds to the given time.

Called by: procedures PlotData, PlotLogTimeRL, and PaperPlot.

LogY function Line 805

Purpose: to calculate the plotting coordinate on the logarithmic y scale on the VDU screen for a given time.

This procedure is very similar to LogX above.

Called by: procedures PlotData, PlotLogDdRL, and PaperPlot.

LogXP function Line 197

Purpose: to calculate the plotting coordinate on the logarithmic x scale on the plotter for a given time.

This procedure is very similar to LogX above.

Called by: procedure PaperPlot and subprocedure LogTimeLines.

LogYP function Line 204

Purpose: to calculate the plotting coordinate on the logarithmic y scale on the plotter for a given drawdown.

This procedure is very similar to LogX above.

Called by: procedure PaperPlot and subprocedure LogDdLines.

ManualEntry procedure Line 692

Purpose: to obtain (from the user) and verify maximum and minimum values for the graph when the user has expressed a wish to not accept those produced automatically by the program.

It is quite possible that the user will only want to change one of the four values which define the boundaries of the graph. For this reason it is made as easy as possible to leave any automatically selected values as they are. Any new values for graph limits are entered in lines 698 to 707.

The remainder of the procedure checks the validity of the values as they stand after the users entries. These checks are self explanatory.

Called by: procedure PrepForGraph.

Calls: function ReadReal of file FIRST.SEG

OpenGraphFile procedure Line 234

Purpose: to accept a file name from the user, check for the existence of a file by that name, and open the file in preparation to placing in it the plotter instructions for drawing a graph.

The section of code from line 241 to 251 will repeat until either the user enters a file name for which DOS cannot find an existing file, or until the user instructs the program to rewrite the existing file.

The section of code from line 253 to 256 stores the data in a disk file temporarily. The data will later be read back from this file and sent to the plotter if the user has asked for a plotter graph.

Called by: procedure DiskGraph.

Calls: function CapOptions of file FIRST.SEG, and function FileExist

PlotData procedure Line 858

Purpose: to control the plotting of time/drawdown values on a screen graph.

This procedure has two modes; the first produces isolated plots, and the second produces plots connected by a line. When the procedure is drawing the connecting line and it is to produce a plot, it must be able to decide whether to simply move to the point and plot, (as is required for the first

plot) or to draw a line from the previous point, and then plot. Boolean variable Started keeps track of whether or not a sequence of plots that may be interconnected has been commenced. It is initially set to false in line 863.

Lines 866 to 869 test whether the current point will fall within the graph boundaries. No attempt at plotting is made if this set of tests is failed. The next group of lines, those from 872 to 875, cause the x coordinate for the current data point to be calculated by a call to the appropriate function. The y coordinate is obtained for either a log or linear scale by the code from lines 877 to 880.

The shape that is plotted on the screen is dependent upon which curve is involved. Procedure PlotShape, which is called from line 881, takes care of producing the required type of plot. Boolean variable Join is true only when the user has indicated a desire to have the plotted points joined by a line. If appropriate, the joining line is drawn by line 882. Line 883 records the current plotting coordinates for possible use in production of the next connecting line.

Called by: procedure ScreenGraph.

Calls: functions LinearX, LogX, RootX, LinearY, and RootY; and procedure PlotShape.

PlotLinDdRL procedure Line 891

Purpose: to draw and label linear drawdown reference lines on the y axis of the screen graph.

This procedure has quiet a bit in common with subprocedure LinDdLines of procedure PlotRefLines (for the plotter), but here no consideration need be given to pen travel distance resulting in a simpler procedure. TempVal is given the y screen coordinate of each line to be plotted starting with the top boundary of the graph in program line 893. The reference line is drawn by line 897. The labelling process is a little more complicated because text is not written to the screen according to x-y coordinates, but by line and column number. Starting from the top, each successive screen text line corresponds to an increase of eight in the y plotting coordinate. The approximately correct position for the drawdown line label is found by line 900, while the label itself is produced by one or other of the calls to Format2 in line 902 and 903. The label is written on the screen by line 904.

Line 908 labels the maximum graphable drawdown which is assumed to be positive, and lines 909 to 913 label the minimum graphable drawdown which may well be negative. The only effective difference in labelling a drawdown line

with a negative value is that an extra character is required to handle the minus sign.

Called by: procedure ScreenGraph.

Calls: functions LinearY and Format2.

PlotLinTimeRL procedure Line 917

Purpose: to draw and label linear time reference lines on the x axis of the screen graph.

This procedure is similar to PlotLinDdRL, described above. The most significant difference is in the labelling of the lines. Particular care must be taken with the labelling of the right hand side time reference line because if a character is placed in column 80 of line 25 the screen will scroll with disastrous effects on the graph. It should be recalled that function Format2 will produce a string longer than that called for when it is impossible to produce a valid representation of the number any other way. For example, if the greatest plotable time is 10000 minutes then Format2 will produce the string "10000" although line 930 calls for a string four characters long. If the greatest plotable time is greater than 90 000 000 minutes then scrolling will take place unless the program is recompiled with the 73 in line 930 reduced to cause the maximum time to be plotted further left. (This still might not work correctly because there will not be space to print such large numbers along the bottom of the graph.)

Called by: procedure ScreenGraph.

Calls: functions LinearX and Format2

PlotLogDdRL procedure Line 947

Purpose: to draw and label logarithmic drawdown reference lines on the y axis of the screen graph.

Called by: procedure ScreenGraph.

Calls: functions LogY, Format2, and ExpTen.

PlotLogTimeRL procedure Line 933

Purpose: to draw and label logarithmic time reference lines on the x axis of the screen graph.

Called by: procedure ScreenGraph.

Calls: functions LogX, Format2, and ExpTen.

PlotRefLines procedure Line 262

Purpose: to cause the plotter to plot the appropriate reference lines.

While this major procedure makes up 381 lines of the program, it is subdivided into a number of subprocedures. All of the subprocedures are described separately.

The main, or controlling, part of PlotRefLines begins at line 631. Boolean variable Tog is used in controlling the direction of the drawing of the reference lines as mentioned under the description of procedure LinDdLines, above. There are only two types of drawdown scales, linear and logarithmic. The appropriate subprocedure is called from line 633. There are three types of time reference lines and the choice among the subprocedures is made by the case statement from line 634 to 639.

Called by: procedure DiskGraph.

The main part of this procedure calls: procedures LogDdLines, LinDdLines, LinTimeLines, LogTimeLines, and RootTimeLines.

PlotRootTimeRL procedure Line 972

Purpose: to draw and label square root of time reference lines on the x axis of the screen graph.

Called by: procedure ScreenGraph.

Calls: functions RootX, Format2, and ExpTen.

PlotShape procedure Line 836

Purpose: to plot a particular distinctively shaped point on the screen, the shape used depending on the which curve is being plotted.

The data points plotted on the screen corresponding to the first curve are shaped like a small 'x', those of the second curve are a small solid rectangle, and those of the third curve are a small diamond shape. Turbo Pascal has a standard procedure which is capable of plotting a point on the screen at designated x and y coordinates, this procedure simply varies the x and/or y values (for a series of points) sent to that procedure to produce the desired shape surrounding the point corresponding to the correct time/-drawdown value. Obviously, if one wanted to one could very easily alter these shapes; or add new ones to allow the plotting of more than three curves on one screen. The upper limit of three has been chosen because that seems to be as many curves as can be plotted without visual confusion.

Called by: procedure PlotData.

PlotterGraph procedure Line 1159

Purpose: to control the production of a graph on a plotter.

As far as the program is concerned, producing a graph on a plotter is practically the same as sending the instructions required to produce such a graph to a disk file. For this reason this procedure calls procedure DiskGraph to produce the temporary file, TEMP.DAT, which contains all the plotter instructions. After DiskGraph has finished it's task this procedure reads back the instructions and sends them out to the device connected to the first parallel port. (It seems that Turbo Pascal has no instruction for outputing to the second parallel port, or to a serial port.)

Called by: the main part of program PLOTWTD.

Calls: procedure DiskGraph.

PaperPlot procedure Line 1039

Purpose: to send the instructions for plotting one time/drawdown curve, on the plotter, to a disk file.

The first curve is drawn in black ink, so the same pen that was used for producing the graph axes and reference lines is used in plotting the first set of data (for simplicity, as well as to allow the best possible results from monochrome photocopying). Therefore a new pen is selected only for curves after the first (line 1044). The plotter itself is programmed in such a way as to cause the pen carrier to return to the point at which it received a pen changing instruction after changing the pen. This can result in considerable unnecessary pen movement, so the pen is moved close to the pen storage area before issuing the pen change instruction (line 1046).

The name of the file from which the time/drawdown data came for producing each curve is printed along the top of the graph, unless there are too many to fit in. This is handled by the code from line 1049 to 1053.

If the program has been instructed to connect the plots with a line, then plainly it cannot produce a connecting line while only the position of the first plot is known. The Boolean variable Started is given the value false in line 1054 to record the fact that it is inappropriate (rather than simply not requested) to draw a connecting line at this time.

The actual plotting of the data points is done by the section of the procedure from line 1055 to 1090. All of the data within a time/drawdown data set are checked through for plotable values by the code from line 1055 to 1061. The x and y coordinates for each plotable value are calculated by the section from 1062 to 1079, with the function being used for the calculation depending on the type of graph. Lines 1083 and 1084 either cause

the plotter to draw a line from the current pen position to the newly calculated coordinates and plot a point, or to move to the new position with the pen up and plot a point. (The plotter instruction 'd' means 'draw', 'm' means 'move', and 'n5' means 'draw symbol number 5' which is a small 'x' shape.

Called by: procedure DiskGraph.

Calls: functions LinearXP, LinearYP, LogXP, LogYP, and RootTimeXP.

PrepForGraph procedure Line 1011

Purpose: to inform the user of the limits of the data, obtain from the user the required graph type and limits, and do other work preparatory to producing a graph either on the screen or on the plotter.

Line 1014 calls procedure GetGraphType which finds out from the user which type of graph he requires. Procedure Remove0, which is called from line 1016, will remove those values which are not only unplotable, but would cause an error on any attempt to calculate their plotting coordinates (see below).

When data is to be plotted on a log of time scale then the logarithm of every time to be used is calculated and placed in a set of vectors, one for each curve to be plotted. If necessary other sets of vectors may be filled for log of drawdown or square root of time. This work is controlled by lines 1017 to 1022.

The finding of the maximums and minimums of the data, and informing the user of those values is done by lines 1023 to 1028. If the user expresses a desire to use other values for the limits of the graph then procedure ManualEntry is called to allow entry of the user's required graph limits.

If the graph is to be drawn on the screen then line 1036 calls procedure GraphMaxMin to have the exact values calculated for the graph limits, and for the reference line steps. If the graph is to be drawn on paper then this procedure will be called from a different point in the program.

Called by: procedures ScreenGraph and DiskGraph.

Calls: procedures GetGraphType, Remove0, GetMaxMin, and GraphMaxMin; and function Response of file FIRST.SEG.

Remove0 procedure Line 668

Purpose: to remove from the time and drawdown vectors, values which would cause errors if an attempt was made to calculate their logarithms or square roots, as appropriate for the graph type to be produced.

Values that must be removed from the data include: drawdowns not greater than zero if the graph is double logarithmic type, times not greater than zero if the graph is of double logarithmic or semilogarithmic type, times less than zero if the graph is of any type other than double linear. The rather complex 'if' statement of lines 675 to 679 looks for any such data, and most of the remainder of the procedure has the job of removing it.

The removal process involves overwriting the offending datum with the value immediately following in the same vector, then refilling this element with the value in the following element, etc. right up to the end of the vector. To retain the time/drawdown/discharge rate relationship across the vectors, each must be treated in the same way. eg. If a drawdown value of less than zero has been discovered in a set of data to be plotted on a double logarithmic graph, then not only must the drawdown value be removed, but also the associated time and discharge values so that the next time/drawdown/discharge rate sets are not disrupted.

Called by: procedure PrepForGraph.

RootTimeLines subprocedure Line 582

Purpose: to produce and label drawdown reference lines on the x scale of a square root of time graph on a plotter.

This subprocedure is very similar to subprocedure LinDdLines which is described above.

Called by: the main part of procedure PlotRefLines.

Calls: functions Format2 and RootXP.

RootX function Line 826

Purpose: to calculate the plotting coordinate on the square root x scale on the VDU screen for a given time.

Very similar to function LinearX described above.

Called by: procedures PlotData, PlotRootTimeRL, and PaperPlot.

RootXP function Line 225

Purpose: to calculate the plotting coordinate on the square root x scale on the plotter for a given time.

This function is very similar to function LinearXP described above.

Called by: procedure PaperPlot and subprocedure RootTimeLines.

ScreenGraph procedure Line 1111

Purpose: to control the production of a graph on the VDU screen.

If the flag SameFiles is set then it means that the same files as have already been graphed are to be graphed again, although perhaps using a different part of the data, or on a different type of graph. Consequently, if this flag is not set, then it is necessary to to call procedure GetSetOf-Files to have the user direct which files he wants loaded and graphed.

The call to procedure PrepForGraph causes the graphics mode to be entered and the graphing specifications to be established. Lines 1117 to 1122 cause the data to be plotted, one curve at a time with pauses at the users discretion in between, until all curves have been plotted. Finally, lines 1125 to 1130 call up the appropriate reference line procedures.

Called by: the main part of program PLOTWTD.

Calls: procedures GetSetOfFiles, PrepForGraph, PlotData, PlotLinDdRL, PlotLinTimeRL, PlotLogTimeRL, PlotLogDdRL, and PlotRootTimeRL.

7. KEY LINES OF PROGRAM PLOTWTD

```
  6 {#}{$I FIRST.SEG}
  8 {#}{$I READ.PRC}
 46 Function FileExist {Test for the existence of a given file}
 58 Function Log {Log to base ten}
 64 Function ExpTen {Antilog to base ten}
 70 Function Format2 {Produces a non exponential string form of a given
    number,
 90 Procedure GraphMaxMin; {Calculate maximums and minimums for both scales
 98 {# Calculate maximums and minimums for the drawdown scale of the graph,
113     {#----- Calculate the step for the drawdown ref. lines -----#}
125     {#----- Calculate the minimum drawdown for the graph -----#}
141  #----- Calculate maximums and minimums for the time scale -----#
196 {#----- Calculate plotting coordinates for paper plot -----#}
197 Function LogXP {Produces an X value for plotting, given log of time}
204 Function LogYP {Produces Y value for plotting, given log of drawdown}
211 Function LinearXP {Produces X value for plotting, given a time}
218 Function LinearYP {Produces a Y values for plotting, given a drawdown}
225 Function RootXP {Calculates X value for plotting on a root time scale,
232 {#------ End of calculation of plotting coordinates -----#}
234 Procedure OpenGraphFile; {Name a file, and open it
261 {#----- Beginning of major procedure PlotRefLines (on paper) -----#}
262 Procedure PlotRefLines; {Produce and file ref. line instructions.
269   procedure GetDdMaxMin; {Get drawdown limits standard scale log-log
    graph}
298   procedure GetTimeMaxMin; {Get time limits standard scale log-log
    graph}
327   procedure LogDdLines; {Plotter ref. lines, log drawdown scale}
403   procedure LinDdLines; {Plotter ref. lines, linear drawdown scale}
464   procedure LinTimeLines; {Plotter linear time reference lines}
513   procedure LogTimeLines; {Plotter ref. lines, log of time scale}
582   procedure RootTimeLines; {Plotter square root of time reference lines}
```

```
 631 begin {# main part of procedure PlotRefLines}
 641 {#----- End of major procedure PlotRefLines (on paper) -----#}
 643 Procedure GetGraphType; {What type of graph does user want?  ie. scales}
 668 Procedure RemoveO  {Remove negative or zero times}
 692 Procedure ManualEntry; {Manual entry of graph limits, if required}
 746 Procedure GetMaxMin; {Get maximums and minimums for all the curves}
 770 Procedure LoadTimeLog; {Load log of time vector}
 779 Procedure LoadDdLog; {Load log of drawdown vector}
 788 Procedure LoadRootTime; {Load root of time vector}
 797 {#----- Calculate screen plotting coordinates -----#}
 798 Function LogX {Plotting coord., log X scale, screen graph}
 805 Function LogY {Plotting coord., log Y scale, screen graph}
 812 Function LinearX {Plotting coord., linear X scale, screen graph}
 819 Function LinearY {Plotting coord., linear Y scale, screen graph}
 826 Function RootX {Plotting coord., root of time scale, screen graph}
 834 {#----- End of calculation of plotting coordinates -----#}
 836 Procedure PlotShape {Plot a different shape for each curve on screen}
 858 Procedure PlotData; {Control of data plotting on the screen}
 890 {#----- Reference lines on the screen -----#}
 891 Procedure PlotLinDdRL; {Linear drawdown ref. line on screen}
 917 Procedure PlotLinTimeRL; {Linear time ref. line on screen}
 933 Procedure PlotLogTimeRL; {Log of time ref. line on screen}
 947 Procedure PlotLogDdRL; {Log of drawdown ref. line on screen}
 972 Procedure PlotRootTimeRL; {Square root of time ref. line on screen}
 987 {#----- End of reference lines on the screen -----#}
 989 Procedure GetAFile; {Control loading of discharge test data file}
1011 Procedure PrepForGraph; {Prepare for producing a graph}
1039 Procedure PaperPlot; {Paper plotting instructions for one curve to disk
     file}
1093 Procedure GetSetOfFiles; {Control the loading of a set of data files}
1111 Procedure ScreenGraph; {Control production of a screen graph}
1135 Procedure DiskGraph; {Control production of a disk file graph}
1159 Procedure PlotterGraph; {Control production of a graph on a plotter}
1173 {#----- End of procedures and functions -----#}
1175 begin {# main part of program}
1218 end. {#}
```

8. KEY LINES OF PROGRAM PLOTWTD2

```
   7 {#}{$I FIRST.SEG}
   9 {#}{$I READ.PRC}
  44 Function FileExist {Test for the existence of a given file}
  56 Function Log {Log to base ten}
  62 Function ExpTen {Antilog to base ten}
  68 Function Format2 {Produces a non exponential string form of a given
     number,
  88 Procedure GraphMaxMin; {Calculate maximums and minimums for both scales
  96 {# Calculate maximums and minimums for the drawdown scale of the graph,
 111     {#----- Calculate the step for the drawdown ref. lines -----#}
 123     {#----- Calculate the minimum drawdown for the graph -----#}
 139  #----- Calculate maximums and minimums for the time scale -----#
 194 {#----- Calculate plotting coordinates for paper plot -----#}
 195 Function LogXP {Produces an X value for plotting, given log of time}
 202 Function LogYP {Produces Y value for plotting, given log of drawdown}
 209 Function LinearXP {Produces X value for plotting, given a time}
 216 Function LinearYP {Produces a Y values for plotting, given a drawdown}
 223 Function RootXP {Calculates X value for plotting on a root time scale,
 230 {#------ End of calculation of plotting coordinates -----#}
 232 Procedure OpenGraphFile; {Name a file, and open it
```

```
 251 {#----- Beginning of major procedure PlotRefLines (on paper) -----#}
 252 Procedure PlotRefLines; {Produce and file ref. line instructions.
 259   procedure LogDdLines; {Ref. lines, log drawdown scale}
 340   procedure LinDdLines; {Ref. lines, linear drawdown scale}
 405   procedure LinTimeLines; {Linear time reference lines}
 454   procedure LogTimeLines; {Ref. lines, log of time scale}
 520   procedure RootTimeLines; {Draw square root of time reference lines}
 569 begin {# main part of procedure PlotRefLines}
 579 {#----- End of major procedure PlotRefLines -----#}
 581 Procedure GetGraphType; {What type of graph does user want?  ie. scales}
 593 Procedure Remove0  {Remove negative or zero times}
 617 Procedure ManualEntry; {Manual entry of graph limits, if required}
 671 Procedure GetMaxMin; {Get maximums and minimums for all the curves}
 695 Procedure LoadTimeLog; {Load log of time vector}
 704 Procedure LoadDdLog; {Load log of drawdown vector}
 713 Procedure LoadRootTime; {Load root of time vector}
 722 {#----- Calculate screen plotting coordinates -----#}
 723 Function LogX {Plotting coord., log X scale, screen graph}
 730 Function LogY {Plotting coord., log Y scale, screen graph}
 737 Function LinearX {Plotting coord., linear X scale, screen graph}
 744 Function LinearY {Plotting coord., linear Y scale, screen graph}
 751 Function RootX {Plotting coord., root of time scale, screen graph}
 759 {#----- End of calculation of plotting coordinates -----#}
 761 Procedure PlotShape {Plot a different shape for each curve on screen}
 783 Procedure PlotData; {Control of data plotting on the screen}
 814 {#----- Reference lines on the screen -----#}
 815 Procedure PlotLinDdRL; {Linear drawdown ref. line on screen}
 841 Procedure PlotLinTimeRL; {Linear time ref. line on screen}
 857 Procedure PlotLogTimeRL; {Log of time ref. line on screen}
 871 Procedure PlotLogDdRL; {Log of drawdown ref. line on screen}
 896 Procedure PlotRootTimeRL; {Square root of time ref. line on screen}
 912 Procedure GetAFile; {Control loading of discharge test data file}
 934 Procedure PrepForGraph; {Prepare for producing a graph}
 962 Procedure PlotPoint {Plot one point on the graph}
 970 Procedure PaperPlot; {Paper plotting instructions for one curve to disk
     file}
1027 Procedure GetSetOfFiles;
1045 Procedure ScreenGraph; {Control production of a screen graph}
1069 Procedure DiskGraph; {Control production of a disk file graph}
1089 {#----- End of procedures and functions -----#}
1090 begin {# main part of program}
1132 end. {#}
```

9. LISTING OF PROGRAM PLOTWTD (ROLAND VERSION)

```
 1 Program PLOTWTD_PAS;
 2 {To graph a set of discharge test data on screen or a Roland type
   plotter}
 3
 4 {$R+}
 5
 6 {#}{$I FIRST.SEG}
 7
 8 {#}{$I READ.PRC}
 9
10 type
11   GraphTypes=(Linear, SemiLog, LogLog, RootTime);
12   PapSizes=(Large, Small);
13   Devices=(Plotter, Screen, Disk);
14   VecRec=record
15     TimeVec, LogTimeVec, RootTimeVec, DdVec,
16     LogDdVec, RateVec: MainVec;
17   end;
18
19 var
20   Valid, Again, SameFiles, Join, FullLines: Boolean;
21   NumOfCurves, CurveNum: byte;
22   Ch: Char;
23   GenInt, PlotR, PlotB, Result, X, Y, X1: integer;
24   NumData: array[1..10] of integer;
25   MaxDd, MinDd, MaxTime, MinTime, LogDdRange, LogTimeRange: real;
26   DdStep, MaxGraphDd, MinGraphDd, MinLogTime: real;
27   MaxRootTime, MinRootTime, MaxGraphTime, MinGraphTime: real;
28   MaxLogDd, MinLogDd, LinTimeStep, RootTimeStep: real;
29   MinLogGDd, MaxLogGDd, MinLogGTime, MaxLogGTime: real;
30   Distance: array[1..10] of real;
31   TestFile: array[1..10] of MedString;
32   Device: Devices;
33   GraphType: GraphTypes; PapSize: PapSizes;
34   TestType: array[1..10] of Test; WellType: array[1..10] of Well;
35   VRP: array[1..10] of ^VecRec;
36   OutFileName: MedString; GF: text;
37
38 const
39   PlotLt=220; PlotRL=3620; PlotRS=2360;   {Lt=Left, R=Right, T=Top,
   B=Bottom}
40   PlotBL=140; PlotBS=960; PlotT=2520;     {S=Small paper, L=Large}
41   {With standard log-log scale, top-bottom on large paper is 2.377
   cycles
42    (ie. 1:240), and left-right is 3.420 cycles (ie. 1:2630).}
43   Left=48; right=639; Top=0; Bottom=185;
44   LnTen=2.302585093; MaxFiles=7;
45
46 Function FileExist {Test for the existence of a given file}
47 (FileName: ShortString): boolean;
48 var ThisFile: File of byte;
49 begin
50   Assign(ThisFile,FileName);
51   {$I-}
52   Reset(ThisFile);
53   {$I+}
54   FileExist:=(IOResult=0);
```

```
  close(ThisFile);
end; {Function FileExist}

Function Log {Log to base ten}
(Num: real): real;
begin
  Log:=Ln(Num)/LnTen;
end; {Function Log}

Function ExpTen {Antilog to base ten}
(Num: real): real;
begin
  ExpTen:=Exp(Num*LnTen);
end; {Function ExpTen}

Function Format2 {Produces a string form of a given number}
  (Num: real; Leng: integer): ShortString;
var
  Sign: Boolean;  NumPlaces: integer;  TString: ShortString;
begin
  Sign:=Num>0; if not Sign then begin Num:=-Num; Leng:=Leng-1 end;
  str(Num:20:Leng-2,TString);
  while copy(TString,1,1)=' '
    do TString:=Copy(TString,2,Length(TString)-1);
  if (Length(TString)>Leng)
  then if (Log(Num)*1.00001<Leng)
  then begin
    NumPlaces:=Leng-trunc(Log(Num))-2;
    if NumPlaces<0 then NumPlaces:=0;
    str(Num:Leng:NumPlaces,TString)
  end {then}
  else TString:=Copy(TString,1,Pos('.',TString)-1);
  if not Sign then TString:='-'+TString; Format2:=TString;
end; {Function format2}

Procedure GraphMaxMin; {Calculate maximums and minimums for both scales
of the graph, and the step for non log ref. lines.}
var
  I: integer;
  Multiplier, TempTime: real;
const
  GraphStep: array[1..3] of real =(1,2,5);
  GraphRef: array[1..12] of real
  =(1.,1.2,1.5,2.,2.5,3.,4.,5.,6.,7.,8.,9.0);
{# Calculate maximums and minumums for the drawdown scale of the graph,
 and calculate the drawdown step for the graph's ref. lines}
begin
  Multiplier:=0.001;      {Calculate the maximum drawdown for the graph}
  if GraphType<>LogLog then
  begin
    MaxGraphDd:=GraphRef[1]*Multiplier; I:=1;
    while MaxGraphDd<MaxDd do
    begin
      I:=I+1; if I=11
      then begin
        Multiplier:=Multiplier*10; I:=1;
```

```
110       end;
111       MaxGraphDd:=GraphRef[I]*Multiplier;
112     end;
113     {#----- Calculate the step for the drawdown ref. lines -----#}
114     Multiplier:=0.0001; DdStep:=GraphStep[1]*Multiplier; I:=1;
115     while (DdStep*8)<(MaxGraphDd-MinDd) do
116     begin
117       if I<3 then I:=I+1
118       else begin I:=1; Multiplier:=Multiplier*10 end;
119       DdStep:=GraphStep[I]*Multiplier
120     end;
121     if (DdStep*8)<(MaxGraphDd) then
122       if I<3
123         then DdStep:=GraphStep[I+1]*Multiplier
124         else DdStep:=GraphStep[1]*Multiplier*10;
125     {#----- Calculate the minimum drawdown for the graph -----#}
126     MinGraphDd:=MaxGraphDd-15*DdStep;
127     while MinGraphDd+DdStep<=MinDd do MinGraphDd:=MinGraphDd+DdStep;
128     if (MinGraphDd<0) and (MinDd>=0) then MinGraphDd:=0;
129   end             {if GraphType<>LogLog}
130   else begin      {if GraphType=LogLog}
131     MaxLogGDd:=Int(Log(MaxDd)+0.99);
132     if MinDd>0.001
133     then begin
134       TempVal:=Log(MinDd); if TempVal<0 then TempVal:=TempVal-1;
135       MinLogGDd:=Int(TempVal);
136     end
137     else MinLogGDd:=-3;
138     MinGraphDd:=ExpTen(MinLogGDd); MaxGraphDd:=ExpTen(MaxLogGDd);
139   end; {if Graph=LogLog}
140  {End of first section:
141  #----- Calculate maximums and minimums for the time scale -----#
142   of the graph, and calculate the time step for the graphs ref lines}
143   if (GraphType=Linear) or (GraphType=RootTime)
144   then begin
145     I:=1; Multiplier:=0.01; MaxGraphTime:=GraphRef[1]*Multiplier;
146     while MaxGraphTime<MaxTime do
147     begin
148       if I<10 then begin
149         I:=I+1; MaxGraphTime:=GraphRef[I]*Multiplier;
150       end
151       else begin
152         Multiplier:=Multiplier*10; I:=1;
153         MaxGraphTime:=GraphRef[I]*Multiplier;
154       end;
155     end; {while}
156     I:=1; Multiplier:=0.01; TempTime:=GraphRef[1]*Multiplier;
    MinGraphTime:=MinTime;
157     while TempTime<=MinTime do
158     begin
159       MinGraphTime:=TempTime;
160       if I<10 then begin
161         I:=I+1; TempTime:=GraphRef[I]*Multiplier;
162       end
163       else begin
164         Multiplier:=Multiplier*10; I:=1;
165         TempTime:=GraphRef[I]*Multiplier;
166       end;
167     end; {while}
168     if MinGraphTime<MaxGraphTime/6 then MinGraphTime:=0;
```

```
169     if GraphType=RootTime
170     then begin
171       if MinGraphTime<0.25*MaxGraphTime then MinGraphTime:=0;
172       if (Device=Screen)
173         then RootTimeStep:=sqrt(MaxGraphTime-MinGraphTime)/5
174         else RootTimeStep:=sqrt(MaxGraphTime-MinGraphTime)/10;
175       MaxRootTime:=sqrt(MaxGraphTime);
176       MinRootTime:=sqrt(MinGraphTime);
177     end; {if GraphType=RootTime}
178     if GraphType=Linear
179     then begin
180       if MinGraphTime<0.25*MaxGraphTime then MinGraphTime:=0;
181       if (Device=Screen)
182        then LinTimeStep:=(MaxGraphTime-MingraphTime)/5
183        else LinTimeStep:=(MaxGraphTime-MingraphTime)/10;
184     end {if GraphType=Linear}
185   end {if, or, then}
186   else begin {Graph semilog or log}
187     If MinTime<=0.1 then MinTime:=0.1;
188     MaxLogGTime:=Int(Log(MaxTime)+0.99);
189     MaxGraphTime:=ExpTen(MaxLogGTime);
190     MinLogGTime:=Int(Log(MinTime*1.00001));
191     if MinTime<0.99999 then MinLogGTime:=MinLogGTime-1;
192     MinGraphTime:=ExpTen(MinLogGTime);
193   end; {else}
194 end; {Procedure GraphMaxMin}
195
196 {#----- Calculate plotting coordinates for paper plot -----#}
197 Function LogXP {Produces an X value for plotting, given log of time}
198 (Num: real): integer;
199 begin
200   LogXP:=round(PlotLt+(PlotR-PlotLt)*
201     (Num-MinLogGTime)/(MaxLogGTime-MinLogGTime));
202 end; {Function LogXP}
203
204 Function LogYP {Produces Y value for plotting, given log of drawdown}
205 (Num: real): integer;
206 begin
207   LogYP:=round(PlotB+(PlotT-PlotB)*
208     (Num-MinLogGDd)/(MaxLogGDd-MinLogGDd));
209 end; {Function LogYP}
210
211 Function LinearXP {Produces X value for plotting, given a time}
212 (Num: real): integer;
213 begin
214   LinearXP:=round(PlotLt+(PlotR-PlotLt)*(Num-MinGraphTime)/
215     (MaxGraphTime-MinGraphTime));
216 end; {Function LinearXP}
217
218 Function LinearYP {Produces a Y values for plotting, given a drawdown}
219 (Num: real): integer;
220 begin
221   LinearYP:=round(PlotT+(PlotB-PlotT)*(Num-MinGraphDd)/
222     (MaxGraphDd-MinGraphDd));
223 end; {Function LinearYP}
224
225 Function RootXP {Calculates X value for plotting on a root time scale,
226   from a given root of time value.}
227 (Num: real): integer;
```

```
228 begin
229   RootXP:=round(PlotLt+(PlotR-PlotLt)*
230     (Num-MinRootTime)/(MaxRootTime-MinRootTime));
231 end; {Function RootXP}
232 {#------ End of calculation of plotting coordinates -----#}
233
234 Procedure OpenGraphFile; {Name a file, and open it
235   ready to receive plotter instructions}
236 var
237   Go: boolean;
238 begin
239   if Device=Disk
240   then begin
241     repeat
242       Go:=true;
243       write('Please enter the full file name for the output data ');
244       readln(OutFileName);
245       if FileExist(OutFileName) then
246       begin
247         write('There is a file by that name, ');
248         if CapOptions('Rewrite it, or Change file name? ')=1
249         then Go:=true else Go:=false;
250       end;
251     until Go;
252   end
253   else begin
254     writeln('The data will be saved to a temporary file ',
255       'called ''TEMP.DAT''.');
256     OutFileName:='TEMP.DAT';
257   end; {if-then-else}
258   Assign(GF,OutFileName); Rewrite(GF);
259 end; {Procedure OpenGraphFile}
260
261 {#----- Beginning of major procedure PlotRefLines (on paper) -----#}
262 Procedure PlotRefLines; {Produce and file ref. line instructions.
263 Send all reference line drawing instructions to a disk file.
264 The type of graph has been decided previously}
265 var
266   Tog: boolean;
267   I, J: integer;
268
269   procedure GetDdMaxMin; {Get drawdown limits standard scale log-log
    graph}
270   var
271     Valid: boolean;
272     TempMax, Temp: real;
273   begin
274     TempMax:=MinLogGDd+LogDdRange;
275     if TempMax<Log(MaxDd) then
276     begin
277       writeln;
278       writeln('   There is insufficient room on the drawdown scale for '
279       ,'the whole of');
280       writeln('the data.  Please enter the minimum drawdown that you '
281       ,'require for');
282       writeln('this graph.  It must be a power of 10, from 0.0001 to '
283       ,'1 (metres)');
284       write('(Note that max. drawdown will be 316 times min.
    drawdown.) ');
```

```
285       repeat
286         Temp:=ReadReal(1);
287         if ((Temp<>0.0001) and (Temp<>0.001) and (Temp<>0.01)
288         and (Temp<>0.1) and (Temp<>1))
289         or (Temp>=MaxDd) then Valid:=false else Valid:=true;
290         if not Valid then writeln('Invalid entry!');
291       until Valid;
292       MinLogDd:=Log(Temp); MinLogGDd:=MinLogDd; MinDd:=ExpTen(MinLogDd);
293       MaxLogGDd:=MinLogGDd+LogDdRange;
294     end
295     else MaxLogGDd:=TempMax;
296   end; {sub procedure GetDdMaxMin}
297
298   procedure GetTimeMaxMin; {Get time limits standard scale log-log
    graph}
299   var
300     Valid: boolean;
301     TempMax, Temp: real;
302   begin
303     TempMax:=MinLogGTime+LogTimeRange;
304     if TempMax<Log(MaxTime) then
305     begin
306       writeln;
307       writeln('   There is insufficient room on the time scale for '
308       ,'the whole of');
309       writeln('the data.  Please enter the minimum time that you '
310       ,'require for');
311       writeln('this graph.  It must be a power of 10, from 0.01 to '
312       ,'1000 (minutes)');
313       write('(Note that max. time will be 3460 times min. time.) ');
314       repeat
315         Temp:=ReadReal(1);
316         if ((Temp<>0.01) and (Temp<>0.1) and (Temp<>1)
317         and (Temp<>10) and (Temp<>100) and (Temp<>1000))
318         or (Temp>=MaxTime) then Valid:=false else Valid:=true;
319         if not Valid then writeln('Invalid entry!');
320       until Valid;
321       MinLogTime:=Log(Temp); MinLogGTime:=MinLogTime;
322       MinTime:=ExpTen(MinLogTime);
    MaxLogGTime:=MinLogGTime+LogTimeRange;
323     end
324     else MaxLogGTime:=TempMax;
325   end; {sub procedure GetTimeMaxMin}
326
327   procedure LogDdLines; {Plotter ref. lines, log drawdown scale}
328   var
329     J, RefNum, IntLog: integer;
330     Drawdown: real;
331   begin
332     if (PapSize=Large) and (GraphType=LogLog) then GetDdMaxMin;
333     Drawdown:=ExpTen(MinLogGDd); writeln(GF,'j1');
334     RefNum:=1; IntLog:=round(MinLogGDd);
335     if FullLines
336     then begin
337       J:=PlotB;
338       while J<=PlotT do
339       begin
340         if Tog
341         then begin
```

```
342           if (RefNum=1) or (RefNum=3) then
343             if Drawdown>=0.01
344             then writeln(GF,'m',PlotLt-130,', ',J,' p',
    Format2(Drawdown,4))
345             else writeln(GF,'m',PlotLt-130,', ',J,' p',
    Format2(Drawdown,5));
346           writeln(GF,'m',PlotLt,', ',J,' d',PlotR,', ',J);
347         end
348         else begin
349           writeln(GF,'m',PlotR,', ',J,' d',PlotLt,',',J);
350           if (RefNum=1) or (RefNum=3) then
351             if Drawdown>=0.01
352             then writeln(GF,'m',PlotLt-130,', ',J,' p',
    Format2(Drawdown,4))
353             else writeln(GF,'m',PlotLt-130,', ',J,' p',
    Format2(Drawdown,5));
354         end; {if then else Tog}
355         if RefNum=9
356         then begin RefNum:=1; IntLog:=IntLog+1; end
357         else RefNum:=RefNum+1;
358         Drawdown:=ExpTen(IntLog)*RefNum;
359         if Tog then Tog:=false else Tog:=true;
360         J:=LogYP(Log(Drawdown));
361       end; {while J}
362       if PapSize=Large then
363         writeln(GF,'m',PlotR,',',PlotT,' d',PlotLt,',',PlotT);
364     end    {if FullLines}
365     else begin {Implied ref. lines}
366       writeln(GF,'m',PlotLt,',',PlotT,' d',PlotR,',',PlotT,
367       ' d',PlotR,',',PlotB,' d',PlotLt,',',PlotB,' d',PlotLt,',',PlotT);
368       J:=PlotB;
369       while J<=PlotT do
370       begin
371         if (RefNum=4) or (RefNum=1) then
372           writeln(GF,'m',PlotLt-130,',',J,' p',Format2(Drawdown,4));
373         if (J<>PlotB) and (J<>PlotT) and (RefNum=1) then
374           writeln(GF,'m',PlotLt,', ',J,' d',PlotLt+20,', ',J);
375         if (J<>PlotB) and (J<>PlotT) and (RefNum<>1) then
376           writeln(GF,'m',PlotLt,', ',J,' d',PlotLt+10,', ',J);
377         if RefNum=9
378         then begin RefNum:=1; IntLog:=IntLog+1; end
379         else RefNum:=RefNum+1;
380         Drawdown:=ExpTen(IntLog)*RefNum;
381         J:=LogYP(Log(Drawdown));
382       end; {while}
383       Drawdown:=ExpTen(MinLogGDd);
384       RefNum:=1; IntLog:=round(MinLogGDd); J:=PlotB;
385       while J<=PlotT do
386       begin
387         if (J<>PlotB) and (J<>PlotT) and (RefNum=1) then
388           writeln(GF,'m',PlotR,', ',J,' d',PlotR-20,', ',J);
389         if (J<>PlotB) and (J<>PlotT) and (RefNum<>1) then
390           writeln(GF,'m',PlotR,', ',J,' d',PlotR-10,', ',J);
391         if RefNum=9
392         then begin RefNum:=1; IntLog:=IntLog+1; end
393         else RefNum:=RefNum+1;
394         Drawdown:=ExpTen(IntLog)*RefNum;
395         J:=LogYP(Log(Drawdown));
396       end; {while}
```

```
397     end; {if then else FullLines}
398     writeln(GF,'m',PlotLt-160,',',(PlotT+PlotB) div 2-130,', q1');
399     writeln(GF,'pDrawdown (m)');
400     writeln(GF,'q0');
401   end; {sub procedure LogDdLines}
402
403   procedure LinDdLines; {Plotter ref. lines, linear drawdown scale}
404   var
405     J: integer;
406     Drawdown: real;
407   begin
408     Drawdown:=MinGraphDd; writeln(GF,'j1');
409     if FullLines
410     then begin
411       J:=PlotT;
412       while J>=PlotB do
413       begin
414         if Tog
415         then begin
416           writeln(GF,'m',PlotLt-130,', ',J,' p',Format2(Drawdown,4));
417           writeln(GF,'m',PlotLt,', ',J,' d',PlotR,', ',J);
418         end
419         else begin
420           writeln(GF,'m',PlotR,', ',J,' d',PlotLt,',',J);
421           writeln(GF,'m',PlotLt-130,', ',J,' p',Format2(Drawdown,4));
422         end; {if then else Tog}
423         Drawdown:=Drawdown+DdStep;
424         if Tog then Tog:=false else Tog:=true;
425         J:=LinearYP(Drawdown);
426       end; {while J}
427       if Drawdown<MaxGraphDd+DdStep*0.6 then
428       begin
429         writeln(GF,'m',PlotR,',',PlotB,' d',PlotLt,',',PlotB);
430         writeln(GF,'m',PlotLt-130,', ',LinearYP(MaxGraphDd),' p',
431           Format2(MaxGraphDd,4))
432       end;
433     end    {if FullLines}
434     else begin {Implied ref. lines}
435       writeln(GF,'m',PlotLt,',',PlotT,' d',PlotR,',',PlotT,
436       ' d',PlotR,',',PlotB,' d',PlotLt,',',PlotB,' d',PlotLt,',',PlotT);
437       J:=PlotT;
438       while J>=PlotB do
439       begin
440         writeln(GF,'m',PlotLt-130,',',J,' p',Format2(Drawdown,4));
441         if (J<>PlotB) and (J<>PlotT) then
442           writeln(GF,'m',PlotLt,',',J,' d',PlotLt+20,',',J);
443         Drawdown:=Drawdown+DdStep;
444         J:=LinearYP(Drawdown);
445       end; {while}
446       if Drawdown<MaxGraphDd+DdStep*0.6 then
447         writeln(GF,'m',PlotLt-130,', ',LinearYP(MaxGraphDd),' p',
448         Format2(MaxGraphDd,4));
449       Drawdown:=MinGraphDd;
450       J:=PlotT;
451       while J>=PlotB do
452       begin
453         if (J<>PlotB) and (J<>PlotT) then
454           writeln(GF,'m',PlotR,', ',J,' d',PlotR-20,', ',J);
455         Drawdown:=Drawdown+DdStep;
```

```
456         J:=LinearYP(Drawdown);
457       end; {while}
458     end; {if then else FullLines}
459     writeln(GF,'m',PlotLt-160,',',(PlotT+PlotB) div 2-130,', q1');
460     writeln(GF,'pDrawdown (m)');
461     writeln(GF,'q0');
462   end; {sub procedure LinDdLines}
463
464   procedure LinTimeLines; {Plotter linear time reference lines}
465   var
466     J: integer;
467     Time: real;
468   begin
469     Time:=MinGraphTime; Tog:=true;
470     if FullLines
471     then begin
472       J:=PlotLt;
473       while J<=PlotR do
474       begin
475         if Tog
476         then begin
477           writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(Time,4));
478           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotT);
479         end
480         else begin
481           writeln(GF,'m',J,', ',PlotT,' d',J,', ',PlotB);
482           writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(Time,4));
483         end; {if then else Tog}
484         Time:=Time+LinTimeStep;
485         if Tog then Tog:=false else Tog:=true;
486         J:=LinearXP(Time);
487       end; {while J}
488     end    {if FullLines}
489     else begin {Implied ref. lines}
490       J:=PlotLt;
491       while J<=PlotR do
492       begin
493         writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(Time,4));
494         if (J<>PlotLt) and (J<>PlotR) then
495           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotB+20);
496         Time:=Time+LinTimeStep;
497         J:=LinearXP(Time);
498       end; {while}
499       Time:=MinGraphTime;
500       J:=PlotLt;
501       while J<=PlotR do
502       begin
503         if (J<>PlotLt) and (J<>PlotR) then
504           writeln(GF,'m',J,',',PlotT,' d',J,',',PlotT-20);
505         Time:=Time+LinTimeStep;
506         J:=LinearXP(Time);
507       end; {while}
508     end; {if then else FullLines}
509     writeln(GF,'m',(PlotR+PlotLt) div 2+PlotLt-320,',',PlotB-100);
510     writeln(GF,'pTime (min.)');
511   end; {sub procedure LinTimeLines}
512
513   procedure LogTimeLines; {Plotter ref. lines, log of time scale}
514   var
```

```
515     J, RefNum, IntLog: integer;
516     Time: real;
517   begin
518     if (PapSize=Large) and (GraphType=LogLog) then GetTimeMaxMin;
519     Time:=ExpTen(MinLogGTime); Tog:=true;
520     RefNum:=1; IntLog:=round(MinLogGTime);
521     if FullLines
522     then begin
523       J:=PlotLt;
524       while J<=PlotR do
525       begin
526         if Tog
527         then begin
528           if RefNum=1 then
529             writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(Time,4));
530           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotT);
531         end
532         else begin
533           writeln(GF,'m',J,', ',PlotT,' d',J,',',PlotB);
534           if RefNum=1 then
535             writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(Time,4));
536         end; {if then else Tog}
537         if RefNum=9
538         then begin RefNum:=1; IntLog:=IntLog+1; end
539         else RefNum:=RefNum+1;
540         Time:=ExpTen(IntLog)*RefNum;
541         if Tog then Tog:=false else Tog:=true;
542         J:=LogXP(Log(Time));
543       end; {while J}
544       if (PapSize=Large) and (GraphType=LogLog) then
545         writeln(GF,'m',PlotR,',',PlotT,' d',PlotR,',',PlotB);
546     end    {if FullLines}
547     else begin {Implied ref. lines}
548       J:=PlotLt;
549       while J<=PlotR do
550       begin
551         if RefNum=1 then
552           writeln(GF,'m',J-40,',',PlotB-50,' p',Format2(Time,4));
553         if (J<>PlotLt) and (J<>PlotR) and (RefNum=1) then
554           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotB+20);
555         if (J<>PlotLt) and (J<>PlotR) and (RefNum<>1) then
556           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotB+10);
557         if RefNum=9
558         then begin RefNum:=1; IntLog:=IntLog+1; end
559         else RefNum:=RefNum+1;
560         Time:=ExpTen(IntLog)*RefNum;
561         J:=LogXP(Log(Time));
562       end; {while}
563       Time:=ExpTen(MinLogGTime);
564       RefNum:=1; IntLog:=round(MinLogGTime); J:=PlotLt;
565       while J<=PlotR do
566       begin
567         if (J<>PlotLt) and (J<>PlotR) and (RefNum=1) then
568           writeln(GF,'m',J,', ',PlotT,' d',J,', ',PlotT-20);
569         if (J<>PlotLt) and (J<>PlotR) and (RefNum<>1) then
570           writeln(GF,'m',J,', ',PlotT,' d',J,', ',PlotT-10);
571         if RefNum=9
572         then begin RefNum:=1; IntLog:=IntLog+1; end
573         else RefNum:=RefNum+1;
```

```
574         Time:=ExpTen(IntLog)*RefNum;
575         J:=LogXP(Log(Time));
576       end; {while}
577     end; {if then else FullLines}
578     writeln(GF,'m',(PlotR+PlotLt) div 2+PlotLt-320,',',PlotB-100);
579     writeln(GF,'pTime (min.)');
580   end; {sub procedure LogTimeLines}
581
582   procedure RootTimeLines; {Plotter square root of time reference lines}
583   var
584     J: integer;
585     RTime: real;
586   begin
587     RTime:=MinRootTime; Tog:=true;
588     if FullLines
589     then begin
590       J:=PlotLt;
591       while J<=PlotR do
592       begin
593         if Tog
594         then begin
595           writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(sqr(RTime),4));
596           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotT);
597         end
598         else begin
599           writeln(GF,'m',J,', ',PlotT,' d',J,', ',PlotB);
600           writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(sqr(RTime),4));
601         end; {if then else Tog}
602         RTime:=RTime+RootTimeStep;
603         if Tog then Tog:=false else Tog:=true;
604         J:=RootXP(RTime);
605       end; {while J}
606     end    {if FullLines}
607     else begin {Implied ref. lines}
608       J:=PlotLt;
609       while J<=PlotR do
610       begin
611         writeln(GF,'m',J-40,', ',PlotB-50,' p',Format2(sqr(RTime),4));
612         if (J<>PlotLt) and (J<>PlotR) then
613           writeln(GF,'m',J,', ',PlotB,' d',J,', ',PlotB+20);
614         RTime:=RTime+RootTimeStep;
615         J:=RootXP(RTime);
616       end; {while}
617       RTime:=MinGraphTime;
618       J:=PlotLt;
619       while J<=PlotR do
620       begin
621         if (J<>PlotLt) and (J<>PlotR) then
622           writeln(GF,'m',J,',',PlotT,' d',J,',',PlotT-20);
623         RTime:=RTime+RootTimeStep;
624         J:=RootXP(RTime);
625       end; {while}
626     end; {if then else FullLines}
627     writeln(GF,'m',(PlotR+PlotLt) div 2+PlotLt-320,',',PlotB-100);
628     writeln(GF,'pTime (min.)');
629   end; {sub procedure RootTimeLines}
630
631 begin {# main part of procedure PlotRefLines}
632   Tog:=true;
```

```
633   if GraphType=LogLog then LogDdLines else LinDdLines;
634   Case GraphType of
635     Linear:   LinTimeLines;
636     SemiLog:  LogTimeLines;
637     LogLog:   LogTimeLines;
638     RootTime: RootTimeLines;
639   end; {of cases}
640 end; {Procedure PlotRefLines}
641 {#----- End of major procedure PlotRefLines (on paper) -----#}
642
643 Procedure GetGraphType; {What type of graph does user want?  ie. scales}
644 begin
645   write('What type of graph, ');
646   GraphType:=
647     GraphTypes(CapOptions('Linear, Semi-log, lOg-log, Root-time? ')-1);
648   if (GraphType=LogLog) and (Device<>Screen) then
649   begin
650     writeln;
651     writeln('  If you require a standard sized graph (76mm per log
    cycle)',
652       ' choose large');
653     writeln('paper.  The scale will be chosen to suit the graph if
    small',
654       ' paper is used.');
655   end;
656   if Device<>Screen then
657   begin
658     writeln('What paper size, large (A3=420x298mm) or small
    (A4=298x210mm)? ');
659     write('Press L or S ');
660     if Response('LS')='L'
661       then begin PapSize:=Large; PlotR:=PlotRL; PlotB:=PlotBL end
662       else begin PapSize:=Small; PlotR:=PlotRS; PlotB:=PlotBS end;
663     write('Do you want full reference lines? ');
664     if Response('YN')='Y' then FullLines:=true else FullLines:=false;
665   end; {if Device}
666 end; {Procedure GetGraphType}
667
668 Procedure Remove0  {Remove negative or zero times}
669 (GraphType: GraphTypes);
670 begin
671   for CurveNum:=1 to NumOfCurves do
672   begin
673     J:=1;
674     repeat
675       if (((GraphType=SemiLog) or (GraphType=LogLog)) and
676         (VRP[CurveNum]^.TimeVec[J]<=0)) or
677       ((GraphType=LogLog) and (VRP[CurveNum]^.DdVec[J]<=0)) or
678       ((GraphType=RootTime) and (VRP[CurveNum]^.TimeVec[J]<0))
679       then begin
680         for I:=J+1 to NumData[CurveNum] do
681         begin VRP[CurveNum]^.TimeVec[I-1]:=VRP[CurveNum]^.TimeVec[I];
682           VRP[CurveNum]^.DdVec[I-1]:=VRP[CurveNum]^.DdVec[I];
683           VRP[CurveNum]^.RateVec[I-1]:=VRP[CurveNum]^.RateVec[I];
684         end; {for I}
685         NumData[CurveNum]:=NumData[CurveNum]-1; J:=J-1;
686       end; {if}
687       J:=J+1;
688     until J>NumData[CurveNum];
```

```
689   end; {for CurveNum}
690 end; {Procedure Remove0}
691
692 Procedure ManualEntry; {Manual entry of graph limits, if required}
693 var Valid:boolean;
694 begin
695   writeln;
696   Valid:=true;
697   repeat
698     writeln('Type 0 and press <Enter> if you want to leave any');
699     writeln('of the below limits unchanged.'); writeln;
700     TempVal:=0; write('Lower drawdown limit? '); TempVal:=ReadReal(1);
701     if TempVal<>0 then MinDd:=TempVal;
702     TempVal:=0; write('Upper drawdown limit? '); TempVal:=ReadReal(1);
703     if TempVal<>0 then MaxDd:=TempVal;
704     TempVal:=0; write('Lower time limit? '); TempVal:=ReadReal(1);
705     if TempVal<>0 then MinTime:=TempVal;
706     TempVal:=0; write('Upper time limit? '); TempVal:=ReadReal(1);
707     if TempVal<>0 then MaxTime:=TempVal;
708     if MaxDd<-200 then
709     begin
710       Valid:=false;
711       writeln('Value of Maximum drawdown is too low!');
712     end;
713     if MaxDd<=MinDd then
714     begin
715       Valid:=false;
716       writeln('Max. drawdown is invalid in it''s relationship to min. ',
717       'drawdown!');
718     end;
719     if (MinDd<=0) and (GraphType=LogLog) then
720     begin
721       Valid:=false;
722       writeln('Minimum drawdown cannot be less than, or equal to zero ',
723       'for a log-log graph!');
724     end;
725     if (MinTime<=0) and ((GraphType=Semilog) or (GraphType=LogLog)) then
726     begin
727       Valid:=false;
728       writeln('Minimum time cannot be equal to, or less than zero
    for a ',
729       'log graph!');
730     end;
731     if (MinTime<0) and (GraphType=RootTime) then
732     begin
733       Valid:=false;
734       writeln('Minimum time cannot be less than zero for a root time ',
735       'graph!');
736     end;
737     if MinTime>=MaxTime then
738     begin
739       Valid:=false;
740       writeln('Maximum time is invalid in it''s relationship to
    minimum ',
741       'time!');
742     end;
743   until Valid=true;
744 end;  {Procedure ManualEntry}
745
```

```
746 Procedure GetMaxMin; {Get maximums and minimums for all the curves}
747 var I: integer;
748 begin
749   MaxTime:=-1e36; MinTime:=1e36;
750   MaxDd:=-1e30;MinDd:=1e30;
751   for CurveNum:=1 to NumOfCurves do
752   begin
753     for I:=1 to NumData[CurveNum] do
754     begin
755       if VRP[CurveNum]^.TimeVec[I]>MaxTime
756         then MaxTime:=VRP[CurveNum]^.TimeVec[I];
757       if VRP[CurveNum]^.TimeVec[I]<MinTime
758         then MinTime:=VRP[CurveNum]^.TimeVec[I];
759       if VRP[CurveNum]^.DdVec[I]>MaxDd
760         then MaxDd:=VRP[CurveNum]^.DdVec[I];
761       if VRP[CurveNum]^.DdVec[I]<MinDd
762         then MinDd:=VRP[CurveNum]^.DdVec[I];
763     end; {for I}
764   end; {for CurveNum}
765   if GraphType=LogLog then
766   begin MinLogDd:=Log(MinDd); MaxLogDd:=Log(MaxDd); end;
767   if (GraphType=LogLog) or (GraphType=SemiLog) then
    MinLogTime:=Log(MinTime);
768 end; {Procedure GetMaxMin}
769
770 Procedure LoadTimeLog; {Load log of time vector}
771 begin
772   for CurveNum:=1 to NumOfCurves do
773   begin
774     for I:=1 to NumData[CurveNum] do
775     VRP[CurveNum]^.LogTimeVec[I]:=Log(VRP[CurveNum]^.TimeVec[I]);
776   end; {for CurveNum}
777 end;
778
779 Procedure LoadDdLog; {Load log of drawdown vector}
780 begin
781   for CurveNum:=1 to NumOfCurves do
782   begin
783     for I:=1 to NumData[CurveNum] do
784     VRP[CurveNum]^.LogDdVec[I]:=Log(VRP[CurveNum]^.DdVec[I]);
785   end {for CurveNum}
786 end;
787
788 Procedure LoadRootTime; {Load root of time vector}
789 begin
790   for CurveNum:=1 to NumOfCurves do
791   begin
792     for I:=1 to NumData[CurveNum] do
793     VRP[CurveNum]^.RootTimeVec[I]:=Sqrt(VRP[CurveNum]^.TimeVec[I]);
794   end; {for CurveNum}
795 end;
796
797 {#----- Calculate screen plotting coordinates -----#}
798 Function LogX {Plotting coord., log X scale, screen graph}
799 (Num: real): integer;
800 begin
801   LogX:=round(Left+(Right-Left)*
802     (Num-MinLogGTime)/(MaxLogGTime-MinLogGTime));
803 end; {Function LogX}
```

```
804
805 Function LogY {Plotting coord., log Y scale, screen graph}
806 (Num: real): integer;
807 begin
808   LogY:=Bottom-round(Top+(Bottom-Top)*
809     (Num-MinLogGDd)/(MaxLogGDd-MinLogGDd));
810 end; {Function LogY}
811
812 Function LinearX {Plotting coord., linear X scale, screen graph}
813 (Num: real): integer;
814 begin
815   LinearX:=round(Left+(Right-Left)*(Num-MinGraphTime)/
816     (MaxGraphTime-MinGraphTime));
817 end; {Function LinearX}
818
819 Function LinearY {Plotting coord., linear Y scale, screen graph}
820 (Num: real): integer;
821 begin
822   LinearY:=round(Top+(Bottom-Top)*(Num-MinGraphDd)/
823     (MaxGraphDd-MinGraphDd));
824 end; {Function LinearY}
825
826 Function RootX {Plotting coord., root of time scale, screen graph}
827 (Num: real): integer;
828 {Calculates an X value for plotting on a root time scale, from a given
829 root of time value.}
830 begin
831   RootX:=round(Left+(Right-Left)*(Num-MinRootTime)/
832     (MaxRootTime-MinRootTime));
833 end; {Function RootX}
834 {#----- End of calculation of plotting coordinates -----#}
835
836 Procedure PlotShape {Plot a different shape for each curve on screen}
837 (X, Y, I: integer);
838 begin
839   Case
840     CurveNum of
841     1: begin                                    {Plot an X shape}
842       Plot(X,Y,I); Plot(X+1,Y+1,I); Plot(X-1,Y+1,I);
843       Plot(X+1,Y-1,I); Plot(X-1,Y-1,I);
844     end; {Case 1}
845     2: begin                                    {Plot a Block}
846       Plot(X-1,Y,I); Plot(X,Y,I); Plot(X+1,Y,I);
847       Plot(X-1,Y+1,I); Plot(X,Y+1,I); Plot(X+1,Y+1,I);
848       Plot(X-1,Y-1,I); Plot(X,Y-1,I); Plot(X+1,Y-1,I);
849     end; {Curve 2}
850     3: begin                                    {Plot a diamond shape}
851       Plot(X,Y-2,1); Plot(X-1,Y-1,1); Plot(X+1,Y-1,1);
852       Plot(X-2,Y,1); Plot(X+2,Y,1); Plot(X-1,Y+1,1);
853       Plot(X+1,Y+1,1); Plot(X,Y+2,1);
854     end; {Curve 3}
855   end; {of cases}
856 end; {Procedure PlotShape}
857
858 Procedure PlotData; {Control of data plotting on the screen}
859 var
860   OldX, OldY: integer;
861   Started: boolean;
862 begin
```

```
863   Started:=false;
864   for I:=1 to NumData[CurveNum] do
865   begin
866     if (VRP[CurveNum]^.TimeVec[I]>=MinGraphTime) and
867        (VRP[CurveNum]^.TimeVec[I]<=MaxGraphTime) and
868        (VRP[CurveNum]^.DdVec[I]>=MinGraphDd) and
869        (VRP[CurveNum]^.DdVec[I]<=MaxGraphDd)
870     then begin
871       case GraphType of
872         Linear: X:=LinearX(VRP[CurveNum]^.TimeVec[I]);
873         Semilog: X:=LogX(VRP[CurveNum]^.LogTimeVec[I]);
874         LogLog: X:=LogX(VRP[CurveNum]^.LogTimeVec[I]);
875         RootTime: X:=RootX(VRP[CurveNum]^.RootTimeVec[I]);
876       end; {of cases}
877       if GraphType<>LogLog then
878         Y:=LinearY(VRP[CurveNum]^.DdVec[I])
879       else
880         Y:=LogY(VRP[CurveNum]^.LogDdVec[I]);
881       PlotShape(X,Y,1);
882       if (Join=true) and (Started=true) then draw(OldX,OldY,X,Y,1);
883       OldX:=X; OldY:=Y;
884       if VRP[CurveNum]^.RateVec[I+1]<>VRP[CurveNum]^.RateVec[I]
885         then Started:=false else Started:=true;
886     end; {if Time}
887   end; {for I}
888 end; {Procedure PlotData}
889
890 {#----- Reference lines on the screen -----#}
891 Procedure PlotLinDdRL; {Linear drawdown ref. line on screen}
892 begin
893   TempVal:=MinGraphDd;
894   While TempVal<=MaxGraphDd do
895   begin
896     Y:=LinearY(TempVal);
897     Draw(Left,Y,Right,Y,1);
898     if TempVal>MinGraphDd
899     then begin
900       GotoXY(1,(Y div 8)+1);
901       if TempVal<0
902         then Short:=Format2(TempVal,5)
903         else Short:=Format2(TempVal,4);
904       write(Short);
905     end; {then}
906     TempVal:=TempVal+DdStep;
907   end; {while}
908   GotoXY(1,24);  Short:=Format2(MaxGraphDd,4); write(Short);
909   GotoXY(1,1);
910   if MinGraphDd<0
911     then Short:=Format2(MinGraphDd,5)
912     else Short:=Format2(MinGraphDd,4);
913   write(Short);
914   Draw(Left,Bottom,Right,Bottom,1);
915 end; {Procedure PlotLinDdRL}
916
917 Procedure PlotLinTimeRL; {Linear time ref. line on screen}
918 begin
919   TempVal:=MinGraphTime;
920   While TempVal<=MaxGraphTime do
921   begin
```

```
922     X:=LinearX(TempVal);
923     Draw(X,Top,X,Bottom,1);
924     if TempVal<MaxGraphTime*0.99
925     then begin
926       GotoXY((X div 8)-1,25); Short:=Format2(TempVal,4); write(Short);
927     end; {then}
928     TempVal:=TempVal+LinTimeStep;
929   end; {while}
930   GotoXY(73,25); Short:=Format2(MaxGraphTime,4); write(Short);
931 end; {Procedure PlotLinTimeRL}
932
933 Procedure PlotLogTimeRL; {Log of time ref. line on screen}
934 begin
935   for I:=round(MinLogGTime) to round(MaxLogGTime) do
936   begin
937     X:=LogX(I);
938     Draw(X,Top,X,Bottom,1);
939     if I<Round(MaxLogGTime)
940     then begin
941       GotoXY((X div 8)-2,25); Short:=Format2(ExpTen(I),4); write(Short);
942     end;
943   end;
944   GotoXY(72,25); Short:=Format2(ExpTen(round(MaxLogGTime)),4);
    write(Short);
945 end; {Procedure PlotLogTimeRL}
946
947 Procedure PlotLogDdRL; {Log of drawdown ref. line on screen}
948 begin
949   for I:=round(MinLogGDd) to round(MaxLogGDd) do
950   begin
951     Y:=LogY(I);
952     Draw(Left,Y,Right,Y,1);
953     if I>round(MinLogGDd)
954     then begin
955       GotoXY(1,(Y div 8)+1);
956       if ExpTen(I)>2 then write(ExpTen(I):5:0)
957       else write(ExpTen(I):5:3);
958     end;
959     TempVal:=Log(ExpTen(I)*4);
960     Y:=LogY(TempVal);
961     Draw(Left,Y,Right,Y,1);
962     if TempVal<round(MaxLogGDd)
963     then begin
964       GotoXY(1,(Y div 8)+1);
965       if ExpTen(TempVal)>2 then write(ExpTen(TempVal):5:0)
966       else write(ExpTen(TempVal):5:3);
967     end;
968     GotoXY(1,24); write(ExpTen(MinLogGDd):5:3);
969   end; {for}
970 end; {Procedure PlotLogDdRL}
971
972 Procedure PlotRootTimeRL; {Square root of time ref. line on screen}
973 begin
974   TempVal:=MinRootTime;
975   While TempVal<=MaxRootTime do
976   begin
977     X:=RootX(TempVal);
978     Draw(X,Top,X,Bottom,1);
979     if TempVal<MaxRootTime
```

```
    then begin
      GotoXY((X div 8),25); Short:=Format2(sqr(TempVal),4);
  write(Short);
    end; {then}
    TempVal:=TempVal+RootTimeStep;
  end; {while}
  GotoXY(74,25); Short:=Format2(sqr(MaxRootTime),4); write(Short);
end; {Procedure PlotRootTimeRL}
{#----- End of reference lines on the screen -----#}

Procedure GetAFile; {Control loading of discharge test data file}
var TempInt: integer;
begin
  ReadTestDataFile(VRP[CurveNum]^.TimeVec, VRP[CurveNum]^.DdVec,
    VRP[CurveNum]^.RateVec, TestType[CurveNum], WellType[CurveNum],
    Distance[CurveNum], NumData[CurveNum]);
  if NumData[CurveNum]=0 then
  begin
    ClrScr;
    write('Directory: which type of data file, ');
    TempInt:=CapOptions('Wtd, or Ftd? ');
    halt(TempInt+62);
  end;
  ViewAlterData            {Sumary of data read from file}
    (VRP[CurveNum]^.TimeVec, VRP[CurveNum]^.DdVec,
    VRP[CurveNum]^.RateVec, TestType[CurveNum], WellType[CurveNum],
    Distance[CurveNum], NumData[CurveNum]);
  TestFile[CurveNum]:=FileName;
  while pos('\',TestFile[CurveNum])<>0 do  {delete path name}
    delete(TestFile[CurveNum],1,pos('\',TestFile[CurveNum]));
end; {Procedure GetAFile}

Procedure PrepForGraph; {Prepare for producing a graph}
var Answer: ShortString;
begin
  GetGraphType;
  writeln;
  Remove0(GraphType); {Remove times (and drawdowns) <=0, adjust NumData}
  if (GraphType=SemiLog) or (GraphType=LogLog) then
    LoadTimeLog;               {Load log of time vectors}
  if GraphType=LogLog then
    LoadDdLog;                 {Load log of drawdown vectors}
  if GraphType=RootTime then
    LoadRootTime;              {Load square root of time vectors}
  GetMaxMin;           {Get the maximum and minimum times and drawdowns}
  writeln('Drawdown ranges from ',MinDd:7:3,' to ',MaxDd:7:3,'m.');
  writeln('Time ranges from ',MinTime:7:3,' to ',MaxTime:7:3,'min.');
  writeln('The graph will be based on these limits unless you enter ',
  'other limits');
  writeln('manually.');
  write('Do you want to alter any drawdown or time limits? ');
  Short:='YN'; Answer:=Response(Short);
  if Answer='Y' then ManualEntry;
  writeln; write('Join data points? ');
  Short:='YN'; Answer:=Response(Short);
  if Answer='Y' then Join:=true else Join:=false;
  if Device=Screen then
  begin GraphMaxMin; HiRes; end;
end; {Procedure PrepForGraph}
```

```
1038
1039 Procedure PaperPlot; {Paper plotting instructions for one curve to disk
     file}
1040 var
1041   Started: boolean;
1042   X, Y: integer;
1043 begin
1044   if (CurveNum<>1) then
1045   begin
1046     writeln(GF,'m100,1100');
1047     writeln(GF,'j',CurveNum:1); {Get next pen}
1048   end;
1049   if ((PapSize=Small) and (NumOfCurves<7)) or (PapSize=Large) then
1050   begin
1051     writeln(GF,'m',PlotLt+100+CurveNum*300,',',PlotT+50); {Print file
     name}
1052     writeln(GF,'p',TestFile[CurveNum]);
1053   end;
1054   Started:=false;
1055   for I:=1 to NumData[CurveNum] do
1056   begin
1057     if (VRP[CurveNum]^.TimeVec[I]>=MinTime)
1058       and (VRP[CurveNum]^.TimeVec[I]<=MaxTime)
1059       and (VRP[CurveNum]^.DdVec[I]>=MinDd)
1060       and (VRP[CurveNum]^.DdVec[I]<=MaxDd)
1061     then begin
1062       case GraphType of
1063         Linear: begin
1064           X:=LinearXP(VRP[CurveNum]^.TimeVec[I]);
1065           Y:=LinearYP(VRP[CurveNum]^.DdVec[I]);
1066         end;
1067         Semilog: begin
1068           X:=LogXP(VRP[CurveNum]^.LogTimeVec[I]);
1069           Y:=LinearYP(VRP[CurveNum]^.DdVec[I]);
1070         end;
1071         LogLog: begin
1072           X:=LogXP(VRP[CurveNum]^.LogTimeVec[I]);
1073           Y:=LogYP(VRP[CurveNum]^.LogDdVec[I]);
1074         end;
1075         RootTime: begin
1076           X:=RootXP(VRP[CurveNum]^.RootTimeVec[I]);
1077           Y:=LinearYP(VRP[CurveNum]^.DdVec[I]);
1078         end;
1079       end; {of cases}
1080       if (X>=PlotLt) and (X<=PlotR) and (Y>=PlotB) and (Y<=PlotT)
1081       then begin
1082         if Join and Started
1083         then writeln(GF,'d',X,',',Y,' n5')
1084         else writeln(GF,'m',X,',',Y,' n5');
1085         if VRP[CurveNum]^.RateVec[I+1]<>VRP[CurveNum]^.RateVec[I]
1086           then Started:=false else Started:=true;
1087       end
1088       else Started:=false;
1089     end; {if Time}
1090   end; {for I}
1091 end; {Procedure PaperPlot}
1092
1093 Procedure GetSetOfFiles; {Control the loading of a set of data files}
1094 var
```

```
1095   MaxCurves: integer;
1096 begin
1097   if Device=Screen then MaxCurves:=3 else MaxCurves:=MaxFiles;
1098   CurveNum:=1; writeln;
1099   repeat
1100     writeln('File No. ',CurveNum); GetAFile; Answer:='N';
1101     CurveNum:=CurveNum+1;
1102     if CurveNum<=MaxCurves then
1103     begin
1104       write('Another set of data? '); Short:='YN';
1105       Answer:=Response(Short); writeln;
1106     end;
1107   until (Answer='N') or (CurveNum>MaxCurves);
1108   NumOfCurves:=CurveNum-1;
1109 end; {Procedure GetSetOfFiles}
1110
1111 Procedure ScreenGraph; {Control production of a screen graph}
1112 begin
1113   if not SameFiles then GetSetOfFiles;
1114   PrepForGraph;
1115   CurveNum:=1;
1116   gotoXY(1,25); write('Press space bar to procede.');
1117   repeat
1118     PlotData;
1119     CurveNum:=CurveNum+1;
1120     Ch:='x';
1121     if CurveNum<=NumOfCurves then repeat Read(Kbd,Ch) until Ch=' ';
1122   until CurveNum>NumOfCurves;
1123   Ch:='x'; repeat Read(Kbd,Ch) until Ch=' ';
1124   gotoXY(1,25); write('                              ');
1125   case GraphType of
1126     Linear:  begin PlotLinDdRL; PlotLinTimeRL; end;
1127     Semilog: begin PlotLinDdRL; PlotLogTimeRL; end;
1128     LogLog:  begin PlotLogDdRL; PlotLogTimeRl; end;
1129     RootTime:begin PlotLinDdRL; PlotRootTimeRL; end;
1130   end; {cases}
1131   Ch:='x'; repeat Read(Kbd,Ch) until Ch<>'x';
1132   textmode; TextColor(green);
1133 end; {Procedure ScreenGraph}
1134
1135 Procedure DiskGraph; {Control production of a disk file graph}
1136 begin
1137   if not SameFiles then GetSetOfFiles;
1138   PrepForGraph;  {Get data max. min., allow manual alteration of max.
     min.}
1139   GraphMaxMin;   {Get max. min. for graph rather than for data}
1140   if GraphType=LogLog then
1141   begin
1142     LogDdRange:=(PlotT-PlotB)/760; LogTimeRange:=(PlotR-PlotLt)/760;
1143   end;
1144   OpenGraphFile; {Prepare to send plotter instructions to disk file}
1145   writeln(GF,'j1'); {Pick up pen number one}
1146   writeln(GF,'m',PlotLt,',',PlotT+50); {Print type of graph}
1147   Case GraphType of
1148     Linear: writeln(GF,'pLinear');
1149     SemiLog: writeln(GF,'pSemilog');
1150     LogLog: writeln(GF,'pLog-log');
1151     RootTime: writeln(GF,'pRoot-time');
1152   end; {of cases}
```

```
1153   PlotRefLines;  {Send ref. line instructions to plotter}
1154   for CurveNum:=1 to NumOfCurves do PaperPlot;
1155   writeln(GF,'h'); {Home pen holder}
1156   Close(GF);
1157 end; {Procedure DiskGraph}
1158
1159 Procedure PlotterGraph; {Control production of a graph on a plotter}
1160 var
1161   InFileName: ShortString;
1162 begin
1163   DiskGraph;
1164   InFileName:='TEMP.DAT';
1165   Assign(GF,InFileName); reset(GF);
1166   while not eof(GF) do
1167   begin
1168     readln(GF,Long);
1169     writeln(lst,Long);
1170   end;
1171   close(GF);
1172 end; {Procedure PlotterGraph}
1173 {#----- End of procedures and functions -----#}
1174
1175 begin {# main part of program}
1176   for I:=1 to MaxFiles do new(VRP[I]); Again:=false; SameFiles:=false;
1177   clrscr; TextColor(green);
1178   writeln('PLOTWTD;     PLOT Well Test Data');
1179   writeln('Discharge/Recharge test graphing program');
1180   repeat
1181     writeln;
1182     write('Do you want a ');
1183     Device:=Devices
1184       (CapOptions('Plotter graph, Screen graph, or output to
    Disk? ')-1);
1185     if SameFiles and (Device=Screen) and (NumOfCurves>3)
1186       then NumOfCurves:=3;
1187     Case Device of
1188       Plotter: PlotterGraph;
1189       Screen : ScreenGraph;
1190       Disk   : DiskGraph;
1191     end; {of cases}
1192     write('Another graph? '); Short:='YN';
1193     Answer:=Response(Short);
1194     if Answer='N' then Again:=false else Again:=true;
1195     if Again=true
1196     then begin
1197       write('Do you want to use the same files? '); Short:='YN';
1198       Answer:=Response(Short);
1199       if Answer='N' then SameFiles:=false else
1200       begin
1201         SameFiles:=true;
1202         writeln;
1203         writeln('   Note that if you have plotted a log-log graph
    then ',
1204         'drawdowns <=0 will');
1205         writeln('have been removed.  Similarly times <=0 will have
    been ',
1206         'removed if you have');
1207         writeln('plotted a semilog or log-log graph.');
1208         writeln;
```

```
1209       end; {else}
1210     end; {if Again}
1211   until not Again;
1212   {The code below should only be used when this program is used as a
     chain
1213    file.}
1214   for I:=1 to MaxFiles do dispose(VRP[I]);
1215   ChainTo('GWMENU.CHN',IOCode);
1216   if IOCode<>0 then
1217   writeln('Unable to chain to program GWMENU.CHN!');
1218 end. {#}
```

10. LISTING OF PROGRAM PLOTWTD (HEWLETT-PACKARD VERSION)

```
  1 Program PLOTWTD2_PAS;
  2 {  To graph a set of discharge test data on screen or a plotter.  This
  3  version is for a Hewlett-Packard Color Pro plotter.}
  4
  5 {$R+}
  6
  7 {#}{$I FIRST.SEG}
  8
  9 {#}{$I READ.PRC}
 10
 11 type
 12   GraphTypes=(Linear, SemiLog, LogLog, RootTime);
 13   Devices=(Screen, Disk);
 14   VecRec=record
 15     TimeVec, LogTimeVec, RootTimeVec, DdVec,
 16     LogDdVec, RateVec: MainVec;
 17   end;
 18
 19 var
 20   Valid, Again, SameFiles, Join, FullLines: Boolean;
 21   NumOfCurves, CurveNum: byte;
 22   Ch: Char;
 23   GenInt, Result, X, Y, X1: integer;
 24   NumData: array[1..10] of integer;
 25   MaxDd, MinDd, MaxTime, MinTime, LogDdRange, LogTimeRange: real;
 26   DdStep, MaxGraphDd, MinGraphDd, MinLogTime: real;
 27   MaxRootTime, MinRootTime, MaxGraphTime, MinGraphTime: real;
 28   MaxLogDd, MinLogDd, LinTimeStep, RootTimeStep: real;
 29   MinLogGDd, MaxLogGDd, MinLogGTime, MaxLogGTime: real;
 30   Distance: array[1..10] of real;
 31   TestFile: array[1..10] of MedString;
 32   Device: Devices;
 33   GraphType: GraphTypes;
 34   TestType: array[1..10] of Test; WellType: array[1..10] of Well;
 35   VRP: array[1..10] of ^VecRec;
 36   OutFileName: MedString; GF: text;
 37
 38 const
 39   PlotL=1000; PlotR=9500;   {L=Left, R=Right, T=Top, B=Bottom}
 40   PlotB=500;  PlotT=6900;
 41   Left=48; right=639; Top=0; Bottom=185;
 42   LnTen=2.302585093; MaxFiles=7;
 43
 44 Function FileExist          ## As in program PLOTWTD
 45
```

```
Function Log                ## As in program PLOTWTD

Function ExpTen             ## As in program PLOTWTD

Function Format2            ## As in program PLOTWTD

Procedure GraphMaxMin       ## As in program PLOTWTD

{#----- Calculate plotting coordinates for paper plot -----#}
Function LogXP {Produces an X value for plotting, given log of time}
(Num: real): integer;
begin
  LogXP:=round(PlotL+(PlotR-PlotL)*
    (Num-MinLogGTime)/(MaxLogGTime-MinLogGTime));
end; {Function LogXP}

Function LogYP              ## As in program PLOTWTD

Function LinearXP {Produces X value for plotting, given a time}
(Num: real): integer;
begin
  LinearXP:=round(PlotL+(PlotR-PlotL)*(Num-MinGraphTime)/
    (MaxGraphTime-MinGraphTime));
end; {Function LinearXP}

Function LinearYP {Produces a Y values for plotting, given a drawdown}
(Num: real): integer;
begin
  LinearYP:=round(PlotT+(PlotB-PlotT)*(Num-MinGraphDd)/
    (MaxGraphDd-MinGraphDd));
end; {Function LinearYP}

Function RootXP {Calculates X value for plotting on a root time scale,
  from a given root of time value.}
(Num: real): integer;
begin
  RootXP:=round(PlotL+(PlotR-PlotL)*
    (Num-MinRootTime)/(MaxRootTime-MinRootTime));
end; {Function RootXP}
{#------ End of calculation of plotting coordinates -----#}

Procedure OpenGraphFile; {Name a file, and open it
  ready to receive plotter instructions}
var
  Go: boolean;
begin
  repeat
    Go:=true;
    write('Please enter the full file name for the output data ');
    readln(OutFileName);
    if FileExist(OutFileName) then
    begin
      write('There is a file by that name, ');
      if CapOptions('Rewrite it, or Change file name? ')=1
      then Go:=true else Go:=false;
    end;
  until Go;
  Assign(GF,OutFileName); Rewrite(GF);
end; {Procedure OpenGraphFile}
```

```
105
106 {#----- Beginning of major procedure PlotRefLines (on paper) -----#}
107 Procedure PlotRefLines; {Produce and file ref. line instructions.
108 Send all reference line drawing instructions to a disk file.
109 The type of graph has been decided previously}
110 var
111   Tog: boolean;
112   I, J: integer;
113
114   procedure LogDdLines; {Ref. lines, log drawdown scale}
115   var
116     J, RefNum, IntLog: integer;
117     Drawdown: real;
118   begin
119     Drawdown:=ExpTen(MinLogGDd); writeln(GF,'sp1');
120     RefNum:=1; IntLog:=round(MinLogGDd);
121     if FullLines
122     then begin
123       J:=PlotB;
124       while J<=PlotT do
125       begin
126         if Tog
127         then begin
128           if (RefNum=1) or (RefNum=3) then
129             if Drawdown>=0.01
130             then writeln(GF,'pa',PlotL-520,', ',J,' lb',
    Format2(Drawdown,4),
131               #3)
132             else writeln(GF,'pa',PlotL-520,', ',J,' lb',
    Format2(Drawdown,5),
133               #3);
134           writeln(GF,'pa',PlotL,', ',J,' pd pa',PlotR,', ',J,' pu');
135         end
136         else begin
137           writeln(GF,'pa',PlotR,', ',J,' pd pa',PlotL,',',J,' pu');
138           if (RefNum=1) or (RefNum=3) then
139             if Drawdown>=0.01
140             then writeln(GF,'pa',PlotL-520,', ',J,' lb',
    Format2(Drawdown,4),
141               #3)
142             else writeln(GF,'pa',PlotL-520,', ',J,' lb',
    Format2(Drawdown,5),
143               #3);
144         end; {if then else Tog}
145         if RefNum=9
146         then begin RefNum:=1; IntLog:=IntLog+1; end
147         else RefNum:=RefNum+1;
148         Drawdown:=ExpTen(IntLog)*RefNum;
149         if Tog then Tog:=false else Tog:=true;
150         J:=LogYP(Log(Drawdown));
151       end; {while J}
152     end    {if FullLines}
153     else begin {Implied ref. lines}
154       writeln(GF,'pa',PlotL,',',PlotT,' pd pa',PlotR,',',PlotT,' pa',
    PlotR,
155       ',',PlotB,' pa',PlotL,',',PlotB,' pa',PlotL,',',PlotT,' pu');
156       J:=PlotB;
157       while J<=PlotT do
158       begin
```

```
159         if (RefNum=4) or (RefNum=1) then
160           writeln(GF,'pa',PlotL-520,',',J,' lb',Format2(Drawdown,4),#3);
161         if (J<>PlotB) and (J<>PlotT) and (RefNum=1) then
162           writeln(GF,'pa',PlotL,', ',J,' pd pa',PlotL+80,', ',J,' pu');
163         if (J<>PlotB) and (J<>PlotT) and (RefNum<>1) then
164           writeln(GF,'pa',PlotL,', ',J,' pd pa',PlotL+40,', ',J,' pu');
165         if RefNum=9
166         then begin RefNum:=1; IntLog:=IntLog+1; end
167         else RefNum:=RefNum+1;
168         Drawdown:=ExpTen(IntLog)*RefNum;
169         J:=LogYP(Log(Drawdown));
170       end; {while}
171       Drawdown:=ExpTen(MinLogGDd);
172       RefNum:=1; IntLog:=round(MinLogGDd); J:=PlotB;
173       while J<=PlotT do
174       begin
175         if (J<>PlotB) and (J<>PlotT) and (RefNum=1) then
176           writeln(GF,'pa',PlotR,', ',J,' pd pa',PlotR-80,', ',J,' pu');
177         if (J<>PlotB) and (J<>PlotT) and (RefNum<>1) then
178           writeln(GF,'pa',PlotR,', ',J,' pd pa',PlotR-40,', ',J,' pu');
179         if RefNum=9
180         then begin RefNum:=1; IntLog:=IntLog+1; end
181         else RefNum:=RefNum+1;
182         Drawdown:=ExpTen(IntLog)*RefNum;
183         J:=LogYP(Log(Drawdown));
184       end; {while}
185     end; {if then else FullLines}
186     writeln(GF,'pa',PlotL-800,',',(PlotT+PlotB) div 2+520);
187     Short:='Drawdown m';
188     for I:=1 to length(Short) do
189     begin
190       writeln(GF,'lb'+copy(Short,I,1)+#3);
191       writeln(GF,'pr-110,-190');
192     end;
193   end; {sub procedure LogDdLines}
194
195   procedure LinDdLines; {Plotter ref. lines, linear drawdown scale}
196   var
197     J: integer;
198     Drawdown: real;
199   begin
200     Drawdown:=MinGraphDd; writeln(GF,'sp1');
201     if FullLines
202     then begin
203       J:=PlotT;
204       while J>=PlotB do
205       begin
206         if Tog
207         then begin
208           writeln(GF,'pa',PlotL-520,', ',J,' lb',
    Format2(Drawdown,4),#3);
209           writeln(GF,'pa',PlotL,', ',J,' pd pa',PlotR,', ',J,' pu');
210         end
211         else begin
212           writeln(GF,'pa',PlotR,', ',J,' pd pa',PlotL,',',J,' pu');
213           writeln(GF,'pa',PlotL-520,', ',J,' lb',
    Format2(Drawdown,4),#3);
214         end; {if then else Tog}
215         Drawdown:=Drawdown+DdStep;
```

```
216         if Tog then Tog:=false else Tog:=true;
217         J:=LinearYP(Drawdown);
218       end; {while J}
219       if Drawdown<MaxGraphDd+DdStep*0.6 then
220       begin
221         writeln(GF,'pa',PlotR,',',PlotB,' pd pa',PlotL,',',PlotB,' pu');
222         writeln(GF,'pa',PlotL-520,', ',LinearYP(MaxGraphDd),' lb',
223           Format2(MaxGraphDd,4),#3)
224       end;
225     end    {if FullLines}
226     else begin {Implied ref. lines}
227       writeln(GF,'pa',PlotL,',',PlotT,' pd pa',PlotR,',',PlotT,' pa',
    PlotR,
228       ',',PlotB,' pa',PlotL,',',PlotB,' pa',PlotL,',',PlotT,' pu');
229       J:=PlotT;
230       while J>=PlotB do
231       begin
232         writeln(GF,'pa',PlotL-520,',',J,' lb',Format2(Drawdown,4),#3);
233         if (J<>PlotB) and (J<>PlotT) then
234           writeln(GF,'pa',PlotL,',',J,' pd pa',PlotL+80,',',J,' pu');
235         Drawdown:=Drawdown+DdStep;
236         J:=LinearYP(Drawdown);
237       end; {while}
238       if Drawdown<MaxGraphDd+DdStep*0.6 then
239         writeln(GF,'pa',PlotL-520,', ',LinearYP(MaxGraphDd),' lb',
240         Format2(MaxGraphDd,4),#3);
241       Drawdown:=MinGraphDd;
242       J:=PlotT;
243       while J>=PlotB do
244       begin
245         if (J<>PlotB) and (J<>PlotT) then
246           writeln(GF,'pa',PlotR,', ',J,' pd pa',PlotR-80,', ',J,' pu');
247         Drawdown:=Drawdown+DdStep;
248         J:=LinearYP(Drawdown);
249       end; {while}
250     end; {if then else FullLines}
251     writeln(GF,'pa',PlotL-800,',',(PlotT+PlotB) div 2+520);
252     Short:='Drawdown m';
253     for I:=1 to length(Short) do
254     begin
255       writeln(GF,'lb'+copy(Short,I,1)+#3);
256       writeln(GF,'pr-110,-190');
257     end;
258   end; {sub procedure LinDdLines}
259
260   procedure LinTimeLines; {Plotter linear time reference lines}
261   var
262     J: integer;
263     Time: real;
264   begin
265     Time:=MinGraphTime; Tog:=true;
266     if FullLines
267     then begin
268       J:=PlotL;
269       while J<=PlotR do
270       begin
271         if Tog
272         then begin
```

```
273           writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(Time,4),#3);
274           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotT,' pu');
275         end
276         else begin
277           writeln(GF,'pa',J,', ',PlotT,' pd pa',J,', ',PlotB,' pu');
278           writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(Time,4),#3);
279         end; {if then else Tog}
280         Time:=Time+LinTimeStep;
281         if Tog then Tog:=false else Tog:=true;
282         J:=LinearXP(Time);
283       end; {while J}
284     end    {if FullLines}
285     else begin {Implied ref. lines}
286       J:=PlotL;
287       while J<=PlotR do
288       begin
289         writeln(GF,'pa',J-160,', ',PlotB-200,' lb',Format2(Time,4),#3);
290         if (J<>PlotL) and (J<>PlotR) then
291           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotB+80,' pu');
292         Time:=Time+LinTimeStep;
293         J:=LinearXP(Time);
294       end; {while}
295       Time:=MinGraphTime;
296       J:=PlotL;
297       while J<=PlotR do
298       begin
299         if (J<>PlotL) and (J<>PlotR) then
300           writeln(GF,'pa',J,',',PlotT,' pd pa',J,',',PlotT-80,' pu');
301         Time:=Time+LinTimeStep;
302         J:=LinearXP(Time);
303       end; {while}
304     end; {if then else FullLines}
305     writeln(GF,'pa',(PlotR+PlotL) div 2+PlotL-1280,',',PlotB-400);
306     writeln(GF,'lbTime (min.)',#3);
307   end; {sub procedure LinTimeLines}
308
309   procedure LogTimeLines; {Plotter ref. lines, log of time scale}
310   var
311     J, RefNum, IntLog: integer;
312     Time: real;
313   begin
314     Time:=ExpTen(MinLogGTime); Tog:=true;
315     RefNum:=1; IntLog:=round(MinLogGTime);
316     if FullLines
317     then begin
318       J:=PlotL;
319       while J<=PlotR do
320       begin
321         if Tog
322         then begin
323           if RefNum=1 then
324             writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(Time,4),#3);
325           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotT,' pu');
326         end
327         else begin
328           writeln(GF,'pa',J,', ',PlotT,' pd pa',J,',',PlotB,' pu');
```

```
329           if RefNum=1 then
330             writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(Time,4),#3);
331         end; {if then else Tog}
332         if RefNum=9
333         then begin RefNum:=1; IntLog:=IntLog+1; end
334         else RefNum:=RefNum+1;
335         Time:=ExpTen(IntLog)*RefNum;
336         if Tog then Tog:=false else Tog:=true;
337         J:=LogXP(Log(Time));
338       end; {while J}
339     end    {if FullLines}
340     else begin {Implied ref. lines}
341       J:=PlotL;
342       while J<=PlotR do
343       begin
344         if RefNum=1 then
345           writeln(GF,'pa',J-160,',',PlotB-200,' lb',Format2(Time,4),#3);
346         if (J<>PlotL) and (J<>PlotR) and (RefNum=1) then
347           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotB+80,' pu');
348         if (J<>PlotL) and (J<>PlotR) and (RefNum<>1) then
349           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotB+40,' pu');
350         if RefNum=9
351         then begin RefNum:=1; IntLog:=IntLog+1; end
352         else RefNum:=RefNum+1;
353         Time:=ExpTen(IntLog)*RefNum;
354         J:=LogXP(Log(Time));
355       end; {while}
356       Time:=ExpTen(MinLogGTime);
357       RefNum:=1; IntLog:=round(MinLogGTime); J:=PlotL;
358       while J<=PlotR do
359       begin
360         if (J<>PlotL) and (J<>PlotR) and (RefNum=1) then
361           writeln(GF,'pa',J,', ',PlotT,' pd pa',J,', ',PlotT-80,' pu');
362         if (J<>PlotL) and (J<>PlotR) and (RefNum<>1) then
363           writeln(GF,'pa',J,', ',PlotT,' pd pa',J,', ',PlotT-40,' pu');
364         if RefNum=9
365         then begin RefNum:=1; IntLog:=IntLog+1; end
366         else RefNum:=RefNum+1;
367         Time:=ExpTen(IntLog)*RefNum;
368         J:=LogXP(Log(Time));
369       end; {while}
370     end; {if then else FullLines}
371     writeln(GF,'pa',(PlotR+PlotL) div 2+PlotL-1280,',',PlotB-400);
372     writeln(GF,'lbTime (min.)',#3);
373   end; {sub procedure LogTimeLines}
374
375   procedure RootTimeLines; {Plotter square root of time reference lines}
376   var
377     J: integer;
378     RTime: real;
379   begin
380     RTime:=MinRootTime; Tog:=true;
381     if FullLines
382     then begin
383       J:=PlotL;
384       while J<=PlotR do
385       begin
386         if Tog
```

```
387         then begin
388           writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(sqr(RTime),4),#3);
389           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotT,' pu');
390         end
391         else begin
392           writeln(GF,'pa',J,', ',PlotT,' pd pa',J,', ',PlotB,' pu');
393           writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(sqr(RTime),4),#3);
394         end; {if then else Tog}
395         RTime:=RTime+RootTimeStep;
396         if Tog then Tog:=false else Tog:=true;
397         J:=RootXP(RTime);
398       end; {while J}
399     end    {if FullLines}
400     else begin {Implied ref. lines}
401       J:=PlotL;
402       while J<=PlotR do
403       begin
404         writeln(GF,'pa',J-160,', ',PlotB-200,' lb',
    Format2(sqr(RTime),4),#3);
405         if (J<>PlotL) and (J<>PlotR) then
406           writeln(GF,'pa',J,', ',PlotB,' pd pa',J,', ',PlotB+80,' pu');
407         RTime:=RTime+RootTimeStep;
408         J:=RootXP(RTime);
409       end; {while}
410       RTime:=MinGraphTime;
411       J:=PlotL;
412       while J<=PlotR do
413       begin
414         if (J<>PlotL) and (J<>PlotR) then
415           writeln(GF,'pa',J,',',PlotT,' pd pa',J,',',PlotT-80,' pu');
416         RTime:=RTime+RootTimeStep;
417         J:=RootXP(RTime);
418       end; {while}
419     end; {if then else FullLines}
420     writeln(GF,'pa',(PlotR+PlotL) div 2+PlotL-1280,',',PlotB-400);
421     writeln(GF,'lbTime (min.)',#3);
422   end; {sub procedure RootTimeLines}
423
424 begin {# main part of procedure PlotRefLines}
425   Tog:=true;
426   if GraphType=LogLog then LogDdLines else LinDdLines;
427   Case GraphType of
428     Linear:   LinTimeLines;
429     SemiLog:  LogTimeLines;
430     LogLog:   LogTimeLines;
431     RootTime: RootTimeLines;
432   end; {of cases}
433 end; {Procedure PlotRefLines}
434 {#----- End of major procedure PlotRefLines (on paper) -----#}
435
436 Procedure GetGraphType; {What type of graph does user want?  ie. scales}
437 begin
438   write('What type of graph, ');
439   GraphType:=
440     GraphTypes(CapOptions('Linear, Semi-log, lOg-log, Root-time? ')-1);
441   if Device<>Screen then
442   begin
```

```
443     write('Do you want full reference lines? ');
444     if Response('YN')='Y' then FullLines:=true else FullLines:=false;
445   end; {if Device}
446 end; {Procedure GetGraphType}
447
448 Procedure Remove0           ## As in program PLOTWTD
449
450 Procedure ManualEntry       ## As in program PLOTWTD
451
452 Procedure GetMaxMin         ## As in program PLOTWTD
453
454 Procedure LoadTimeLog       ## As in program PLOTWTD
455
456 Procedure LoadDdLog         ## As in program PLOTWTD
457
458 Procedure LoadRootTime      ## As in program PLOTWTD
459
460 {#----- Calculate screen plotting coordinates -----#}
461 Function LogX               ## As in program PLOTWTD
462
463 Function LogY               ## As in program PLOTWTD
464
465 Function LinearX            ## As in program PLOTWTD
466
467 Function LinearY            ## As in program PLOTWTD
468
469 Function RootX              ## As in program PLOTWTD
470 {#----- End of calculation of plotting coordinates -----#}
471
472 Procedure PlotShape         ## As in program PLOTWTD
473
474 Procedure PlotData          ## As in program PLOTWTD
475
476 {#----- Reference lines on the screen -----#}
477 Procedure PlotLinDdRL       ## As in program PLOTWTD
478
479 Procedure PlotLinTimeRL     ## As in program PLOTWTD
480
481 Procedure PlotLogTimeRL     ## As in program PLOTWTD
482
483 Procedure PlotLogDdRL       ## As in program PLOTWTD
484
485 Procedure PlotRootTimeRL    ## As in program PLOTWTD
486 {#----- End of reference lines on the screen -----#}
487
488 Procedure GetAFile          ## As in program PLOTWTD
489
490 Procedure PrepForGraph      ## As in program PLOTWTD
491
492 Procedure PlotPoint {Plot one point on the graph}
493   (Pen: byte);
494 begin
495   if Pen=0
496   then writeln(GF,'pr25,25,-50,-50,25,25,25,-25,-50,50 pu')
497   else writeln(GF,'pd pr25,25,-50,-50,25,25,25,-25,-50,50 pu');
498 end; {Procedure PlotPoint}
499
500 Procedure PaperPlot; {Paper plotting instructions for one curve to disk
    file}
```

```
501 var
502   Started: boolean;
503   X, Y: integer;
504 begin
505   if (CurveNum<>1) then
506   begin
507     writeln(GF,'pa100,1100');
508     writeln(GF,'sp',CurveNum:1); {Get next pen}
509   end;
510   if NumOfCurves<8 then
511   begin
512     writeln(GF,'pa',PlotL+1000+CurveNum*1000,',',PlotT+200); {File name}
513     writeln(GF,'lb',TestFile[CurveNum],#3);
514   end;
515   Started:=false;
516   for I:=1 to NumData[CurveNum] do
517   begin
518     if (VRP[CurveNum]^.TimeVec[I]>=MinTime)
519       and (VRP[CurveNum]^.TimeVec[I]<=MaxTime)
520       and (VRP[CurveNum]^.DdVec[I]>=MinDd)
521       and (VRP[CurveNum]^.DdVec[I]<=MaxDd)
522     then begin
523       case GraphType of
524         Linear: begin
525           X:=LinearXP(VRP[CurveNum]^.TimeVec[I]);
526           Y:=LinearYP(VRP[CurveNum]^.DdVec[I]);
527         end;
528         Semilog: begin
529           X:=LogXP(VRP[CurveNum]^.LogTimeVec[I]);
530           Y:=LinearYP(VRP[CurveNum]^.DdVec[I]);
531         end;
532         LogLog: begin
533           X:=LogXP(VRP[CurveNum]^.LogTimeVec[I]);
534           Y:=LogYP(VRP[CurveNum]^.LogDdVec[I]);
535         end;
536         RootTime: begin
537           X:=RootXP(VRP[CurveNum]^.RootTimeVec[I]);
538           Y:=LinearYP(VRP[CurveNum]^.DdVec[I]);
539         end;
540       end; {of cases}
541       if (X>=PlotL) and (X<=PlotR) and (Y>=PlotB) and (Y<=PlotT)
542       then begin
543         if Join and Started
544         then begin
545           writeln(GF,'pd pa',X,',',Y); PlotPoint(0);
546           writeln(GF,'pa',X,',',Y)
547         end
548         else begin writeln(GF,'pa',X,',',Y); PlotPoint(1) end;
549         if VRP[CurveNum]^.RateVec[I+1]<>VRP[CurveNum]^.RateVec[I]
550           then Started:=false else Started:=true;
551       end
552       else Started:=false;
553     end; {if Time}
554   end; {for I}
555 end; {Procedure PaperPlot}
556
557 Procedure GetSetOfFiles     ## As in program PLOTWTD
558
559 Procedure ScreenGraph       ## As in program PLOTWTD
```

```
560
561 Procedure DiskGraph; {Control production of a disk file graph}
562 begin
563   if not SameFiles then GetSetOfFiles;
564   PrepForGraph;  {Get data max. min., allow manual alteration of max.
    min.}
565   GraphMaxMin;   {Get max. min. for graph rather than for data}
566   OpenGraphFile; {Prepare to send plotter instructions to disk file}
567   writeln(GF,'sp1'); {Pick up pen number one}
568   writeln(GF,'pa',PlotL,',',PlotT+200); {Print type of graph}
569   Case GraphType of
570     Linear: writeln(GF,'lbLinear',#3);
571     SemiLog: writeln(GF,'lbSemilog',#3);
572     LogLog: writeln(GF,'lbLog-log',#3);
573     RootTime: writeln(GF,'lbRoot-time',#3);
574   end; {of cases}
575   PlotRefLines;  {Send ref. line instructions to plotter}
576   for CurveNum:=1 to NumOfCurves do PaperPlot;
577   writeln(GF,'pa0,0;sp0;'); {Home pen holder, replace pen}
578   Close(GF);
579 end; {Procedure DiskGraph}
580 {#----- End of procedures and functions -----#}
581
582 begin {# main part of program}
583   for I:=1 to MaxFiles do new(VRP[I]); Again:=false; SameFiles:=false;
584   clrscr; TextColor(green);
585   writeln('PLOTWTD;     PLOT Well Test Data');
586   writeln('Discharge/Recharge test graphing program');
587   repeat
588     writeln;
589     write('Do you want a ');
590     Device:=Devices
591       (CapOptions('Screen graph, or output to Disk? ')-1);
592     if SameFiles and (Device=Screen) and (NumOfCurves>3)
593       then NumOfCurves:=3;
594     Case Device of
595       Screen : ScreenGraph;
596       Disk   : DiskGraph;
597     end; {of cases}
598     write('Another graph? '); Short:='YN';
599     Answer:=Response(Short);
600     if Answer='N' then Again:=false else Again:=true;
601     if Again=true
602     then begin
603       write('Do you want to use the same files? '); Short:='YN';
604       Answer:=Response(Short);
605       if Answer='N' then SameFiles:=false else
606       begin
607         SameFiles:=true;
608         writeln;
609         writeln('   Note that if you have plotted a log-log graph
    then ',
610         'drawdowns <=0 will');
611         writeln('have been removed.  Similarly times <=0 will have
    been ',
612         'removed if you have');
613         writeln('plotted a semilog or log-log graph.');
614         writeln;
615       end; {else}
```

```
616     end; {if Again}
617   until not Again;
618   {The code below should only be used when this program is used as a
    chain
619    file.}
620   for I:=1 to MaxFiles do dispose(VRP[I]);
621   ChainTo('GWMENU.CHN',IOCode);
622   if IOCode<>0 then
623   writeln('Unable to chain to program GWMENU.CHN!');
624 end. {#}
```

Chapter 7

Chapter one explained some computer applications which calculated the coefficients of the well equation from drawdown data recorded in a pumped well. This chapter deals with a program which may be used to analyse data from piezometers in an effort to both discover the type of aquifer being tested, and to quantify the parameters of that aquifer.

An explanation of all the means that a hydrogeologist may use to decide on the type of aquifer with which he is dealing is not within the scope of this book. However, this program provides a curve fitting tool which is capable of finding the best fit between a given aquifer's theoretical response and a set of real data, and this can be a great aid in the examination of field data. (Of course this program covers a selection of only a few aquifer types and boundary configurations. No doubt more will be handled by future programs of a similar type.)

While I believe that users may find that this program is the most useful in this book, it is also the most dangerous. This program gives answers to questions. If the wrong question was asked, the program will still give an answer; the best one it can. It cannot tell you which question you should ask! It is inevitable that some people will misuse program ANALYZE. Why is it that when a computer gives the wrong answer the user can so often avoid any responsibility by saying "that was due to a computer error"? If he analysed an aquifer test by using the wrong class of type curve would he be able to say that the error was due to a type curve error? It seems that the more powerful the machine, the greater is it's potential for misuse. The automobile is a good example; an essential of the modern life style, but in the hands of a fool it is a lethal weapon. So it is with this program, used wisely it should be of great use, used carelessly it will mislead you.

Program ANALYZE can be used to find the parameter values giving the best fit for a given aquifer type and a given set of field data. No effort has been made to write a program which will decide, for the user, the type of aquifer with which he is dealing; rather, when the user decides on the most likely aquifer configuration he can use this program to aid in quantifying transmissivity, storage coefficient, leakage coefficient, boundary distance, etc. Also, by allowing the hydrogeologist to compare the best fit curve for a particular aquifer type to his field data, some help will be provided in refining the conceptual model.

1. AIM OF THE PROGRAM

The computerised calculation of transmissivity from a selected segment of data is straight forward, and has been implemented in earlier programs such as in Clarke (1987). The computerised calculation of storage coefficient when transmissivity is known was also explained in that work. The fitting of a Theis drawdown curve to a set of data was done by McElwee in his program Theisfit (1980), and the fitting of a Leaky Artesian Aquifer type curve to a set of data was done by Cobb et al. in program Leakyfit (1982). (The former program was written in Basic, and the original versions of the latter two were written in Fortran.) On the other hand, all of these earlier programs had the disadvantage that the user had to make frequent reference to written notes on his data, or at least to his memory, and had to type numbers into the computer at the time of the analysis.

ANALYZE is capable of performing the jobs of all these programs and also covers curve fitting for bounded, strip, and unconfined aquifers; and the user need make minimal reference to written notes or memory. Most of the information required in performing the analyses is available on screen while the job is being done. The user loads a file containing his discharge test data into the computer from disk, then indicates which of those data are to be considered for his analysis by marking them on a graph on the computer's screen. Just as much control over the parts of the data that are used for an analysis is available here as when the job is done by the more traditional manual methods. The discharge test data are typed into the computer once, (probably by use of program DTDHA) and then the same data can be used for as many analyses as are required.

The curve fitting techniques of McElwee (TheisFit) and Cobb et al. (LeakyFit) are used in program ANALYZE, but a more general method which I have called the Diminishing Adjustment method is used for most of the solutions.

2. LIMITATIONS OF THE PROGRAM

It cannot be too greatly stressed that computer analysis is no substitute for expert knowledge and sound judgement. If this program is used in an inappropriate way, then it will produce useless answers; worse, it will produce misleading answers. If, for example, a Theis fit is found for all of the data from a discharge test showing evidence of a bounded aquifer, then the resulting values for transmissivity and storage coefficient will have little meaning.

3. USING ANALYZE

It will be assumed that the user has a file of discharge test data which he wants to analyse. The simplest way of getting field data into the data file is to type them in using program DTDHA. If you prefer, a Spread Sheet program such as Lotus 1-2-3 or any ASCII file editor program or word processor could be used for producing the data file, so long as the file format rules are strictly followed. (See Appendix A for a description of the data file formats used in these programs, and Appendix D for information on conversion of Lotus data to the WTD file format used in these programs.) If the user is wanting only to familiarise himself with the program, then he can produce synthetic data using the programs of Chapter Two. This course is highly recommended, as it will give some feel for using the program.

ANALYZE is accessed from the GW set of programs primary menu. Refer to the notes on Getting Started in the Introduction if you are not sure how to get to this point. Once ANALYZE has been called up it will ask for the name of the file containing your data, and having been given that name, it will attempt to load the file. If the file is not found, a message will be given to that effect. Path names may be given with the file name, but if you keep all your discharge test data files on a particular menu then writing the path for every file as it is required may become irksome. (Refer to the notes on the use of the DOS Append command and on customising file GW.BAT in the Preliminary section.)

After program ANALYZE has successfully loaded your data file, you will be given the opportunity of viewing the data, and then you will be informed of the maximums and minimums for time and drawdown within your data. This having been done, pressing any key will get you to the ANALYZE main menu. ANALYZE has two levels of menus, and from this main menu you must specify the aquifer type to be assumed for analysis. It is not essential to entirely restrict your analysis to the part of the program specifically dealing with that aquifer type from which your data came. For example, if your data came from a strip aquifer, you can still use the confined aquifer section to have transmissivity and storage coefficient calculated for early times, before the boundary effects begin to show up. It is in decisions such as this that the exercise of sound human judgement is essential.

If you are to use the program to full advantage then you must understand the way in which solutions and fits are obtained, at least in general terms. It is not necessary for you to understand the way the program works from a programmer's point of view, but only the ways in which the values given relate to your data. This sort of understanding will give an appreciation of

the validity, or otherwise, of a particular set of calculated aquifer parameters.

3.1. The graph

When it is time to pick out the parts of your field data that are to be used for a particular analysis they are plotted onto a semilogarithmic graph on the screen, with times in minutes on the x scale and drawdowns in metres on the y scale. Only the time of the first time reference line will be displayed in the lower left corner; all other time reference lines are in multiples of ten from this starting point. (The first time reference line will normally be one minute, so the next will be 10mins., 100mins. etc. The first time reference line is labelled to only one decimal place, so if there are readings in your file for a time of less than 0.1 minutes, this first line will be labelled 0.0, although it will in fact be 0.01mins., or less.) The drawdown reference lines will be more fully labelled.

4. A CONFINED AQUIFER

This is the first alternative given in the ANALYZE main menu.

The Theis equation (Theis, 1935) is the basis of this part of program ANALYZE; and this is, in a sense, the core of the program. The solution to the Theis equation is given by:

$$s = \frac{Q\ W(u)}{4\ Pi\ T} \qquad (7.1)$$

where

$$u = \frac{r^2\ S}{4\ T\ t}\ , \qquad (7.2)$$

T is transmissivity,
S is storage coefficient,
r is the distance between the discharging well and the piezometer in which the drawdown was measured,
t is the duration of discharge,
Q is the discharge rate, and
W(u) is the well function of u.

(The Theis equation is explained in many basic groundwater texts, so it will not be described again here.)

If the facilities in this section are to be entirely valid, then strict criteria must be met with. The aquifer must be infinite in extent, homogeneous, isotropic, and constant in thickness; and it must release water to the well with no delay. The well must have negligible diameter and negligible storage. In practice, useful work can be done when conditions fall far short of these criteria.

4.1. Calculation of transmissivity

To have transmissivity calculated, press key 1 at the confined aquifer menu. This will cause the screen to clear and a semilogarithmic graph to be drawn using the data from the file which you loaded when you entered this program. A message will briefly be displayed at the bottom of the screen asking you to "Set a marker at the right end", followed by "L=left, R=right, O=OK, T=time".

Now you must indicate the higher time limit of the section of data that you want used for calculation of transmissivity. This is done by moving the marker which is at the extreme left of the graph along the graph by pressing the R and L keys until it is where you want it. Every time you press the R key, the marker will move five plots toward the right (unless it is very near the end of the data); if you hold down the R key, the marker will move continuously toward the right. L will cause the marker to move one plot to the left. If you want to know the time and/or discharge rate associated with the plot at the marker's current position, press T.

You will notice that all plots to the left of this first marker will be red, while those to the right of it will be green; the aim is to make all those plots that you want used for the analysis red. When you have the first marker at the place you want it, press O (not zero). This will cause the position of the right hand, or first, marker to be fixed in it's present position, and a message will be displayed "First marker set". This will shortly be followed by "Set a marker at the left end", and then again by "L=left, R=right, O=OK, T=time".

The next step is to move the marker indicating the lower time limit of the data to be used for the calculation. This is achieved in exactly the same way as with the first pointer. The difference on the screen is that now the plots to the left of the marker that you are moving will revert to their original green. The plots on the graph that are red when you press the O key are those that will be used for calculation of transmissivity. If this sounds complicated in description, it is simple in execution.

The delay caused by the time taken for the actual linear regression of the indicated data, and calculation of transmissivity, will probably be imperceptible. "Transmissivity = x" will be displayed, where x will be replaced by the calculated value for your chosen data. This will remain on the screen until you press a key; then it will be replaced by the message "Another Transmissivity?". Here you can answer with Y to repeat the above procedure, perhaps with a different section of data, or you can press N to return to the Confined Aquifer menu.

Calculation of transmissivity is the simplest of all the options on any of the menus, but will be found to be very useful. It is essential that you realize that the transmissivity value produced by this option is the same as would be obtained by plotting a segment of the data on semilogarithmic graph paper, measuring the delta s slope for a straight line fitting those data, and calculating transmissivity by using the well known equation:

$$T = \frac{0.183\ Q}{\text{Delta s}} \qquad (7.3)$$

T and Q were defined above, Delta s is the slope (ie. the drawdown for each $\log_{10}$ cycle) of the best fit straight line when the data is plotted on semilogarithmic paper (see Bouwer, 1987 p99, Freeze and Cherry, 1979, p347, or Marino and Luthin, 1982, p297). It is important that this be born in mind whenever you use the transmissivity option. Whenever this method is inappropriate for the data concerned, then the calculated transmissivity will be invalid. Transmissivities calculated for leaky confined aquifers, bounded aquifers, and strip aquifers will be valid so long as they are calculated for the data collected before the effects of leakage or boundaries have shown up. Take care to choose data from times late enough for the Delta s slope to approximate a straight line, and beware of partial penetration effects. There may well be occasions when boundary effects show up before the straight line stage has been reached, in this case perhaps TheisFit (described below) should be used to obtain a simultaneous T and S for early data. In the case of a semistrip aquifer (high T inside a strip, and lower T outside), quantifying T by this method for very late times should give the transmissivity outside of the strip.

I suggest that this feature will be found useful in obtaining transmissivities from data recorded in both a pumped well and an observation well,

and in obtaining a first approximation of transmissivity before running many of the more advanced curve fitting options.

This is the only option of program ANALYZE that is applicable to data from a discharging well, because of the difficulty of meaningfully defining R in this case.

4.2. Calculation of storage coefficient

Before storage coefficient is calculated, the time/drawdown/discharge rate datum set to be used for the calculation must be indicated on the screen. This is done in a similar way to the indication of the section of data to be used for calculation of transmissivity described above, with the difference that here there is only one marker to be set. As soon as the marker is in place the 0 key may be pressed. Transmissivity will now be requested and storage coefficient will be calculated with no noticeable delay. For some values of time, discharge rate, and transmissivity, it will not be possible for the program to calculate a storage coefficient capable of producing the given drawdown; in these cases the message will be displayed "Storage failed to converge".

After either the storage coefficient is calculated, or the solution is found to be impossible, press any key to move on. You may then have another storage coefficient calculated, or go back to the Confined Aquifer menu.

METHOD The Theis equation in it's usual form is used to calculate drawdown; given transmissivity, storage coefficient, discharge rate, distance from the discharging well, and total duration of discharge. In relation to the screen graph, the Theis equation is such that for any given plot and transmissivity, there is only one value of storage coefficient possible (if the aquifer is a simple confined one). The application in this program uses Newtons method to solve the Theis equation for storage coefficient, given values for all the other variables (Clarke, 1987).

This option purposely does not calculate the transmissivity to be used for the assessment of storage. It was felt that as the evaluation of transmissivity requires expert judgement, and should not be left to an automated procedure. The transmissivity entered before the calculation will greatly effect the resulting storage coefficient, so care must be exercised.

4.3. Fitting a Theis curve to the data

TheisFit (McElwee, 1980) calculates both a storage coefficient and transmissivity to best suit a segment of your data. That segment of data is marked out in exactly the same way as was described for the calculation of

transmissivity. As in the calculation of T, the maximum number of data that can be used in TheisFit is 100; if your indicated segment includes more than this it will be reduced with the message "Selecting a subset of the data", and the record numbers of the data sets used will be displayed.

On pressing the 0 (OK) key after marking out your data, the graph will be replaced with the message "TheisFit has started" and then a series of intermediate transmissivity and storage values will be displayed. The displayed values should, and usually will, quickly converge toward a solution which will then be displayed with the final value of the root mean square error. In operations such as this the speed of your computer will be apparent. A basic PC without a numeric co-processor will be a little slow if a large number of time/drawdown values are involved.

After you have had an opportunity to make a note of the calculated values (you should also note the root mean square error), the graph will be displayed again, this time with the fitted Theis curve on it, in addition to your field data. You may compare the fitted data (yellow) against your selected field data (red) and visually asses the accuracy of the fit. The graphic quality of the fit may be compared with the root mean square error displayed previously. (Some colours may be different on some computers.)

Procedure TheisFit uses a translated and modified version of the original program by McElwee (1980) to statistically fit a Theis type curve to a set of data. It should be found valuable for quickly and easily evaluating both transmissivity and storage coefficient for your data as a whole, or for some part of your data. Again, this can easily be misused, so take care.

As this is a complex procedure, it will probably fail to converge to a solution at times, and may even cause a computer error in a few cases. It is possible, although unlikely, that the program will, under some unforeseen circumstances, get into an endless loop. If so, all that can be done is to reboot (hold down Ctrl and Alt while pressing Del) and perhaps try some other section of the data. At the time of writing this, I know of no data that will cause Theisfit to produce a computer error, or get into a dead loop.

5. LEAKY CONFINED AQUIFERS

Just as the Theis equation is basic to the analysis of data from a confined aquifer so the Hantush-Jacob (1955) Leaky Aquifer equation is basic to the analysis of data from a leaky aquifer. The solution to this equation is given by:

$$s = \frac{Q\ W(u,r/L)}{4\ Pi\ T}\ , \qquad (7.4)$$

where L is the leakage coefficient,
r is the distance between the discharging well and the piezometer where the drawdown is measured,
Q is the discharge rate,
T is transmissivity,
W(u,r/L) is the Leaky Artesian Well function,
and u is described in equation (7.2) above.

5.1. Calculation of leakage coefficient

The calculation of the leakage coefficient is similar to the calculation of storage, from the point of view of the user. The difference from the users point of view is that since it is leakage that we are now interested in, the chosen point should be one from a part of the graph where leakage has become significant. This calculation requires that the user enters both transmissivity and storage coefficient.

Just as the Theis equation can be solved for storage, so can the Leaky Artesian Well function be solved for leakage coefficient if both transmissivity and storage coefficient are known. The previous two options can be used to evaluate T and S, and then this option can solve for leakage. The user should be aware that each of these latter two evaluations depends on it's predecessor for it's accuracy, so an error may tend to accumulate.

This option will probably find most use in producing an approximate value of leakage which may then be used as a first approximation for LeakyFit.

The method used to evaluate leakage is Successive Bisection. It is not possible (given the authors knowledge, or lack thereof) directly to calculate a leakage for a given drawdown, but it is possible to calculate a drawdown for a given leakage. This method starts with two experimental values for leakage; one is too low, and the other is too high. A mid point is taken, and a drawdown is calculated for this. If this drawdown is found to indicate that the current experimental value for leakage is too low, then the old low limit is discarded and this becomes the new low limit. Similarly, if the experimental value is too high, it will become the new high limit. The mid-point is now taken from the two limits, which are closer together than they were, and the process is repeated until the experimental drawdown agrees with

the recorded drawdown with a negligible error. (Please read the remarks on definition of Leakage Coefficient on page 142.)

5.2. Fitting a leaky aquifer type curve to the data

LeakyFit was translated and modified from an original Fortran language version (Cobb et al., 1982). It fits a leaky artesian aquifer type curve to a set of data. (The original Fortran version produced a best fit to data from a group of piezometers.) While TheisFit produces it's own first estimates for transmissivity and storage coefficient, this program must be given a starting value for those two parameters, as well as leakage coefficient. (The thickness of the semiconfining layer will also be requested, but the value given by the user will have no effect on the iterative solution; it will only be used to calculate the vertical hydraulic conductivity after a solution has been reached.)

This is an even more complex procedure than TheisFit, so it is prone to failure. The most likely problem is that it will fail to converge to a solution. This is more likely to happen if the initial guesses are wide of the mark, so it is worth taking some care in obtaining reasonable values for transmissivity and storage coefficient before running LeakyFit (As for TheisFit, I know of no data that will cause this program to crash at the present.)

Selection of data for curve fitting

Selection of the data to be used for LeakyFit is very similar to selection of the data for all of the remaining curve fitting routines. First you must inform the program of the number of data points you want used for the curve fitting.

From experience, it seems that LeakyFit is much more likely to converge to a solution if a larger number of points are used. I suggest using at least ten points, and if convergence is not achieved you might try twenty. The other curve fitting routines will probably achieve convergence quite successfully with a smaller number of points, but in any case enough points should be given to properly represent the trend of the data as a whole.

The method of using all (or at least up to 100) of the data points as in TheisFit is not used in this and the following routines mainly because the time taken to obtain a solution is roughly in proportion to the number of points used.

You are asked to set a marker at the right end (the late time end) of the data; this is identical to the setting of the marker for calculation of

storage. Once this marker is set, the other markers will be placed, evenly spaced, through all the data to the left of the first mark. You may be content to leave these markers where they are, or you may wish to move one or more of them, because of data of doubtful validity or other reasons. The marker immediately to the left of that which you have just placed will be a different colour to the others; this signifies that this is the marker which is currently being placed. It may be moved right or left (R or L keys), or it may be left where it is by pressing the O (OK) key. You will be given an opportunity to move all the markers in this way.

After the last marker has been placed the program will request estimates of the transmissivity, storage coefficient, and leakage coefficient. Iteration toward a solution begins as soon as the last value is entered. If LeakyFit is to achieve convergence it will probably do so in a fairly small number of iterations, most likely no more than 15.

On achieving convergence the best fit values for transmissivity, storage coefficient, and leakage coefficient will be displayed, as well as the root mean square error. Press any key and you will be shown the graph as you left it, with the fitted data displayed in yellow against your data in red.

6. A BOUNDED CONFINED AQUIFER

The boundary considered here is a water tight, impervious, or discharge boundary (three names for the same thing). Boundaries are simulated by the use of image wells as explained in Chapter two, pp. 126 and 127, and also in Bouwer (1978), Freeze and Cherry (1979), Marino and Luthin (1982), and Clarke (1987). The aquifer is assumed to be infinite (apart from the boundary), homogeneous, and fully confined. The pumped well and piezometer must both be on the same side of the boundary.

6.1. Curve fitting for T, S, and image well distance

The process up to and including the selection of the data to be used for the curve fitting is identical here to that used for LeakyFit, but the computational method used for the curve fitting is entirely different.

The Diminishing Adjustment method for curve fitting.

As in the LeakyFit case the user must enter approximate values for the parameters to be evaluated. The program then measures the fit that would be achieved with the field data indicated in comparison to a simulation using the given parameter values. The quality of the fit is measured as a root mean square error. The very simple method used to improve the fit involves:

1/ Temporarily adjusting the values for the unknowns, in all possible combinations, and testing the quality of the fit for each combination of adjustments.

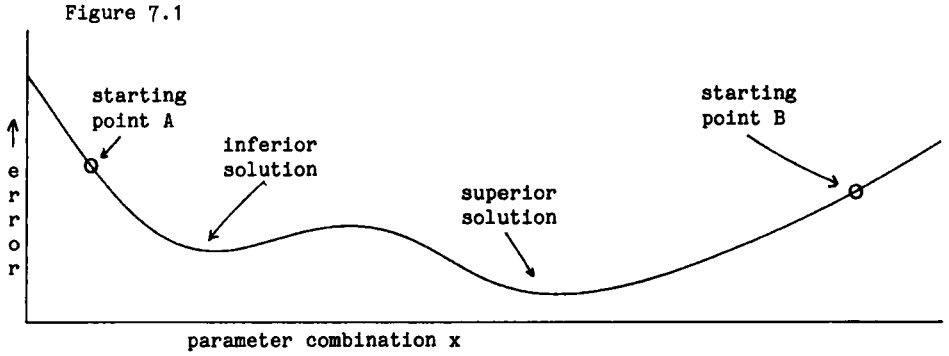
2/ Recording any combination of adjustments that gives a better fit, and altering the unknowns to the temporary values responsible for this better fit.

3/ Whenever the adjusting process fails to produce an improved fit, decreasing the magnitude of the factor used to produce the adjustments.

The three steps above are repeated until either no further significant adjustments are being made, or the root mean square error has dropped below a certain preset value. (The acceptable maximum root mean square error is set at 0.002 or 0.005 in various procedures. This may, of course, be altered by the user.) The final values for the parameters are then displayed. The details of the operational aspects of this technique are given in the Description by Procedure and Function section later in this chapter.

While this method is not as fast as that used in LeakyFit, it is more readily understandable, and is almost guaranteed to converge toward a better fit than that achieved by the original estimates. It would seem that the method may fail to find the best possible fit for a reason which is best explained by reference to an illustration.

Figure 7.1



A diagrammatic representation of a possible limitation of the Diminishing Adjustment method of curve fitting. Refer to the text.

Figure 7.1 is an attempt to illustrate, in simplified and symbolic form, a possible cause for the Diminishing Adjustment method not finding the best possible fit. (I should say that I do not know if this type of problem does actually occur.) If only one parameter is unknown then this problem is unlikely to arise because a curve produced for varying T, S, image well distance etc. will have a shape broadly similar to a that of a quadratic

function. ie. The direction of curvature will be the same in all places. When dealing with two or three unknowns, this may no longer be the case.

The two small circles represent initial estimates of parameter values. The curved line represents how possible combinations of parameter values may approach, or recede from, a perfect solution. The perfect solution would be a point where the curved line touched the x axis; in this case, as in most real cases, a perfect solution is impossible. The two dimensional figure attempts to symbolically represent a figure which should be drawn in four dimensions; imagination is required. The Diminishing Adjustment method must move toward a better solution as indicated by a reduced error. On this figure then, if the initial estimates are indicated by point A, then depending on how wide ranging is the search for a better fit, the method might move toward the inferior solution. (A very similar problem can arise in applications of Newton's Method on complex functions of one variable.)

Getting back to the specific problem to be solved by this part of the program, all combinations of T, S, and image well distance multiplied or divided by two, or left alone, are tested to see if they might produce a better fit for all the selected data than that achieved by the original estimates. Any changes yielding a better fit are accepted, and adjustments are then searched for from the new starting point. If no better fit is found, then the multiplier, two, is reduced to it's square root, and the process repeated. This operation is repeated until either an acceptable fit is obtained, or until the multiplier associated with each parameter has become very close to one.

6.2. Curve fitting for image well distance only

Here it is assumed that the values of transmissivity and storage coefficient are known, so only the distance to the image well is required. (A discharge boundary is assumed for this procedure.) The method used is much the same as that used for T, S, and image well distance, but because there is only one unknown the solution is found much more quickly.

6.3. Curve fitting for a semibounded aquifer

As with the above two curve fitting routines, this assumes an infinite (apart from the boundary), homogeneous, confined aquifer. The difference is that in this case the pumped and monitored aquifer is in contact with a second aquifer beyond the boundary. The second aquifer has the same storage coefficient as the first, but a different transmissivity. For this routine,

T and S must be provided by the user, T2 (the transmissivity beyond the boundary) and image well distance are calculated to best fit the data.

7. A CONFINED STRIP AQUIFER

The aquifer here is treated as a confined aquifer bounded on two sides by straight line, parallel boundaries. If the aquifer is actually unconfined then the same method may be applied and errors will be small so long as drawdown is small in comparison to aquifer thickness.

7.1. Finding width of the strip, given T and S

Calculation of drawdown in a strip aquifer is much slower than for an infinite or bounded aquifer because of the large number of drawdowns that must be calculated for the image wells. The speed of this (and the next) solution is still quite tolerable even with the slowest PC without a maths co-processor, because there is only one unknown. The method of Diminishing Adjustment is used here and in all the following algorithms.

For all the strip aquifer operations, both the discharging well and the piezometer are considered to be inside the strip.

7.2. Semistrip - finding width, given T, T2, and S

Here we deal with a strip aquifer bounded on two sides by a pervious boundary beyond which is considered to be a much larger aquifer having the same storage coefficient as within the strip, but different transmissivity. The best fit strip width is found, while the user provides the other three main variables.

7.3. Semistrip - finding width and T2, T and S are given

This solution is slower because two unknowns, as well as the multiple image wells of a strip aquifer, are involved. Execution time may be longer than is desirable for a basic PC without a maths co-processor, but should be acceptable on most configurations. It was developed on a 4.77 MHz PC XT with an 8087 co-processor and the time for convergence to a solution was between one half and two minutes depending upon the number of points used for the fitting, and the initial guesses.

8. AN UNCONFINED AQUIFER

Both of the following solutions use the Jacob's correction to handle the decreasing transmissivity due to partial dewatering of the unconfined aquifer.

8.1. Curve fitting for T, S, and aquifer thickness

This is a three unknown curve fitting routine which should find a use where drawdown in an unconfined aquifer becomes significant in comparison to aquifer thickness. Note that if drawdown is to be significant at a piezometer it must be much greater at the discharging well, in fact the discharging well may be close to being pumped dry before this stage is reached. Looking at this from another angle, if the curve fitting routine is to have any chance of producing a reasonably accurate answer for aquifer thickness, then drawdown must be a significant part of aquifer thickness.

8.2. Aquifer thickness, given T and S

This routine can be used as an alternative to that above. I would suggest experimenting with the two, and especially comparing the results produced by giving this method a small range of reasonable values for T and S.

9. DESCRIPTION BY PROCEDURE AND FUNCTION

This section describes the way in which each part of the program operates from the programming standpoint. It also gives the details of the mathematics of most of the solutions. If you are unsure of the validity of a solution for a particular application, then these notes should be consulted. If you are not interested in technical details, you may prefer to pass over this section.

The procedures and functions are listed in alphabetical order. The order in which they appear in the program is given in the program key lines sections. In addition to file FIRST.SEG and READSAVE.PRC, which have been explained previously, files LEAKFUNC.FUN, and BOUNDFIT.PRC are included in this program. While the procedures and functions of BOUNDFIT.PRC are dealt with in the same section as those of the main file, ANALYZE.PAS, the file itself is listed after the main file.

Some of the options use very similar algorithms for their solution. It would have been possible to reduce the volume of the code by having several solutions share code, instead of having code fully dedicated to each separate solution. The program was written the way it was because it was felt that the greater generalisation of the code would have made it's logic more difficult to follow.

In the notes that follow, as in other parts of this book, procedures and functions that are available only to other procedures (ie. are local, rather than global) are referred to as subprocedures and subfunctions.

The screen graph used to indicate the selected part (or parts) of the data is produced by procedures very similar to those employed in program PLOTWTD, these procedures will not be explained in detail again here. The differences between the two graphs are that this uses colour mode while PLOTWTD uses high resolution, and therefore the constant Right in PLOTWTD was 639 (for high resolution mode) while it has the value of 319 here. In each case Right holds the greatest plotable x value for the particular graphics mode. The graph is produced by procedure PlotData.

Adjust2 subprocedure File BOUNDFIT.PRC Line 233
File ANALYZE.PAS Line 720

Contained within procedures BoundFit3 and StripFit2

Purpose: to adjust the two unknowns, test for improved fit, and reduce the 'increment' factor for either parameter, if the value of that parameter has not been altered.

The explanation that follows applies directly to the version of this subprocedure in file BOUNDFIT.PRC, but the version in file ANALYZE.PAS is almost identical. The subprocedures AdjustAll are also very similar, with the exception that while this subprocedure deals with two variables, they deal with three.

This, and the similar subprocedures, form the heart of the Diminishing Adjustment method of curve fitting. All combinations of increased, decreased, and unaltered values for each variable are produced in turn by the first part of the double loop (beginning in line 241). Procedure Test is called to get a root mean square (RMS) error value for the current combination (line 255). If the trial RMS error is significantly less than the previous lowest RMS error, then the current values for T2 and Distance are recorded, the fact that there has been an improvement is reported, and LowestRms gets the value of CurrentRms. Finally, (beginning in line 275) if one, but not both, of the variables has not been changed then the current increment for that variable is reduced in preparation for the next iteration. 'CurrentInc' is the factor used to make the experimental adjustments.

The aim is to keep on making adjustments of the same geometrical magnitude as long as there is an improvement, but to reduce the size of the trial adjustment as soon as no improvement can be achieved.

Called by: subprocedure CalcBounded of procedure BoundFit3.

Calls: subprocedure Test.prc of procedure BoundFit3.

AdjustAll subprocedure 1/ File ANALYZE.PAS Line 206
2/ File BOUNDFIT.PRC Line 21

Slightly modified versions of this procedure are contained within UnconfinedFit, and BoundFit

Purpose: very similar to Adjust2 above, but whereas that procedure deals with two unknowns, these deal with three. The method is also very similar, the main difference being that instead of $3^2=9$ combinations of adjusted values to test, here we have $3^3=27$ combinations, so the convergence is noticeable slower.

Called by: 1/ subprocedure CalcUnconfined of procedure UnconfinedFit.
2/ subprocedure CalcBounded of procedure BoundFit.

Calls: subprocedure Test.prc.

BoundedMenu procedure File ANALYZE.PAS Line 1650

Purpose: to provide summary information for a decision on the choice of which bounded aquifer solution is required and to record the users response to that choice in the integer variable Option.

Called by: procedure MainMenu.

Calls: subprocedure DisplayMenu, and procedure RunOption.

BoundFit procedure File BOUNDFIT.PRC Line 3

Purpose: to initialise the curve fitting for T, S, and the distance to the image well, assuming a bounded confined aquifer.

BoundFit may be used as an example of several of the curve fitting procedures in this program. It consists of four main parts:

1/ The main, or controlling section of the procedure. This obtains preliminary guesses for the three unknowns and then calls subprocedure CalcBounded to control the curve fitting process.

2/ Subprocedure CalcBounded. This controls the curve fitting process. CalcBounded calls AdjustAll.

3/ Subprocedure AdjustAll does the greatest part of the work of the curve fitting including the adjustment of all guessed unknowns, and production of combinations of experimental values for the unknowns. This subprocedure calls another, Test.

4/ Subprocedure Test checks the quality of all trial value combinations by producing a vector of drawdowns, and then calling on RmsError for calculation of a root mean square error between these calculated values and the real drawdowns.

The subprocedures are explained in detail elsewhere in this section. The first operation of procedure BoundFit itself is to ask the user for guesses for the parameters who's values are to be optimised. CalcBounded is then called to produce the best fit that the method can manage, and then the calculated values are displayed by lines 130 to 132.

Called by: procedure RunOption.

Calls: subprocedure CalcBounded.

BoundFit2 procedure File BOUNDFIT.PRC Line 141

Purpose: to carry out the curve fitting for distance to image well, assuming a bounded confined aquifer.

Fixed values for T and S are obtained from the user and a preliminary assumption of 100 m for the distance to the image well is made. This preliminary value is then halved, left as is, and doubled; and the resulting fit for each alternative is tested against the previously selected field data. If one or other of the changed forms of the preliminary value is found to produce a better fit than the original, it becomes the current value, and the process is repeated. If neither change (doubling or halving) produces a better fit, then the factor used to adjust the guessed image well distance is reduced from two to the square root of two. The above steps are repeated, with the adjusting factor being reduced to the square root of it's previous value every time there is no improvement to the fit. This continues until the adjusting factor is less than 1.001, or there have been more than 25 iterations.

Called by: procedure RunOption.

Calls: functions RmsError and WellFunc.

BoundFit3 procedure File BOUNDFIT.PRC Line 213

Purpose: to initialise the curve fitting for the transmissivity beyond the boundary and the distance to the image well, assuming a semibounded confined aquifer.

Known values for transmissivity (on the near side of the boundary) and storage coefficient, followed by guessed values for T2 and image well distance, are obtained from the user. This procedure is much the same as BoundFit, described above.

Called by: procedure RunOption.

Calls: subprocedure CalcBounded.

CalcBounded subprocedure 1/ File BOUNDFIT.PRC Line 83
2/ File BOUNDFIT.PRC Line 286

Purpose: to control the curve fitting process for:

1/ finding the best fit values for T, S, and image well distance, and:

2/ finding the best fit values for T2 and image well distance.

The assumed aquifer type is confined, and bounded on one side. The first implementation of the subprocedure will be used for the description below.

An early call to Test produces a starting value for the Current root mean square error. The values of the elements of CurrentInc are set to two in line 92. As explained under Adjust2 above, CurrentInc controls the magnitude of the adjustments to the values of the unknowns in the curve fitting trials.

The loop between lines 93 and 112 calls AdjustAll (or Adjust2 in the second implementation) to search for better values in the unknowns. If the Adjust procedure finds a value in one of the unknowns that produces a better root mean square error, then that procedure will take care of selectively reducing values in the vector CurrentInc; if no better fit is found in the present iteration, then CalcBounded will reduce all the elements of the vector CurrentInc. The changed values for each of the elements of CurrentInc are displayed only if they are changed by CalcBounded.

Finally, the test at the end of the loop controls the exit from the curve fitting process. If a very good fit is found (CurrentRms<0.002) the exit will take place on that criterion. Otherwise the exit will be controlled by the values of the elements of CurrentInc. It is assumed that when the values of all elements of this vector become less than 0.01, no further significant improvement in the fit can be found.

Called by: 1/ procedure BoundFit.
2/ procedure BoundFit3.

Calls: 1/ subprocedures Test and AdjustAll.
2/ subprocedures Test and Adjust2.

CalcDd function File ANALYZE.PAS Line 96

Purpose: To calculate a drawdown for a piezometer in an infinite confined aquifer for a fixed set of parameters.

Called by: procedures SideOfStrip, StripFit, and UnconfinedFit2, as well as subprocedure Test of Procedure BoundFit and subprocedure Test of BoundFit3.

CalcTrans procedure File ANALYZE.PAS Line 1143

Purpose: To calculate the transmissivity for a marked section of the data.

The graph is drawn, the data plotted, and a section of that data is selected by the user (lines 1150 and 1151). Now, Pointer1 gets the record number of the first record selected on the graph, ie. the right-most highlighted plot. Similarly, Pointer2 gets the record number of the last record selected. If the discharge rate associated with the first chosen record is found to be zero, then recovery is assumed, and the user is asked to enter the discharge rate to be used in the calculation.

The selected data are to be passed to procedure LinReg (linear regression) in vectors Vec1 and Vec2. Because of the dimensioning of these vectors, the maximum number of elements is that can be used is 100. If it is found (line 1167) that more than 100 data are involved, then a representative set of 100 are chosen (lines 1179 to 1183).

The evaluation operates on the assumption that the transmissivity may be validly calculated from the Delta s slope of the semilogarithmic graph of the data, it is the responsibility of the user to choose appropriate data. The transmissivity is calculated by equation 7.3, given previously.

Called by: procedure ConfinedMenu.

Calls: procedures PlotData, PlotLinDdRL, PlotLogTimeRL, MarkSection, Display, LinReg, and function Response2.

CalcUnconfined subprocedure File ANALYZE.PAS Line 268

Very similar to subprocedure CalcBounded above, but where that improves the guessed values for T, S, and image well distance, this is looking for T, S, and saturated thickness.

Called by: procedure UnconfinedFit.

Calls: subprocedures Test and AdjustAll of procedure UnconfinedFit.

ConfinedMenu procedure File ANALYZE.PAS Line 1571

Purpose: to allow the user to select one or other of the options associated with an infinite confined aquifer.

Called by: procedure MainMenu.

Calls: subprocedure DisplayMenu, and procedures CalcTrans, RunStorage, and RunTheisFit.

D3 sub function File ANALYZE.PAS Line 1275

This is the derivative function of the polynomial approximation to the well equation for the case of u>=1. (The polynomial approximations to the Theis well function were given by Huntoon, 1980, and can be seen in function WellFunc of program file ANALYZE.PAS, lines 61 to 74. The constants used in the approximations are on lines 37 to 39 of the same file.) It is used in the application of Newton's Method in the solution of the well equation for storage coefficient.

Called by: procedure Storage.

D4 sub function File ANALYZE.PAS Line 1285

The derivative function of the polynomial approximation to the well equation for the case of u<1, also used for evaluation of storage coefficient.

Called by: procedure Storage.

Display procedure File ANALYZE.PAS Line 54

Purpose: to display a message on the bottom line of the screen without causing scrolling. As the screen will be in colour graphics mode when the message is displayed, the length of the message is limited to forty characters.

Called by: procedures MovePointer, MarkSection, CalcTrans, Storage, RunStorage, RunLeakage, RunOption.

DisplayMenu subprocedure File ANALYZE.PAS 1/ Line 1552
2/ Line 1575
3/ Line 1623
4/ Line 1655
5/ Line 1695
6/ Line 1740

Purpose: to display the options available at the current menu and indicate which alphanumeric keys may be pressed for selection.

There is one of these simple subprocedures for each of the menus in this program.

Called by: procedures UnconfinedMenu, ConfinedMenu, LeakyMenu, BoundedMenu, StripMenu, and MainMenu.

ExpTen function File ANALYZE.PAS Line 858

Purpose: to calculate the antilogarithm (base ten) of the passed real number.

Called by: procedure PlotLogTimeRL.

GetMaxMin procedure File ANALYZE.PAS Line 833

Purpose: to search through the data for the maximum and minimum time and drawdown so that these will be available for selection of appropriate limits to the screen graph.

Called by: the main part of program ANALYZE.

GraphMaxMin procedure File ANALYZE.PAS Line 864

Purpose: to select appropriate limits, and step size for the drawdown reference lines, for the screen graph.

The operation of the procedure will be clearer if the user has some information on the precise meaning of the variable names. Some of the variables used are:

MinDd, MaxDd; The minimum and maximum drawdowns recorded in the data file.

MinTime, MaxTime; The minimum and maximum times in the data file.

MinGraphDd, MaxGraphDd; The minimum and maximum drawdowns for the y scale of the graph.

MinLogTime, MaxLogTime; The minimum and maximum Logs (base 10) of times to be used as the limits for the time scale of the graph.

DdStep; The drawdown step to be used for the drawdown reference lines. ie. Successive drawdown reference lines will be separated by this amount.

MaxGraphDd is the first variable to be set (beginning at line 877). It begins with a value of 0.001(m) and is increased through 0.002 ... 0.009, 0.01, 0.02 ... 0.09 ... and so on until it is not less than the value of MaxDd. DdStep is set in a similar way, but here the values go through the series 0.0001, 0.0002, 0.0005, 0.001 ... etc., and the variable is fixed when it reaches a value of at least one eighth of the drawdown range to be covered by the graph. This value for DdStep will cause around five to eight drawdown reference lines to be used on the graph.

Next MinGraphDd is set (beginning at line 899) by giving it the initial unrealistically low value of 15 DdSteps below MaxGraphDd, and then raising it, one DdStep at a time, until it is just below MinDd. MinLogTime is given

the greatest integral value that is less than the log of MinTime, and MaxLogTime gets the integral value just greater than the log of MaxTime.

Called by: The main part of program ANALYZE.

Calls: function Log.

Leakage procedure File ANALYZE.PAS Line 1369

Purpose: to calculate the best fit value for leakage coefficient given T, S, and one time/drawdown/discharge rate set.

This algorithm uses a method called successive bisection to move toward a solution. It starts with two experimental values for leakage; one is too low, and the other is too high (line 1384). A mid point is taken, and a drawdown is calculated for this. If this drawdown is found to indicate that the current experimental value for leakage is too low, then the old low limit is dropped and this becomes the new low limit. Similarly, if the experimental value is too high, it will become the new high limit. The mid point is taken from the two limits, which now are closer together by a factor of two, and the process is repeated until the experimental drawdown agrees with the recorded drawdown with a negligible error (0.1mm, line 1392).

The last part of the procedure asks the user for the thickness of the confining layer so that the aquitards hydraulic conductivity can be calculated.

Called by: procedure RunLeakage.

Calls: functions LeakFunc and ReadReal.

LeakFunc function File LEAKFUNC.FUN

Purpose: to evaluate the Leaky Artesian Well Function.

Some notes on this were given in the chapter on program DRAWDOWN, otherwise refer to Cobb et. al., 1982.

Called by: procedures LeakyFit and Leakage.

LeakyFit procedure File ANALYZE.PAS Line 489

Purpose: to find the best fit transmissivity, storage coefficient, and leakage coefficient in respect to a set of time/drawdown/discharge rate values.

This procedure was adapted from a Fortran program written and described by Cobb et. al., 1982. I am not able to explain it's workings.

The original Fortran program used quite a few labels (line numbers) and GoTo statements. Labels in Pascal programs are to be avoided as they make program flow more difficult to follow. My understanding of the operation of

this function was insufficient for me to be able to remove all the labels. The labels remaining are the same as those used by Cobb et. al.

As this subroutine may be a little slow on some computers, especially if no maths co-processor is in use, there are two tests for convergence. The first test (611) makes sure that there is at least some progress towards convergence after the fifth iteration (for some reason it diverges in the first few iterations at times). The second test is the maximum allowed number of iterations. This is controlled by the value held in Itmax (line 503), and is tested in line 616. The program is at present set up to exit if 30 iterations have been carried out without an acceptable solution being reached.

The convergence criterion is the value of the constant Error (line 503). This is tested against values related to inverse leakage coefficient, storage coefficient, and transmissivity, in line 609.

Called by: procedure RunOption.

Calls: function LeakFunc.

LeakyMenu procedure File ANALYZE.PAS Line 1619

Purpose: to allow the user to select from the curve fitting options available for a leaky confined aquifer.

Called by: procedure MainMenu.

Calls: procedures RunLeakage and RunOption.

LinearY function File ANALYZE.PAS Line 916

Purpose: to calculate the y coordinate for plotting on a screen graph with a linear y scale.

Called by: procedures PlotData, PlotLinDdRL, PlotColor.

LinReg procedure File ANALYZE.PAS Line 78

Purpose: to calculate the slope and y intercept best fitting the data of the values passed to the procedure by linear regression.

The data is passed to this procedure in two vectors dimensioned to 100 real numbers each. The equations used are:

$$\text{slope} = \frac{n\,S1 - S2\,S3}{n\,S4 - S2^2}\quad \text{, and} \tag{7.5}$$

$$\text{y intercept} = \frac{S3\,S4 - S2\,S1}{n\,S4 - S2^2} \tag{7.6}$$

where n is the number of datum pairs,

S1 is the sum of all $X_i Y_i$,

S2 is the sum of all X_i,

S3 is the sum of all Y_i, and S4 is the sum of all X_i^2.

Called by: procedures TheisFit, CalcTrans.

LoadTimeLog procedure File ANALYZE.PAS Line 852

This procedure simply loads a vector with the log (base 10) of the values in the times vector.

Called by: The main part of program ANALYZE.

LogX function File ANALYZE.PAS Line 909

Purpose: to calculate the x coordinate for plotting data on a logarithmic time scale on a screen graph.

Called by: procedures PlotData, PlotLogTimeRL, PlotColor.

MainMenu procedure File ANALYZE.PAS Line 1736

Purpose: to allow the user to select the aquifer type to be assumed for the analysis.

It is this menu to which the user goes on successfully loading a file, and to which he returns before exiting from program ANALYZE.

Called by: The main part of program ANALYZE.

Calls: procedures ConfinedMenu, LeakyMenu, BoundedMenu, StripMenu, UnconfinedMenu.

MarkSection procedure File ANALYZE.PAS Line 1129

Purpose: to mark out a section of the data on the graph, rather than a number of individual data points.

The section of graph is marked out by specifying the record numbers of the end points. The point indicating the higher time limit is fixed first. The record number of the greatest time/drawdown value to be considered for the coming operation is placed in Pointer[PointNum] by the call to MovePointer2. This procedure sets PointNum to 1 (the first point), and Pointer[1] to 2 so that the second plot on the screen will be highlighted when the user commences marking out the desired section.

Setting the pointer which marks the last time/drawdown value to be considered is much the same as for the first. It is taken care of by the code from line 1136 to 1138.

Because the pointers are moved according to record numbers they will go toward the right of the screen with increasing time for the drawdown phase of a discharge test, but if t/t1 recovery data is present, then they will go toward the left with increasing time in the recovery phase (increasing time produces decreasing t/t1).

Called by: procedures CalcTrans, RunTheisFit.

Calls: procedures MovePointer, Display.

MovePointer1 procedure File ANALYZE.PAS Line 992

Purpose: To move a pointer on the screen graph to select one of a number time/drawdown values.

Commands accepted:

L key; to move to the 'left'.

O key; to indicate that the pointer's position is 'OK'.

R key; to move to the 'right'.

T key; to display the time and the discharge rate associated with the currently indicated plot.

It is the repeat until loop of lines 1000 to 1056, and the case statement beginning on line 1004 which controls the handling of these commands.

The L and R keys will move the highlighted plot to left and right respectively for drawdown data (so long as the times in that data are in ascending order). But it should be noted that pressing the L key will decrement, and the R key increment, the subscripted variable Pointer[x], which points to one of the time/drawdown/discharge rate datum sets (records). If the value in the time vector decreases with increasing record number, as it will with t/t1 recovery data, then the highlighted plot will move to the left on pressing the R key, and vice versa.

A plot may be any of three colours, yellow (colour number 3) if it is the plot corresponding to the current marker, red (colour 2) if it is a marker previously fixed or yet to be fixed, or green (colour 1) for any ordinary data plot. (The remaining colour, 0, is the background colour.) Since only the current plot can be yellow, and the background colour is not used, it is only necessary to keep track of the red and green plots. The elements of vector ColorOfPlot serve this purpose by having a value of either redd or greenn according to the colours on the screen. ('Red' and 'green' are reserved words.)

In the case of command R, the incrementing of Pointer[1], the first pointer, is conditional upon the number of records between it and the last

record (lines 1023 to 1039). If the last record is closer than five records away, then Pointer[1] is incremented by one rather than five. If the next record is the last record in the file, then the R command is ignored because the tests in both lines 1025 and 1032 are both failed.

When a pointer other than the first pointer is to be moved left or right then the fact that previous pointers have been set must be considered. (If special care was not taken, then while the location of the previously set pointers would remain in memory, their positions on the screen would be lost by their being 'run over' by the current pointer.) For this reason vector ColorOfPlot is referred to whenever it is necessary to return a plot to it's original colour.

When the T key is pressed the time (in minutes) and the discharge rate corresponding to the currently highlighted plot are displayed at the bottom of the screen. (The times are in memory in days as this is most useful unit for calculations.)

Pressing the O key fixes the position of the current plot (indicated by changing it's recorded colour to 'redd' in line 1057), and causes the procedure to be terminated.

Called by: procedures RunStorage, RunLeakage, RunOption.

Calls: procedures PlotColor and Display.

MovePointer2 procedure File ANALYZE.PAS Line 1060

Purpose: To move the first and last pointers indicating a chosen set of data on the screen graph.

This procedure has many similarities to MovePointer1 above, but while that controls any number of individual pointers, this controls only two. Also, the first pointer must leave highlighted plots behind (to the left of) it, and the second must switch off highlighting as it moves right. When pointer one moves left is switches off highlighting, and when pointer two moves left it switches on highlighting.

Consider first the code used for moving the first pointer to the right, lines 1088 to 1100. Line 1088 itself will not permit the pointer to move further right unless it is left of the last plot (Pointer[1]<NumData). To allow the pointer to be moved more quickly, lines 1090 to 1094 move it five plots at a time. The colour of the plots passed over is changed to red by the statement PlotColor(2,I). If the pointer is too close to the end of the data to move it five plots right, then it is moved only one point at a time by lines 1095 to 1099.

Lines 1101 to 1118 move the second pointer to the right. Here we must be careful of moving beyond the first pointer, rather than of moving beyond the end of the data. The two Boolean variables Stop, and Stop5, record whether it is acceptable for the plot to be moved right at all, and whether it can be moved five plots to the right, respectively (lines 1103 to 1105.

The code for moving the pointer left is similar, except that it is a little simpler because there is no need to move five plots at a time.

The code for displaying the time and discharge rate is identical to that in MovePointer1.

Called by: procedure MarkSection.

Calls: procedures PlotColor and Display.

PlotColor procedure File ANALYZE.PAS Line 976

Purpose: to cause an x shape of a given colour to be plotted at the appropriate point on the graph for the data of the indicated record number.

The value assigned to ColorNum is that of the required colour, and that assigned to I is the number of the record which is to be plotted. Functions LogX and LinearY calculate the correct screen coordinates when they are given the log (base 10) of the time, and the drawdown, respectively. Then PlotShape is called to place the x shaped plot on the graph.

Called by: procedures MovePointer, MarkSection, RunStorage, RunLeakage, RunOption.

Calls: functions LogX and LinearY, and procedure PlotShape.

PlotColor2 procedure File ANALYZE.PAS Line 984

Purpose: to cause an x shape to be plotted at the appropriate point on the graph indicating one of the calculated best fit drawdown points.

The X value is calculated to suit the time of the given pointer used to indicate a datum for the curve fitting. The Y value is that of the calculated best fit drawdown at that time.

Called by: procedures RunTheisFit and RunOption.

Calls: functions LogX and LinearY, and procedure PlotShape.

PlotData procedure File ANALYZE.PAS Line 932

Purpose: to plot all the data of the data file onto the screen graph.

This procedure is very similar to PlotColor, above, but while that plots only one data point, this plots all. Also the colour of the plot here is fixed at green, and the fact that all plots are originally green is recorded in vector ColorOfPlot.

Called by: procedures CalcTrans, RunTheisfit, RunStorage, RunLeakage, RunOption.

Calls: functions LogX and LinearY, and procedure PlotShape.

PlotLinDdRL procedure File ANALYZE.PAS Line 946

Purpose: to plot drawdown reference lines on a linear y scale.

The variable TempVal is first given the value of the minimum drawdown of the graph. A reference line is plotted at the corresponding level on the screen, and then TempVal is increased by the amount of DdStep. This process is repeated until all reference lines are drawn, with the intermediate lines being labelled at the same time. Finally the upper and lower limits of the graph are labelled.

Called by: procedure CalcTrans, RunTheisFit, RunStorage, RunLeakage, RunOption.

Calls: function LinearY.

PlotLogTimeRL procedure File ANALYZE.PAS Line 964

Purpose: to plot the time reference lines on a log x scale.

Simply causes a vertical line to be drawn from points on the x scale of the graph corresponding to integral values of log (base 10) time.

Called by: CalcTrans, RunTheisFit, RunStorage, RunLeakage, RunOption.

Calls: function LogX

PlotShape procedure File ANALYZE.PAS Line 925

Purpose: to plot an x shape of a particular colour at the screen coordinates provided.

The colour number is passed to this procedure as I.

Called by: procedures PlotColor and PlotData.

RemoveO procedure File ANALYZE.PAS Line 815

Purpose: to remove all time values of zero, or less than zero, as these would cause a run time error if an attempt was made to calculate their logarithms.

The file is searched from beginning to end, and whenever a zero time value is encountered, all following data sets are moved back one place to overwrite the unwanted record. (Normally, zero or less than zero times should only be found in the first few records.)

Called by: The main part of program ANALYZE.

Response2 function File ANALYZE.PAS Line 41

Purpose: to allow the user to enter any one of a number of legal characters which may then be acted upon.

This function is very similar to Response (described in the Preliminary section under source code file FIRST.SEG), but this variant does not write the character given by the user so as not to upset the graphics screen.

Called by: procedures CalcTrans, RunStorage.

RmsError function File ANALYZE.PAS Line 103

Purpose: to calculate the root mean square error between the current best fit data, and the real field data, for all the data points being used in the curve fitting.

The drawdown of the current best fit is in the vector TrialDd, and the field data is in the vector Drawdown. The sum of the squares of the differences between each datum pair is calculated, the average square is found, and finally the square root of this is taken.

Called by: procedures Test, UnconfinedFit2, StripFit.

RunLeakage procedure File ANALYZE.PAS Line 1407

Purpose: to do the preparatory work for calculation of the inverse leakage coefficient for a given time/drawdown/discharge rate datum set.

This procedure controls the production of the graph, and the setting of the pointer which indicates the datum point to be used for the solution. It also warns the user if there was a change of discharge rate at some time before the chosen data was recorded.

After helping the user to select the data to be used, the procedure changes back to text mode and calls procedure Leakage to have the solution calculated.

Called by: procedure LeakyMenu.

Calls: procedures PlotData, PlotLinDdRL, PlotLogTimeRL, PlotColor, MovePointer1, Display, and Leakage.

RunOption procedure File ANALYZE.PAS Line 1442

Purpose: to do the preliminary work for a number of curve fitting procedures.

RunOption. controls the production of the graph (line 1454), and the setting of the specified number of pointers indicating the data selected for use in the curve fitting (lines 1455 to 1462). Boolean variable NotFarEnough makes sure that the first pointer is shifted sufficiently far to the right to

allow room for the remaining points to the left of it (lines 1467 to 1474). The remainder of the pointers are provisionally set by lines 1476 to 1482. Lines 1483 and 1484 go through these pointers and allow the user to adjust their locations if he wants to.

Lines 1485 to 1488 detect any change of discharge rate in the data from the beginning of the test up to the time indicated by the first pointer to be set. If the discharge rate was altered during this time then the following analysis would be invalid.

After the data has been selected, it is placed in the vectors Vec1, and Vec2, (lines 1491 to 1495) in preparation to passing it to the curve fitting algorithms. Which curve fitting algorithm is chosen is controlled by previously set descriptive variables, AqType, AqExtent, etc. Variable Option indicates the number of the chosen option at the menu procedure that called RunOption.

Procedure StripFit, which is called from line 1503, can solve two curve fitting problems, for a water tight strip, of for a strip with leaky boundaries. (Both solutions produce only a best fit strip width.) Both of these are associated with the FirstSol value of the Solution variable in procedure StripMenu.

Called by: procedures UnconfinedMenu, LeakyMenu, BoundedMenu and StripMenu.

Calls: procedures PlotData, PlotLinDdRL, PlotLogTimeRL, PlotColor, MovePointer, Display, LeakyFit, StripFit, BoundFit, BoundFit2, BoundFit3, UnconfinedFit and UnconfinedFit2.

RunStorage procedure File ANALYZE.PAS Line 1329

Purpose: to do the preparatory work for the calculation of storage coefficient for a given time/drawdown/discharge rate datum set.

This procedure is very similar to RunLeakage, with the main exceptions that it is the procedure Storage that is called here rather than Leakage, and that the result is given with the screen still in graphics mode rather than first returning to text mode.

Called by: procedure ConfinedMenu.

Calls: procedures PlotData, PlotLinDdRL, PlotLogTimeRL, PlotColor, MovePointer, Display and Storage.

RunTheisFit procedure File ANALYZE.PAS Line 1202

Purpose: to allow the user to define a set of his data, and pass that data to TheisFit for fitting with a Theis type curve.

The procedure is very similar to CalcTrans, above. The main differences being that this procedure rejects data having a discharge rate of zero, and this procedure drops out of graphics mode as soon as the data are selected; to allow more detailed progress messages.

Called by: ConfinedMenu.

Calls: procedures PlotData, PlotLinDdRL, PlotLogTimeRL, MarkSection and TheisFit.

SideOfStrip function File ANALYZE.PAS Line 116

Purpose: to calculate the amount of drawdown to be expected in a piezometer in a strip aquifer, due to all the image wells on one side of that aquifer.

This function was adapted from procedure SideOfStrip in program DRAWDOWN. That procedure was simplified to produce this function. There seems no point in repeating the earlier description here.

Called by: procedure StripFit.

Calls: procedure CalcDd.

Storage procedure File ANALYZE.PAS Line 1263

Purpose: to calculate the storage coefficient from the Theis equation.

Procedure RunStorage passes values for drawdown, R (distance from pump to piezometer), discharge rate, and time to this procedure; transmissivity is entered by the user. Given these values, the well function of u may be calculated, (variable WellFunction in line 1294) although u itself is not yet known.

Newton's method is used to solve the well function for u. Functions D3 and D4 are the derived functions of the polynomial approximations for the well function for the cases u>=1 and u<1 respectively. The polynomial approximations were described by Huntoon (1980). Newton's method is described in Miller, (1981) and Grossman, (1984); it's application to the solution of the Theis equation for storage coefficient was explained in Clarke (1987).

The value of u having been obtained from the well function of u, storage coefficient is calculated in line 1320.

Called by: procedure RunStorage.

Calls: function WellFunc.

StripFit procedure File ANALYZE.PAS Line 641

Purpose: to adjust the width of a hypothetical strip aquifer and calculate a resulting set of drawdown figures, until the best possible fit is found with the selected field data.

This procedure is used for both the fully watertight strip aquifer, and the semistrip aquifer cases. The global variable AqExtent records which case is being handled, and function SideOfStrip calculates drawdowns accordingly.

A very similar curve fitting technique is used here as was used in the AdjustAll and Adjust2 procedures above. Variable Fact is given an initial value of 2 in line 60. A guess is made to the width of the strip (300m, line 653) and two variations on this guess are produced by halving it, and doubling it (division and multiplication by Fact, line 659 to 651). If the original guess is found to give the best fit, then Fact is reduced to it's own square root, and the trial repeated; otherwise the greater or lesser strip width is adopted (according to which produced the best fit), and again, the trial is repeated. This procedure is repeated until the root mean square error is reduced to below 0.01m or Fact is reduced to below 1.01.

The trial drawdowns are produced by calls to function SideOfStrip in line 668. Variable FirstTerm has the piezometer to boundary distance, and PumpToBoun has the pump to boundary distance. For simplicity it is assumed that both the piezometer and the pumped well are in the center of the strip, so FirstTerm will equal PumpToBoun. The calculation of strip aquifer drawdown is more fully covered in Chapter Two (program DRAWDOWN) and in Clarke (1987).

Called by: RunOption.

Calls: functions ReadReal, SideOfStrip, RmsError.

StripFit2 procedure File ANALYZE.PAS Line 696

Purpose: to find the best fit for both strip width and the transmissivity outside of the strip which contains the piezometer and discharging well.

This procedure is very similar to BoundFit, except that the work done by subprocedure CalcBounded in that is done here by the main part of procedure StripFit2. Of course there are also similarities to StripFit (explained above) but while that adjusts only one parameter, this procedure has to find the best fit value for two. There seams little point in giving further explanation of this procedure here.

Called by: procedure RunOption.

Calls: subprocedures Test and Adjust2, and procedure GetStripValues.

StripMenu procedure File ANALYZE.PAS Line 1690

Purpose: to allow the user to select one or other of the three strip aquifer curve fitting methods.

Called by: procedure MainMenu.

Calls: procedure RunOption.

Test subprocedure File ANALYZE.PAS Line 199
File ANALYZE.PAS Line 707
File BOUNDFIT.PRC Line 11
File BOUNDFIT.PRC Line 151
File BOUNDFIT.PRC Line 221

Purpose: variations of this procedure produce trial drawdown vectors for the current best fit, or trial, model. A root mean square error is obtained from the trial drawdown vector.

The details of the procedure vary according to the aquifer configuration in question. For example the Test in procedure UnconfinedFit calls procedure UnconfDd to have drawdown calculated with application of the inverse Jacob's correction (to convert confined drawdown to unconfined drawdown), while the Test in procedure BoundFit calls procedure CalcDd twice to get the drawdown due to the real pumped well, and the image well.

Called by: 1/ subprocedure AdjustAll of procedure UnconfinedFit,
2/ procedure StripFit2,
3/ subprocedure AdjustAll of procedure BoundFit,
4/ procedure BoundFit2,
5/ subprocedure AdjustAll of procedure BoundFit3.

Calls: 1/ functions UnconfinedDd and RmsError,
2/ functions CalcDd and RmsError,
3/ functions CalcDd and RmsError,
4/ functions WellFunc and RmsError,
5/ functions CalcDd and RmsError.

TheisFit procedure File ANALYZE.PAS Line 406

Purpose: To find the best fit Theis curve for the selected set of contiguous data.

This procedure was adapted from a Fortran program by McElwee (1980); please refer to McElwee for a detailed description of the procedure. The

procedure uses sensitivity analysis to iteratively improve a first estimate fit with the field data.

This procedure will usually converge to a solution in well under 10 iterations, but in some unusual cases it may go to 20 or so. It is set up to stop trying to find a solution if 30 iterations are reached (line 466). The convergence criteria are on lines 467 and 468; the degree of convergence is accepted when there is a change of less than one part in a thousand in both transmissivity and storage coefficient in the last iteration. All of these values could be changed to suit the reader.

Called by: RunTheisFit.

Calls: procedure LinReg and function WellFunc.

UnconfDd sub function File ANALYZE.PAS Line 184

Purpose: to calculate the amount of drawdown that might be expected in an unconfined aquifer at a given time, assuming (especially) immediate drainage.

First the drawdown that could be expected in an infinite, unbounded confined aquifer, according to the Theis equation is calculated. This is then adjusted using the inverse of Jacob's correction (Kruseman and De Ridder 1970) (line 194) to approximate the effect of the reduction in transmissivity due to the lessening of the saturated thickness by dewatering.

VALIDITY OF THE JACOB CORRECTION Please note that as this procedure assumes that the time required for drainage is negligible, it will most likely be invalid for early times. When the water table is lowered by pumping from an unconfined aquifer, some significant time is taken for the water held in the part of the aquifer above the new water table to drain down to the saturated zone. While the inverse Jacob correction might give a good approximation of unconfined drawdown from a given confined drawdown, it does not take into account the time required for an aquifer to drain. So, for times early in a discharge test when the drainage time is large compared to the reading interval, errors will be quite significant.

Called by: subprocedure Test of procedure UnconfinedFit.

Calls: function WellFunc.

UnconfinedFit procedure File ANALYZE.PAS Line 176

Purpose: to find the best fit unconfined drawdown curve to the selected field data.

Please note the comment on the validity of the Jacob correction above. This procedure uses a method very similar to that used in BoundFit (above) to

find the best fit transmissivity, specific yield, and aquifer thickness to the data selected on the screen graph by the user. The main functional difference between this and BoundFit is that while here the third unknown is thickness of aquifer, in BoundFit it was distance to the image well. Given the similarity of the two curve fitting procedures, there seems no point in explaining this one more fully.

Called by: procedure RunOption.

Calls: subprocedure CalcUnconfined.

UnconfinedFit2 procedure File ANALYZE.PAS Line 325

Purpose: to find the best fit thickness of aquifer for the given transmissivity, specific yield and selected field data.

Considering the limits to the use of the inverse Jacob correction to produce a simulation of unconfined drawdown from confined drawdown (see above), in many cases it may be best to calculate transmissivity and specific yield separately, and then have this procedure find the best fit aquifer thickness for the later test data; rather than have procedure UnconfinedFit calculate all three.

Called by: procedure UnconfinedMenu.

Calls: function ReadReal, sub function UnconfDd and procedure RmsError.

UnconfinedMenu procedure File ANALYZE.PAS Line 1548

Purpose: to allow the user to select one or other of the unconfined aquifer curve fitting options.

Called by: procedure MainMenu.

Calls: procedure RunOption.

WellFunc function File ANALYZE.PAS Line 61

Purpose: to calculate the value of the well function of u.

This function is explained in Chapter two, under program DRAWDOWN.

Called by: function CalcDd, sub function UnconfDd of procedure UnconfinedFit, sub function UnconfDd of procedure UnconfinedFit2, and procedures TheisFit and Storage.

10. REFERENCES

Bouwer, H., 1978. Groundwater Hydrology. McGraw-Hill Kogakusha Ltd. 480pp.

Clarke, D.K., 1987. Microcomputer Programs for Groundwater Studies. Developments in Water Science, 30. Elsevier, Amsterdam/Oxford/New York/Tokyo, 340pp.

Cobb, P.M., McElwee, C.D., and Butt, M.A., 1982. An automated numerical evaluation of leaky aquifer pumping test data etc. Groundwater series 6, Kansas Geo. Surv.

Freeze, R.A., and Cherry, J.A., 1979. Groundwater. Prentice-Hall Inc., New Jersey, USA. pp.604.

Grossman, S.I., 1984. Calculus. Third edition. Academic Press Inc., Orlando, Florida 32887, USA. 1300pp.

Huntoon, P.W., 1980. Computationally efficient polynomial approximations used to program the Theis equation. Groundwater, Vol. 18, No. 2, March-April.

Kruseman, G.P., and De Ridder, N.A., 1970. Analysis and Evaluation of Pumping Test Data. Int. Inst. for Land Reclamation and Improvement, Bull. 11., Wageningen, The Netherlands.

Marino, M.A., and Luthin, J.N., 1982. Seepage and Groundwater. Developments in water Science. Elsevier, Amsterdam-Oxford-New York-Tokyo. 487pp.

McElwee, C.D., 1980. The Theis Equation: Evaluation, sensitivity to storage and transmissivity, and automated fit of pump test data. Kansas Geol. Surv., Uni. of Kansas, USA.

Miller, A.R., 1981. Basic Programs for Scientists and Engineers. Sybex, Berkley, California. 318 pp.

11. KEY LINES OF FILE ANALYZE.PAS

```
  4 {#}{$I First.seg}
  6 {#}{$I Read.prc}
 41 Function Response2 {Identical to Response, except response not
    displayed}
 54 Procedure Display {Display a short message on the bottom line screen}
 61 Function WellFunc {Well function of u}
 76 {#}{$I LeakFunc.fun}
 78 Procedure LinReg {Linear regression}
 96 Function CalcDd {Calculate drawdown in confined aquifer}
103 Function RmsError {Calculate root mean square error}
116 Function SideOfStrip {Drawdown due to image wells on one side of strip}
152 Procedure GetStripValues; {Enter data on strip configuration}
174 {#----- Beginning of overlay procedures -----#}
175 {#----- Unconfined fit procedures -----#}
176 OVERLAY Procedure UnconfinedFit  {T, S, and thickness are calculated.}
184   function UnconfDd {Drawdown in unconfined aquifer, Jacob's correction}
199   procedure Test;
206   procedure AdjustAll; {Alter approximations to improve fit}
268   procedure CalcUnconfined; {Control alteration of estimates}
301 begin {# Main part of procedure UnconfinedFit}
325 OVERLAY Procedure UnconfinedFit2 {T and S given, saturated thickness
    found,
335   function UnconfDd {Drawdown in unconfined aquifer, Jacob's correction}
350 begin {# main part of procedure UnconfinedFit2}
402 {#----- End of Unconfined fit procedures -----#}
404 {#}{$I BoundFit.prc}
406 OVERLAY Procedure TheisFit {From Fortran program by Mc. Elwee}
489 OVERLAY Procedure LeakyFit {From Fortran program by Cobb, McElwee,
    Butt.}
508 begin {# Controlling part of LeakyFit}
641 OVERLAY Procedure StripFit {Strip aquifer curve fitting, width only}
696 OVERLAY Procedure StripFit2 {Leaky boundaries, width and T2 found}
707     procedure Test; {Check quality of fit}
720     procedure Adjust2; {Adjust two unknowns to improve fit}
773 begin {# main part of StripFit2}
813 {#----- End of overlay procedures -----#}
815 Procedure Remove0;   {Remove negative or zero times}
833 Procedure GetMaxMin;  {Get maximums and minimums}
847 Function Log(Num: real): real;
852 Procedure LoadTimeLog; {Load log of time vectors}
858 Function ExpTen {Inverse of log (base ten)}
864 Procedure GraphMaxMin; {Calculate maximums and minimums for both scales
908 {#----- Calculate plotting coordinates -----#}
909 Function LogX {Coodinate on logarithmic x scale}
916 Function LinearY {Coodinate on linear y scale}
922 {#----- End of calculation of plotting coordinates -----#}
924 {#----- Plot Data -----#}
925 Procedure PlotShape  {Plot an X shape}
932 Procedure PlotData; {Plot the data in the defined space}
943 {#----- End of plot data -----#}
945 {#----- Reference lines -----#}
946 Procedure PlotLinDdRL; {Plot linear drawdown reference lines}
964 Procedure PlotLogTimeRL; {Plot log of time reference lines}
973 {#----- End reference lines -----#}
975 {#----- Indicate selected data -----#}
976 Procedure PlotColor {Change a plot to a colour indicated by ColorNum}
984 Procedure PlotColor2{Plot a trial drawdown point}
```

```
 992 Procedure MovePointer1; {Move one of a number of pointers}
1060  Procedure  MovePointer2;  {Move the first or last pointer for a section}
1129 Procedure MarkSection; {Indicate a section of data on the graph}
1140 {#----- End indication of selected data -----#}
1142 {#----- Curve fitting part 2 -----#}
1143 Procedure CalcTrans; {Mark a section of the data, and calculate the
1202 Procedure RunTheisFit; {Control the calling of THEISFIT}
1263 Procedure Storage {Calculate S by Neuton's Method}
1275   function D3 {Derivitive function}
1285   function D4 {Derivitive function}
1291 begin {# Main part of Storage}
1329 Procedure RunStorage; {To calculate Storage Coefficent at a given time}
1369 Procedure Leakage {Calculate leak. coef. T and S are known}
1378 begin {# Main part of procedure Leakage}
1407 Procedure RunLeakage; {Calculate leak. coef. and K of the confining bed}
1440 {#----- End Curve fitting part 2 -----#}
1442 Procedure RunOption; {Prepare for option requiring selected data}
1547 {#----- Menu section -----#}
1548 Procedure UnconfinedMenu;
1552   procedure DisplayMenu;
1571 Procedure ConfinedMenu;
1575   procedure DisplayMenu;
1619 Procedure LeakyMenu;
1623   procedure DisplayMenu;
1650 Procedure BoundedMenu;
1655   procedure DisplayMenu;
1690 Procedure StripMenu;
1695   procedure DisplayMenu;
1736 Procedure MainMenu;
1740   procedure displayMenu;
1811 {#----- End of menu section -----#}
1812 {#----- End of procedures and functions -----#}
1814 begin {# Main part of program}
1857 end. {# of program ANALYZE}
```

12. KEY LINES OF FILE BOUNDFIT.PRC

```
  2 {#----- Procedure BoundFit -----#}
  3 OVERLAY Procedure BoundFit {T, S, and distance of image well calculated}
 11   procedure Test; {Check quality of fit}
 21   procedure AdjustAll; {Adjust all three unknowns}
 83   procedure CalcBounded; {Control the curve fitting}
117 begin {# Main part of procedure BoundFit}
141 OVERLAY Procedure BoundFit2 {T and S given, distance to image well
    found}
151   procedure Test; {Check quality of fit}
213 OVERLAY Procedure BoundFit3  {T and S are given, T2 and image well
    found}
221   procedure Test; {Check quality of fit}
233   procedure Adjust2; {Adjust the two unknowns}
286   procedure CalcBounded; {Control the curve fitting}
317 begin {# Main part of procedure BoundFit3}
342 {#----- End of the BoundFit procedures -----#}
343 {#----- End of Include file BoundFit.prc -----#}
```

13. THE DIFFERENCES BETWEEN FILES LEAKFUN2.FUN AND LEAKFUNC.FUN

The former of these is used by program DRAWDOWN for solution of the leaky artesian well function, while the latter is used here for curve fitting using the leaky artesian well function. A few slight modifications were required.

Where file LEAKFUNC.FUN has: (between the dashed lines)

```
----------------------------
  A, B, F, Wu, Um: real;

  procedure TestValues;
  begin
    TestB:=exp(-(B*A*0.25+Term[I]));
    TestU:=exp(-u);
    if Um>80 then TestUm:=0 else TestUm:=exp(-Um);
    if (TestB<=0) or (u<1) and (TestUm<=0) then
    begin
      Lc:=Lc*0.05; LCount:=LCount+1;
    end
    else
    if (u>1) and (TestU<=0) then
    begin
      Trans:=Trans*10; TransCount:=TransCount+1;
    end
  end; {sub procedure TestValues}

begin  {main part of SSURB}
----------------------------

file LEAKFUN2.FUN has:
----------------------------
  A, B, F, Wu, Um: real;

begin  {main part of SSURB}
----------------------------
```

Where file LEAKFUNC.FUN has:

```
----------------------------
      F:=A*EXP(-(u+B*A*0.25+Term[I]));
      if F<=0 then TestValues;
    end;
----------------------------

LEAKFUN2.FUN has
----------------------------
      F:=A*EXP(-(u+B*A*0.25+Term[I]));
    end;
----------------------------
```

Where file LEAKFUNC.FUN has:

```
----------------------------
      if Um>35 then F:=0 else F:=A*EXP(-(Um+B*A*0.25+Term[I]));
      if F<=0 then TestValues
    end;
----------------------------
```

LEAKFUN2.FUN has

```
----------------------------
     if Um>35 then F:=0 else F:=A*EXP(-(Um+B*A*0.25+Term[I]));
   end;
----------------------------
```

14. LISTING OF FILE ANALYZE.PAS

```
 1 Program ANALYZE_PAS;
 2 {$R+}
 3
 4 {#}{$I First.seg}
 5
 6 {#}{$I Read.prc}
 7
 8 type
 9   Solutions=(FirstSol, SecondSol);
10   SmallVec=array[1..100] of real;
11   ExtentOfAquifer=(Infinite, Bounded, Strip, SemiBounded, SemiStrip);
12   TypeOfAquifer=(Confined, Unconfined, Leaky);
13   Colours=(Greenn, Redd);
14 var
15   ConvergFail, Stop, Finished: boolean;
16   Option, Ch: Char;
17   LCount, TransCount, X, Y, X1: integer;
18   NumData, NumOfDdLogs: integer;
19   TempInt, PointNum, Pointer1, Pointer2: integer;
20   ColorOfPlot: array[1..500] of colours;
21   Pointer: array[1..30] of integer;
22   Distance, u, Lc, Trans, Trans2, StorCoef, Slope, YIntercept: real;
23   Q, MaxDd, MinDd, MinTime, MaxTime, MinLogTime, MaxLogTime: real;
24   StripWidth, MinGraphDd, MaxGraphDd, MinGraphTime, MaxGraphTime: real;
25   DdStep: real;
26   TrialDdVec, Vec1, Vec2: SmallVec;
27   TimeVec, DdVec, RateVec: MainVec;
28   TimeMin, LogTimeMin: MainVec;
29   TestType: Test; WellType: Well;
30   AqExtent: ExtentOfAquifer;
31   AqType: TypeOfAquifer;
32   Solution: Solutions;
33 const
34   Left=48; right=319; Top=0; Bottom=185;
35   LnTen=2.302585093;
36   Tolerance=10000;
37   C0=-0.57721566; C1=0.99999193; C2=-0.24991055; C3=0.05519968;
38   C4=-0.00976004; C5=0.00107857; C6=0.250621;    C7=2.334733;
39   C8=1.681534;    C9=3.330657;
40
41 Function Response2 {Identical to Response, except response not
   displayed}
42 (Targ: ShortString): ShortString;
43 var
44   Temp: byte;
45   Ch: Char;
46 begin
47   repeat
48     read(Kbd,Ch);
49     Temp:=Pos(UpCase(Ch), Targ)
50   until Temp>0;
```

```
  Response2:=UpCase(Ch);
end; {Function Response2}

Procedure Display {Display a short message on the bottom line screen}
(Long: LongString);
begin
  GotoXY(10,25); write('                                        ');
  GotoXY(10,25); write(Long);
end; {Procudure Display}

Function WellFunc {Well function of u}
(u:real): Real;
var
  u2,u3:real;
begin
  if u<=0 then u:=1e-35;
  if u>80 then WellFunc:=0
  else begin
    U2:=U*U; U3:=U2*U;
  if u<1
    then WellFunc:=-ln(u)+C0+C1*U+C2*U2+C3*U3+C4*U2*U2+C5*U2*U3
    else WellFunc:=1/(U*EXP(U))*(C6+C7*U+U2)/(C8+C9*U+U2)
  end
end; {Function WellFunc}

{#}{$I LeakFunc.fun}

Procedure LinReg {Linear regression}
(Vec1, Vec2: Smallvec; TempInt: integer);
var
  I: byte;
  Sum1, Sum2, Sum3, Sum4: real;
begin
  Sum1:=0; Sum2:=0; Sum3:=0; Sum4:=0;
  for I:=1 to TempInt do
  begin
    Sum1:=Sum1+Vec1[I]*Vec2[I];
    Sum2:=Sum2+Vec1[I];
    Sum3:=Sum3+Vec2[I];
    Sum4:=Sum4+Sqr(Vec1[I]);
  end;
  Slope:=(TempInt*Sum1-Sum2*Sum3)/(TempInt*Sum4-Sqr(Sum2));
  YIntercept:=(Sum3*Sum4-Sum2*Sum1)/(TempInt*Sum4-Sqr(Sum2));
end; {Procedure LinReg}

Function CalcDd {Calculate drawdown in confined aquifer}
(Distance, Time, Rate, Trans, StorCoef: real): real;
begin
  u:=sqr(Distance)*StorCoef/(4*Trans*Time);
  CalcDd:=Rate*WellFunc(u)/(4*Pi*Trans);
end; {Function CalcDd}

Function RmsError {Calculate root mean square error}
(Drawdown, TrialDd: SmallVec; NumTimeDdPairs: integer)
: real;
var
  I: integer;
  Sum: real;
```

```
109 begin
110   Sum:=0;
111   for I:=1 to NumTimeDdPairs do
112     Sum:=Sum+sqr(Drawdown[I]-TrialDd[I]);
113   RmsError:=sqrt(Sum/NumTimeDdPairs);
114 end; {Function RmsError}
115
116 Function SideOfStrip {Drawdown due to image wells on one side of strip}
117 (Toggle: byte; StripWidth, Time, FirstTerm, PumpToBoun, Q, AccStrip:
118   real): real;
119 var
120   ItNum, NumStrips: integer;
121   ImageDist, Temp, TempDrawdown, TempRate, ThirdTerm: real;
122 begin
123   ItNum:=1;
124   if AqExtent=SemiStrip then TempRate:=Q;
125   NumStrips:=0; TempDrawdown:=0;
126   repeat
127     begin
128       if Toggle=1 then
129       begin
130         NumStrips:=NumStrips+2; ThirdTerm:=-PumpToBoun; Toggle:=0;
131       end
132       else begin
133         ThirdTerm:=PumpToBoun; Toggle:=1
134       end; {else}
135     end; {if}
136     ImageDist:=sqrt(sqr(FirstTerm+NumStrips*StripWidth+ThirdTerm)+
137       AccStrip);
138     case AqExtent of
139       Strip: Temp:=CalcDd(ImageDist, Time, Q, Trans, StorCoef);
140       SemiStrip: begin
141         TempRate:=TempRate*-(Trans2/Trans-1)/(Trans2/trans+1);
142         Temp:=CalcDd(ImageDist, Time, TempRate, Trans, StorCoef);
143       end; {of SemiStrip case}
144     end; {of cases}
145     TempDrawdown:=TempDrawdown+Temp;
146     if ItNum mod 100=0 then writeln(ItNum:5,' image wells');
147     ItNum:=Succ(ItNum);
148   until ((abs(Temp)<abs(TempDrawdown/Tolerance)) or (abs(Temp)<0.0001));
149   SideOfStrip:=TempDrawdown;
150 end; {Function SideOfStrip}
151
152 Procedure GetStripValues; {Enter data on strip configuration}
153 begin
154   write('What is the storage coefficient? '); StorCoef:=ReadReal(1);
155   write('What is the transmissivity inside the strip? ');
156   Trans:=ReadReal(1);
157   if AqExtent=SemiStrip then
158   begin
159     if Solution=FirstSol then begin
160       write('Transmissivity outside of strip? '); Trans2:=ReadReal(1);
161     end
162     else begin
163       write('Enter your guess for transmissivity outside of strip? ');
164       Trans2:=ReadReal(1);
165     end;
166   end;
167   if Solution=SecondSol then
```

```
168   begin
169     write('Your guess for the width of the strip? ');
170     StripWidth:=ReadReal(1);
171   end;
172 end; {Procedure GetStripValues}
173
174 {#----- Beginning of overlay procedures -----#}
175 {#----- Unconfined fit procedures -----#}
176 OVERLAY Procedure UnconfinedFit  {T, S, and thickness are calculated.}
177 (TimeVec, DdVec: SmallVec; R, Q: real; NumofPointers: integer);
178 var
179   Improvement, Dewatered: Boolean;
180   I, J: integer;
181   CurrentRms, OldRms, TrialRms, u: Real;
182   Guessed, Current, Trial, CurrentInc, TrialInc: SmallVec;
183
184   function UnconfDd {Drawdown in unconfined aquifer, Jacob's correction}
185   (R, Time, T, S: real): real;
186   var
187     ConfDd: real;
188   begin
189     u:=sqr(R)*S/(4*T*Time);
190     ConfDd:=Q*WellFunc(u)/(4*Pi*T);
191     if ConfDd>=Trial[3]/2
192     then begin Dewatered:=true; UnconfDd:=Trial[3] end
193     else begin
194       UnconfDd:=Trial[3]*(1-sqrt(1-(2*ConfDd)/Trial[3]));
195       Dewatered:=false                        {Jacob's correction}
196     end; {if-then-else}
197   end; {sub function UnconfDd}
198
199   procedure Test;
200   begin
201     for I:=1 to NumOfPointers do
202       TrialDdVec[I]:=UnconfDd(R, TimeVec[I], Trial[1], Trial[2]);
203     TrialRms:=RmsError(DdVec, TrialDdVec, NumOfPointers);
204   end; {sub procedure Test}
205
206   procedure AdjustAll; {Alter approximations to improve fit}
207   var
208     Change: array[1..3] of boolean;
209     I, J, K, L: integer;
210     LowestRms: real;
211     Best: array[1..3] of real;
212   begin
213     LowestRms:=CurrentRms;
214     for I:=1 to 3 do
215     begin
216       for J:=1 to 3 do
217       begin
218         for K:=1 to 3 do
219         begin
220           case I of
221             1: Trial[1]:=Current[1]*CurrentInc[1];
222             2: Trial[1]:=Current[1];
223             3: Trial[1]:=Current[1]/CurrentInc[1];
224           end;
225           case J of
226             1: Trial[2]:=Current[2]*CurrentInc[2];
```

```
227             2: Trial[2]:=Current[2];
228             3: Trial[2]:=Current[2]/CurrentInc[2];
229           end;
230           case K of
231             1: Trial[3]:=Current[3]*CurrentInc[3];
232             2: Trial[3]:=Current[3];
233             3: Trial[3]:=Current[3]/CurrentInc[3];
234           end;
235           Test;
236           if TrialRms<LowestRms*0.999 then
237           begin
238             LowestRms:=TrialRms;
239             writeln('Improved values');
240             for L:=1 to 3 do
241             begin
242               Best[L]:=Trial[L];
243               case L of
244                 1: write('Transmissivity =');
245                 2: write('Storage coefficient =');
246                 3: write('Aquifer thickness =');
247               end; {of cases}
248               writeln(Trial[L]:13:6);
249               if Trial[L]<>Current[L]
250                 then Change[L]:=true else Change[L]:=false;
251             end; {for}
252             writeln('RMS error =',TrialRms:8:6); writeln;
253           end; {if Trial}
254         end; {for K}
255       end; {for J}
256     end; {for I}
257     if LowestRms<>CurrentRms
258     then begin
259       CurrentRms:=LowestRms;
260       for I:=1 to 3 do
261       begin
262         if Change[I]=false then CurrentInc[I]:=sqrt(CurrentInc[I]);
263         Current[I]:=Best[I];
264       end {for}
265     end; {if}
266   end; {sub procedure AdjustAll}
267
268   procedure CalcUnconfined; {Control alteration of estimates}
269   begin
270     Current[1]:=Guessed[1]; Trial[1]:=Guessed[1];
271     Current[2]:=Guessed[2]; Trial[2]:=Guessed[2];
272     Current[3]:=Guessed[3]; Trial[3]:=Guessed[3];
273     Test;
274     CurrentRms:=TrialRms;
275     writeln('CurrentRms = ',CurrentRms:8:4); OldRms:=CurrentRms;
276     for I:=1 to 3 do
277     begin CurrentInc[I]:=2; TrialInc[I]:=CurrentInc[I]; end;
278     repeat
279       AdjustAll;
280       if OldRms=CurrentRms then
281       begin
282         writeln('  Current factors: ');
283         for I:=1 to 3 do
284         begin
```

```
285           case I of
286             1: write('Transmissivity: ');
287             2: write('Storage coefficient: ');
288             3: write('Aquifer thickness: ');
289           end; {of cases}
290           CurrentInc[I]:=sqrt(CurrentInc[I]);
291           writeln(CurrentInc[I]:8:6,'  ');
292         end; {for}
293         writeln; writeln;
294       end {if}
295       else
296         OldRms:=CurrentRms;
297     until (CurrentRms<0.005) or
298       ((CurrentInc[1]<1.03) and (CurrentInc[2]<1.03) and
   (CurrentInc[3]<1.03));
299   end; {sub procedure CalcUnconfined}
300
301 begin {# Main part of procedure UnconfinedFit}
302   TextColor(Green); ClrScr;
303   writeln('Please enter guesses for the following:-');
304   writeln;
305   write('Transmissivity? '); Guessed[1]:=ReadReal(2);
306   write('Storage Coeficient? '); Guessed[2]:=ReadReal(2);
307   write('Thickness of the saturated part of the unconfined aquifer? ');
308   Guessed[3]:=ReadReal(2);
309   writeln('Calculating');
310   CalcUnconfined;
311   writeln; writeln('Best fit values are:');
312   for I:=1 to 3 do
313   begin
314     case I of
315       1: writeln('Transmissivity = ',Current[I]:9:2);
316       2: writeln('Storage coefficient = ',Current[I]:9:7);
317       3: writeln('Aquifer thickness = ',Current[I]:8:2);
318     end; {of cases}
319   end;
320   writeln('Final root-mean-square error is ',CurrentRms:9:5);
321   writeln('Press any key to continue.');
322   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
323 end; {END OVERLAY Procedure UnconfinedFit}
324
325 OVERLAY Procedure UnconfinedFit2 {T and S given, saturated thickness
   found,
326   simple unconfined aquifer.}
327 (TimeVec, DdVec: SmallVec; R, Q: real; NumofPointers: integer);
328 var
329   Dewatered, Change, Finished: boolean;
330   ItNum, Best: byte;
331   I, J: integer;
332   SatThick, Factor, u, CurrentRms, BestRms: real;
333   TempThick: array [1..3] of real;
334
335   function UnconfDd {Drawdown in unconfined aquifer, Jacob's correction}
336   (R, Time, T, S: real): real;
337   var
338     ConfDd: real;
339   begin
340     u:=sqr(R)*S/(4*T*Time);
341     ConfDd:=Q*WellFunc(u)/(4*Pi*T);
```

```
342     if ConfDd>=TempThick[J]/2
343     then begin Dewatered:=true; UnconfDd:=TempThick[J] end
344     else begin
345       UnconfDd:=TempThick[J]*(1-sqrt(1-(2*ConfDd)/TempThick[J]));
346       Dewatered:=false                        {Jacob's correction}
347     end; {if-then-else}
348   end; {sub function UnconfDd}
349
350 begin {# main part of procedure UnconfinedFit2}
351   ItNum:=0;
352   writeln('Calculation of saturated thickness, T and S given.');
353   writeln;
354   write('Transmissivity? '); Trans:=ReadReal(1);
355   write('Storage coefficient? '); StorCoef:=ReadReal(1);
356   writeln('Calculating'); writeln;
357   SatThick:=100; Factor:=2; Finished:=false; BestRms:=1000;
358   repeat
359     ItNum:=ItNum+1; Change:=false;
360     for J:=1 to 3 do
361     begin
362       for I:=1 to NumOfPointers do
363       begin
364         case J of
365           1: TempThick[J]:=SatThick/Factor;
366           2: TempThick[J]:=SatThick;
367           3: TempThick[J]:=SatThick*Factor;
368         end; {of cases}
369         TrialDdVec[I]:=UnconfDd(R, TimeVec[I], Trans, StorCoef);
370       end; {for I}
371       CurrentRms:=RmsError(DdVec, TrialDdVec, NumOfPointers);
372       if CurrentRms<BestRms then
373       begin
374         Best:=J;
375         Change:=true;
376         writeln('Improved value = ',TempThick[J]:10:2,'m, new RMS ='
377           ,CurrentRms:7:5);
378         BestRms:=CurrentRms;
379       end; {if improvement}
380     end; {for J}
381     if Change
382       then SatThick:=TempThick[Best]
383       else begin
384         Factor:=Sqrt(Factor);
385         writeln('Iteration ',ItNum,',  New factor =',Factor:8:4);
386     end; {if-then-else}
387     if Factor<1.001 then Finished:=true;
388   until Finished or (ItNum>25);
389   if Finished
390   then begin
391     writeln('Best fit aquifer thickness is ',SatThick:8:2);
392     writeln('Final root-mean-square error is ',BestRms:9:5);
393     writeln('Press any key to continue.');
394     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
395   end
396   else begin
397     writeln('No image well distance calculated - failed to converge.');
398     writeln('Press any key to continue.');
399     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
400   end;
```

```
401 end; {END OVERLAY Procedure UnconfinedFit2}
402 {#----- End of Unconfined fit procedures -----#}
403
404 {#}{$I BoundFit.prc}
405
406 OVERLAY Procedure TheisFit {From Fortran program by Mc. Elwee}
407 (TimeVec, DdVec: SmallVec; R, Q: real; NumOfPairs: integer);
408 type
409   ShortString=String[20];
410 var
411   Error: boolean;
412   Ch: char;
413   ItCount, Number, I, II: integer;
414   lnTime: real;
415   TempTrans, TempStor: real;
416   u, Drawdown, Sigma, DeltStor, DeltaTrans, Dsdt, Dsdsc: real;
417   Sum, Ssus, Ssut, Sutus, Susdif, Sutdif, Num: real;
418   Answer, Short: ShortString;
419   lnTimeVec, SP, DsdtVec, DsdscVec: SmallVec;
420 const
421   P4=12.566371;
422
423 begin
424   writeln('TheisFit has started');
425   ConvergFail:=false;
426   {Calculate the initial guess for Transmissivity and Storage.}
427   for I:=1 to NumOfPairs do lnTimeVec[I]:=ln(TimeVec[I]);
428   LinReg(lnTimeVec, DdVec, NumOfPairs);
429   Trans:=0.183*Q/Slope;
430   lnTime:=-YIntercept/Slope;
431   StorCoef:=2.25*Trans*exp(lnTime)/sqr(R);
432   ItCount:=1;
433   Repeat
434     TempTrans:=Trans; TempStor:=StorCoef;
435     for I:=1 to NumOfPairs do
436     begin
437       u:=Sqr(R)*StorCoef/(4*Trans*TimeVec[I]);
438       if u>50
439         then begin u:=50; {writeln('Value of u adjusted');} end;
440       Drawdown:=(Q/(P4*Trans))*WellFunc(u);
441       Dsdt:=(Q/(P4*Trans*Trans))*(-WellFunc(u)+EXP(-u));
442       Dsdsc:=-(Q/(P4*Trans*StorCoef))*EXP(-u);
443       Sp[I]:=Drawdown; DsdtVec[I]:=Dsdt; DsdscVec[I]:=Dsdsc;
444     end;
445     Ssus:=0; Ssut:=0; Sutus:=0; Susdif:=0; Sutdif:=0;
446     for I:=1 to NumOfPairs do
447     begin
448       Ssus:=DsdscVec[I]*DsdscVec[I]+Ssus;
449       Ssut:=DsdtVec[I]*DsdtVec[I]+Ssut;
450       Sutus:=DsdscVec[I]*DsdtVec[I]+Sutus;
451       Susdif:=DsdscVec[I]*(DdVec[I]-Sp[I])+Susdif;
452       Sutdif:=DsdtVec[I]*(DdVec[I]-Sp[I])+Sutdif;
453     end;
454     DeltStor:=(Ssut*Susdif-Sutus*Sutdif)/(Ssus*Ssut-Sutus*Sutus);
455     if DeltStor<(-0.95*StorCoef) then DeltStor:=-0.95*StorCoef;
456     if DeltStor>(2*StorCoef) then DeltStor:=2*StorCoef;
457     StorCoef:=StorCoef+DeltStor;
458     if StorCoef>1 then StorCoef:=1;
459     DeltaTrans:=(Sutdif-DeltStor*Sutus)/Ssut;
```

```
460     if DeltaTrans<(-0.95*Trans) then DeltaTrans:=-0.95*Trans;
461     if DeltaTrans>(2*Trans) then DeltaTrans:=2*Trans;
462     Trans:=Trans+DeltaTrans;
463     writeln('Iteration ',ItCount:3);
464     writeln('Transmissivity =',Trans:10:2,
465       ', Storage coefficient =',StorCoef:10:8);
466     ItCount:=ItCount+1; if ItCount>30 then ConvergFail:=true;
467   until ((abs((TempTrans-Trans)/Trans)<0.001)
468     and (abs((TempStor-StorCoef)/StorCoef)<0.001))
469     or ConvergFail;
470   if ConvergFail=false then
471   begin
472     write('Final values: Transmissivity =',Trans:10:2);
473     writeln(', Storage coeff. =',StorCoef:10:8);
474     {Calculate the root-mean-square error in drawdown after best fit}
475     Sum:=0;
476     for I:=1 to NumOfPairs do
477     begin
478       Sum:=Sum+(Sp[I]-DdVec[I])*(Sp[I]-DdVec[I]);
479       Sigma:=Sqrt(Sum/NumOfPairs);
480     end;
481     writeln('Root-mean-square error =',Sigma:8:6);
482   end {if ConvergFail=false}
483   else
484     writeln('TheisFit failed to converge.');
485   writeln('Press any key to continue.');
486   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
487 end; {END OVERLAY Procedure TheisFit}
488
489 OVERLAY Procedure LeakyFit {From Fortran program by Cobb, McElwee,
    Butt.}
490 (TimeVec, DdVec: SmallVec; R, Q: real; NumDatPairs: integer);
491 var
492   Ch: Char;
493   I, J, K, N, Iterat: integer;
494   Temp1, Temp2, Temp3, Temp4: real;
495   Sigma, Sg, Tcl, Z, RjSqr: real;
496   Sdels2, Suscds, surcds, sukbds,sukbus, surcuk, Suscur, Surc2: real;
497   Susc2, Sukb2: real;
498   U11, U12, U13, U14, U21, U22, U23, U24, U31, U32, U33, U34: real;
499   Rb, WPlus, WMinus, U, Dels, Ak, Urc, Usc, Ukb, Rbm, Rbp: real;
500   X: array[1..3] of real;
501   Sgs: SmallVec; StanDev: array[1..30] of real;
502 const
503   Error=0.004; Decres=-0.2; Incres=0.5;
504   Itmax=30;
505 label
506   100, 120, 220, 230, 240, 260, 290;
507
508 begin {# Controlling part of LeakyFit}
509   ConvergFail:=false;
510   N:=3; Iterat:=0;
511   write('Thickness of aquitard? '); Tcl:=ReadReal(2);
512   writeln('      Guesses for aq. parameters');
513   write('Transmissivity? '); Trans:=ReadReal(1);
514   write('Storage coefficient? '); StorCoef:=ReadReal(1);
515   write('Inverse leakage coefficient? '); Lc:=ReadReal(2);
516   LCount:=0; TransCount:=0; writeln('LeakyFit has commenced');
517   100:
```

```
518   if (TransCount>0) or (LCount>0) then Iterat:=Iterat-1;
519   LCount:=0; TransCount:=0;
520     {Initialize iterations}
521   120:
522   Iterat:=Iterat+1;
523     {Zero out summations}
524   Sdels2:=0; Suscds:=0; Surcds:=0; Sukbds:=0; Sukbus:=0;
525   Surcuk:=0; Suscur:=0; Surc2:=0;  Susc2:=0;  Sukb2:=0;
526     {Zero out matrix}
527   for K:=1 to N do X[K]:=0;
528     {Compute Rb}
529   Rb:=R*Lc;
530   for I:=1 to NumDatPairs do
531   begin
532     {Compute U}
533     U:=Sqr(R)*StorCoef/(4*Trans*TimeVec[I]);
534     {Compute theoretical drawdown}
535     Sg:=(Q/(4*Pi*Trans))*LeakFunc(U,Rb);
536     if (LCount<>0) or (TransCount<>0) then goto 100;
537     Sgs[I]:=Sg;
538     {Compute difference between theoretical and experimental drawdown}
539     Dels:=DdVec[I]-Sg;
540     If abs(Dels)<1E-3 then Dels:=0;
541     Sdels2:=Sdels2+Sqr(Dels);
542     {Compute dummy coefficient}
543     Z:=U+Sqr(Rb)/(4*U); RjSqr:=Sqr(R);
544     if Z>55 then
545     begin
546       Z:=55; writeln('The value of Z has been adjusted');
547     end;
548     {Compute sensitivity coefficient and summations}
549     Usc:=-(Q/(4*Pi*Trans)*(1/U)*RjSqr/(4*Trans*TimeVec[I]))*Exp(-Z);
550     Ukb:=-Sg/Trans+(Q/(4*Pi*Trans))*
551       ((RjSqr*StorCoef)/(4*Sqr(Trans)*TimeVec[I]))*(1/U)*exp(-Z);
552     if (Usc=0) or (Ukb=0) then Iterat:=Iterat-1;
553     if (Usc=0) or (Ukb=0) then goto 230;
554     Rbm:=R*0.99*Lc;
555     Rbp:=R*1.01*Lc;
556     WPlus:=LeakFunc(U,Rbp);
557     if (LCount<>0) or (TransCount<>0) then goto 100;
558     WMinus:=LeakFunc(U,Rbm);
559     if (LCount<>0) or (TransCount<>0) then goto 100;
560     Urc:=Q/(4*Pi*Trans)*(WPlus-WMinus)/(0.02*Lc);
561     if Urc=0 then Iterat:=Iterat-1;
562     if Urc=0 then goto 230;
563     Sukb2:=Sukb2+Sqr(Ukb); Susc2:=Susc2+Sqr(Usc);
564     Surc2:=Surc2+Sqr(Urc); Suscur:=Suscur+Usc*Urc;
565     Sukbus:=Sukbus+Ukb*Usc; Surcuk:=Surcuk+Urc*Ukb;
566     Suscds:=Suscds+Usc*Dels; Surcds:=Surcds+Urc*Dels;
567     Sukbds:=Sukbds+Ukb*Dels;
568   end; {for I:=1 to NumDatPairs}
569   220:  {Compute matrix to be solved for sensitivity deltas}
570   U11:=Surc2*Lc; U12:=Suscur*StorCoef; U13:=Surcuk*Trans; U14:=Surcds;
571   U21:=Suscur*Lc; U22:=Susc2*StorCoef; U23:=Sukbus*Trans; U24:=Suscds;
572   U31:=Surcuk*Lc; U32:=Sukbus*StorCoef; U33:=Sukb2*Trans; U34:=Sukbds;
573   {Solve matrix by direct Gauss elimination}
574   Temp1:=(U14*U21-U24*U11)*(U12*U31-U32*U11);
575   Temp2:=(U14*U31-U34*U11)*(U12*U21-U22*U11);
```

```
  Temp3:=(U13*U21-U23*U11)*(U12*U31-U32*U11);
  Temp4:=(U13*U31-U33*U11)*(U12*U21-U22*U11);
  X[3]:=(Temp1-Temp2)/(Temp3-Temp4);
  Temp1:=(U14*U21-U24*U11)-(U13*U21-U23*U11)*X[3];
  X[2]:=Temp1/(U12*U21-U22*U11);
  X[1]:=(U14-U13*X[3]-U12*X[2])/U11;
  goto 240;
  230:
  {Update initial guess values of StorCoef, Trans, and Lc}
  X[1]:=Incres; Lc:=Lc*(1+X[1]);
  X[2]:=Incres; StorCoef:=StorCoef*(1+X[2]); if StorCoef>=1 then
StorCoef:=1;
  X[3]:=Incres; Trans:=Trans*(1+X[3]);
  goto 260;
  240:
  {Compute standard deviation for each iteration}
  Sigma:=Sqrt(Sdels2/NumDatPairs);
  StanDev[Iterat]:=Sigma;
  writeln(Iterat:3,' iterations, standard deviation =',Sigma:10:6);
  writeln('Transmissivity =',Trans:10:2);
  writeln('Storage coefficient =',StorCoef:9:7);
  writeln('Inverse leakage coefficient =',Lc:10:7);
  {Update coefficients}
  260:
  if X[1]<Decres then X[1]:=Decres;
  if X[1]>Incres then X[1]:=Incres;
  Lc:=Lc*(1+X[1]);
  if X[2]<Decres then X[2]:=Decres;
  if X[2]>Incres then X[2]:=Incres;
  StorCoef:=StorCoef*(1+X[2]);
  if X[3]<Decres then X[3]:=Decres;
  if X[3]>Incres then X[3]:=Incres;
  Trans:=Trans*(1+X[3]);
  {Check for Delta convergence}
  if (Abs(X[1])<=Error) and (abs(X[2])<=Error) and (abs(X[3])<=Error)
    then goto 290;
  if Iterat>5 then if Sigma*1.05>StanDev[Iterat-3]
  then begin
    ConvergFail:=true;
    goto 290;
  end;
  if Iterat>=Itmax
  then begin
    ConvergFail:=true;
    goto 290;
  end;
  goto 120;
  290:
  if ConvergFail=true
  then begin
    writeln('LeakyFit failed to converge');
  end
  else begin
    writeln; writeln('LeakyFit achieved convergence.');
    writeln('Results were:-');
    {Write out final program status}
    writeln('Transmissivity =',Trans:10:2);
    writeln('Storage coefficient =',StorCoef:9:7);
    writeln('Inverse leakage coefficient =',Lc:10:7);
```

```
634     Ak:=Trans*Tcl*Sqr(Lc);
635     writeln('Hydraulic conductivity of the confining layer =',Ak:10:7);
636   end; {if-then-else}
637   writeln('Press any key to continue.');
638   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
639 end; {OVERLAY Procedure LeakyFit}
640
641 OVERLAY Procedure StripFit {Strip aquifer curve fitting, width only}
642 (Time, Drawdown: SmallVec; Distance, Q: real; NumTimeDdPairs: integer);
643 var
644   Finished: boolean;
645   I, J, Improve, Toggle: integer;
646   TempWidth, u, TempDd, SmallestRms: real;
647   Fact, FirstTerm, PumpToBoun, ThisRmsError, AccStrip: real;
648   Short: ShortString;
649 begin
650   Fact:=2; Improve:=2;
651   GetStripValues;
652   writeln('Calculating'); writeln;
653   StripWidth:=300; {first guess}
654   SmallestRms:=1e30;
655   repeat
656     for J:=1 to 3 do
657     begin
658       case J of
659         1: TempWidth:=StripWidth/Fact;
660         2: TempWidth:=StripWidth;
661         3: TempWidth:=StripWidth*Fact;
662       end; {of cases}
663       AccStrip:=sqr(Distance);   {Square of distance component across
    aquifer}
664       for I:=1 to NumTimeDdPairs do
665       begin
666         TrialDdVec[I]:=CalcDd(Distance, Time[I], Q, Trans, StorCoef);
667         FirstTerm:=TempWidth/2; PumpToBoun:=TempWidth/2; Toggle:=0;
668         TrialDdVec[I]:=TrialDdVec[I]+(SideOfStrip(Toggle, TempWidth,
    Time[I],
669           FirstTerm, PumpToBoun, Q, AccStrip))*2;
670               {Calculate drawdown due to image wells on both sides
671         As the strip is assumed to be symetrical, drawdown due to both
    sides
672         is equal.}
673       end; {for I}
674       ThisRmsError:=RmsError(Drawdown, TrialDdVec, NumTimeDdPairs);
675       if ThisRmsError<SmallestRms then
676       begin
677         SmallestRms:=ThisRmsError;
678         Improve:=J
679       end;
680     end; {for I}
681     case Improve of
682       1: StripWidth:=StripWidth/Fact;
683       2: Fact:=sqrt(Fact);
684       3: StripWidth:=StripWidth*Fact;
685     end; {cases}
686     writeln('Factor =',Fact:10:8,', Stripwidth =',StripWidth:8:3,
687       ', Rms error =',SmallestRms:10:6);
688     Improve:=2;
689   until (ThisRmsError<0.01) or (Fact<1.01);
```

```
690   writeln('Best fit is a strip width of ',StripWidth:10:2);
691   writeln('Final root-mean-square error was ',ThisRmsError:10:8);
692   writeln('Press any key to continue.');
693   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
694 end; {OVERLAY Procedure StripFit}
695
696 OVERLAY Procedure StripFit2 {Leaky boundaries, width and T2 found}
697 (Time, Drawdown: SmallVec; Distance, Q: real; NumTimeDdPairs: integer);
698 var
699   Finished: boolean;
700   I, J, Improve, Toggle: integer;
701   u, TempDd: real;
702   CurrentRms, OldRms, TrialRms, SmallestRms: real;
703   Fact, FirstTerm, PumpToBoun, AccStrip: real;
704   Guessed, Current, Trial, CurrentInc, TrialInc: SmallVec;
705   Short: ShortString;
706
707     procedure Test; {Check quality of fit}
708     begin
709       AccStrip:=sqr(Distance);   {Square of distance across aquifer}
710       for I:=1 to NumTimeDdPairs do
711       begin
712         TrialDdVec[I]:=CalcDd(Distance, Time[I], Q, Trans, StorCoef);
713         FirstTerm:=Trial[1]/2; PumpToBoun:=Trial[1]/2; Toggle:=0;
714         TrialDdVec[I]:=TrialDdVec[I]+(SideOfStrip(Toggle, Trial[1],
    Time[I],
715           FirstTerm, PumpToBoun, Q, AccStrip))*2;
716       end; {for I}
717       TrialRms:=RmsError(Drawdown, TrialDdVec, NumTimeDdPairs);
718     end; {sub procedure Test}
719
720     procedure Adjust2; {Adjust two unknowns to improve fit}
721     var
722       Change: array[1..2] of boolean;
723       I, J, K, L: integer;
724       LowestRms: real;
725       Best: array[1..3] of real;
726     begin
727       LowestRms:=CurrentRms;
728       for I:=1 to 3 do
729       begin
730         for J:=1 to 3 do
731         begin
732           case I of
733             1: Trial[1]:=Current[1]*CurrentInc[1];
734             2: Trial[1]:=Current[1];
735             3: Trial[1]:=Current[1]/CurrentInc[1];
736           end;
737           case J of
738             1: Trial[2]:=Current[2]*CurrentInc[2];
739             2: Trial[2]:=Current[2];
740             3: Trial[2]:=Current[2]/CurrentInc[2];
741           end;
742           Trans2:=Trial[2]; Test;
743           if TrialRms<LowestRms*0.999 then
744           begin
745             LowestRms:=TrialRms;
746             writeln('Improved values');
747             for L:=1 to 2 do
```

```
748             begin
749               Best[L]:=Trial[L];
750               case L of
751                 1: write('Width of strip aquifer =');
752                 2: write('Transmissivity outside of strip =');
753               end; {of cases}
754               writeln(Trial[L]:13:6);
755               if Trial[L]<>Current[L] then Change[L]:=true
756                 else Change[L]:=false;
757             end; {for L}
758             writeln('RMS error =',TrialRms:8:6); writeln;
759           end; {if Trial}
760         end; {for J}
761       end; {for I}
762       if LowestRms<>CurrentRms
763       then begin
764         CurrentRms:=LowestRms;
765         for I:=1 to 2 do
766         begin
767           if Change[I]=false then CurrentInc[I]:=sqrt(CurrentInc[I]);
768           Current[I]:=Best[I];
769         end {for}
770       end; {if}
771     end; {sub procedure Adjust2}
772
773 begin {# main part of StripFit2}
774   Fact:=2; Improve:=2;
775   GetStripValues;
776   writeln('Calculating'); writeln;
777     Guessed[1]:=StripWidth; Guessed[2]:=Trans2;
778     Current[1]:=Guessed[1]; Trial[1]:=Guessed[1];
779     Current[2]:=Guessed[2]; Trial[2]:=Guessed[2];
780     Test;
781     CurrentRms:=TrialRms;
782     writeln('CurrentRms = ',CurrentRms:8:4); OldRms:=CurrentRms;
783     for I:=1 to 2 do
784     begin CurrentInc[I]:=2; TrialInc[I]:=CurrentInc[I]; end;
785     repeat
786       Adjust2;
787       if OldRms=CurrentRms then
788       begin
789         writeln('  Current factors: ');
790         for I:=1 to 2 do
791         begin
792           case I of
793             1: write('Strip width: ');
794             2: write('Transmissivity outside of strip: ');
795           end; {of cases}
796           CurrentInc[I]:=sqrt(CurrentInc[I]);
797           writeln(CurrentInc[I]:8:6,'  ');
798         end; {for}
799         writeln; writeln;
800       end {if}
801       else
802         OldRms:=CurrentRms;
803     until (CurrentRms<0.005) or
804       ((CurrentInc[1]<1.03) and (CurrentInc[2]<1.03));
805     StripWidth:=Current[1]; Trans2:=Current[2];
806   writeln('Best fit is a strip width of ',StripWidth:10:2);
```

```
807   writeln('with transmissivity outside of strip = ',Trans2:10:2);
808   writeln('Final root-mean-square error was ',CurrentRms:10:8);
809   Trial[1]:=StripWidth; Test; {Place values in TrialDdVec for plotting}
810   writeln('Press any key to continue.');
811   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
812 end; {OVERLAY Procedure StripFit2}
813 {#----- End of overlay procedures -----#}
814
815 Procedure Remove0;   {Remove negative or zero times}
816 begin
817   J:=1;
818   repeat
819     if TimeMin[J]<=0
820     then begin
821       for I:=J+1 to NumData do
822       begin
823         TimeMin[I-1]:=TimeMin[I]; TimeVec[I-1]:=TimeVec[I];
824         DdVec[I-1]:=DdVec[I];
825         RateVec[I-1]:=RateVec[I];
826       end; {for I}
827       NumData:=NumData-1; J:=J-1;
828     end; {if}
829     J:=J+1;
830   until J>NumData;
831 end; {Procedure Remove0}
832
833 Procedure GetMaxMin;  {Get maximums and minimums}
834 var I: integer;
835 begin
836   MaxTime:=-1e36; MinTime:=1e36;
837   MaxDd:=-1e30;MinDd:=1e30;
838   for I:=1 to NumData do
839   begin
840     if TimeMin[I]>MaxTime then MaxTime:= TimeMin[I];
841     if TimeMin[I]<MinTime then MinTime:= TimeMin[I];
842     if DdVec[I]>MaxDd then MaxDd:= DdVec[I];
843     if DdVec[I]<MinDd then MinDd:= DdVec[I];
844   end; {for I}
845 end; {Procedure GetMaxMin}
846
847 Function Log(Num: real): real;
848 begin
849   Log:=Ln(Num)/LnTen;
850 end; {Function Log}
851
852 Procedure LoadTimeLog; {Load log of time vectors}
853 begin
854   for I:=1 to NumData do
855   LogTimeMin[I]:= Log(TimeMin[I]);
856 end; {Procedure LoadTimeLog}
857
858 Function ExpTen {Inverse of log (base ten)}
859 (Num: real): real;
860 begin
861   ExpTen:=Exp(Num*LnTen);
862 end; {Function ExpTen}
863
864 Procedure GraphMaxMin; {Calculate maximums and minimums for both scales
```

```
865 of the graph.}
866 var
867   I: integer;
868   Multiplier: real;
869 const
870   GraphStep: array[1..3] of real =(1,2,5);
871   GraphDd: array[1..10] of real =(1.,1.5,2.,3.,4.,5.,6.,7.,8.,9.0);
872    {Calculate maximums and minumums for the drawdown scale of the graph,
873     and calculate the drawdown step for the graph's ref. lines.}
874 begin
875   Multiplier:=0.001;
876                 {Calculate the maximum drawdown for the graph}
877   MaxGraphDd:=GraphDd[1]*Multiplier; I:=1;
878   while MaxGraphDd<MaxDd do
879   begin
880     I:=I+1; if I=11
881     then begin
882       Multiplier:=Multiplier*10; I:=1;
883     end;
884     MaxGraphDd:=GraphDd[I]*Multiplier;
885   end;
886         {Calculate the step for the drawdown ref. lines}
887   Multiplier:=0.0001; DdStep:=GraphStep[1]*Multiplier; I:=1;
888   while (DdStep*8)<(MaxGraphDd-MinDd) do
889   begin
890     if I<3 then I:=I+1
891     else begin I:=1; Multiplier:=Multiplier*10 end;
892     DdStep:=GraphStep[I]*Multiplier
893   end;
894   if (DdStep*8)<(MaxGraphDd) then
895     if I<3
896       then DdStep:=GraphStep[I+1]*Multiplier
897       else DdStep:=GraphStep[1]*Multiplier*10;
898          {Calculate the minimum drawdown for the graph}
899   MinGraphDd:=MaxGraphDd-15*DdStep;
900   while MinGraphDd+DdStep<=MinDd do MinGraphDd:=MinGraphDd+DdStep;
901   if (MinGraphDd<0) and (MinDd>=0) then MinGraphDd:=0;
902      {Calculate maximums and minimums for the time scale of the graph}
903   MaxLogTime:=Int(Log(MaxTime)+0.99);
904   MinLogTime:=Int(Log(MinTime*1.00001));
905   if MinTime<0.99999 then MinLogTime:=MinLogTime-1;
906 end; {Procedure GraphMaxMin}
907
908 {#----- Calculate plotting coordinates -----#}
909 Function LogX {Coodinate on logarithmic x scale}
910 (Num: real): integer; var Temp: real;
911 begin
912   Temp:=(Num-MinLogTime)/(MaxLogTime-MinLogTime);
913   LogX:=round(Left+(Right-Left)*Temp);
914 end; {Function LogX}
915
916 Function LinearY {Coodinate on linear y scale}
917 (Num: real): integer;
918 begin
919   LinearY:=round(Top+(Bottom-Top)*(Num-MinGraphDd)/
920     (MaxGraphDd-MinGraphDd));
921 end; {Function LinearY}
922 {#----- End of calculation of plotting coordinates -----#}
923
```

```
924 {#----- Plot Data -----#}
925 Procedure PlotShape  {Plot an X shape}
926 (X, Y, I: integer);
927 begin
928   Plot(X,Y,I); Plot(X+1,Y+1,I); Plot(X-1,Y+1,I);
929   Plot(X+1,Y-1,I); Plot(X-1,Y-1,I);
930 end; {Procedure PlotShape}
931
932 Procedure PlotData; {Plot the data in the defined space}
933 begin
934   for I:=1 to NumData do
935   begin
936     if (TimeMin[I]>=MinTime) and (TimeMin[I]<=MaxTime)
937     then begin
938       X:=LogX(LogTimeMin[I]); Y:=LinearY(DdVec[I]);
939       PlotShape(X,Y,1); ColorOfPlot[I]:=Greenn;
940     end; {if Time}
941   end; {for I}
942 end; {Procedure PlotData}
943 {#----- End of plot data -----#}
944
945 {#----- Reference lines -----#}
946 Procedure PlotLinDdRL; {Plot linear drawdown reference lines}
947 begin
948   TempVal:=MinGraphDd;
949   While TempVal<=MaxGraphDd do
950   begin
951     Y:=LinearY(TempVal);
952     Draw(Left,Y,Right,Y,1);
953     if TempVal>MinGraphDd
954     then begin
955       GotoXY(1,(Y div 8)+1); write(TempVal:5:3);
956     end; {then}
957     TempVal:= TempVal+DdStep;
958   end; {while}
959   GotoXY(1,24); write(MaxGraphDd:5:3);
960   GotoXY(1,1); write(MinGraphDd:5:3);
961   Draw(Left,Bottom,Right,Bottom,1);
962 end; {Procedure PlotLinDdRL}
963
964 Procedure PlotLogTimeRL; {Plot log of time reference lines}
965 begin
966   for I:=round(MinLogTime) to round(MaxLogTime) do
967   begin
968     X:=LogX(I);
969     Draw(X,Top,X,Bottom,1);
970   end;
971   GotoXY(1,25); write(ExpTen(MinLogTime):7:2);
972 end; {Procedure PlotLogTimeRL}
973 {#----- End reference lines -----#}
974
975 {#----- Indicate selected data -----#}
976 Procedure PlotColor {Change a plot to a colour indicated by ColorNum}
977 (ColorNum: byte; I: integer);
978 begin
979   X:=LogX(LogTimeMin[I]);
980   Y:=LinearY(DdVec[I]);
981   PlotShape(X,Y,ColorNum);
982 end; {PlotColor}
```

```
 983
 984 Procedure PlotColor2{Plot a trial drawdown point}
 985 (ColorNum: byte; I, J: integer);
 986 begin
 987   X:=LogX(LogTimeMin[Pointer[I]]);
 988   Y:=LinearY(TrialDdVec[J]);
 989   PlotShape(X,Y,ColorNum);
 990 end; {PlotColor2}
 991
 992 Procedure MovePointer1; {Move one of a number of pointers}
 993 var
 994   Stop5: Boolean;
 995   Cmd: char;
 996 begin
 997   Display('L=left, R=right, O=OK, T=Time');
 998   ColorOfPlot[Pointer[PointNum]]:=Greenn;
 999   PlotColor(3,Pointer[PointNum]);
1000   repeat
1001     repeat
1002       Read(kbd,Cmd); Cmd:=UpCase(Cmd);
1003     until (Cmd='O') or (Cmd='L') or (Cmd='R') or (Cmd='T');
1004     case Cmd of
1005     'L': begin
1006       if Pointer[PointNum]>1 then
1007       begin
1008         if PointNum=1
1009         then begin
1010           ColorOfPlot[Pointer[1]]:=Greenn;
1011           PlotColor(1,Pointer[1]);
1012           Pointer[1]:=Pointer[1]-1;
1013         end
1014         else begin
1015           if ColorOfPlot[Pointer[PointNum]]=Greenn
1016             then PlotColor(1,Pointer[PointNum])
1017             else PlotColor(2,Pointer[PointNum]);
1018           Pointer[PointNum]:=Pointer[PointNum]-1;
1019         end; {else}
1020       end; {if Pointer[PointNum]>1}
1021     end; {case 'L'}
1022     'R': begin
1023       if PointNum=1 then
1024       begin
1025         if Pointer[1]<NumData-4
1026         then begin
1027           ColorOfPlot[Pointer[1]]:=Greenn;
1028           PlotColor(1,Pointer[1]);
1029           Pointer[1]:=Pointer[1]+5;
1030         end
1031         else begin
1032           if Pointer[1]<NumData
1033           then begin
1034             ColorOfPlot[Pointer[1]]:=Greenn;
1035             PlotColor(1,Pointer[1]);
1036             Pointer[1]:=Pointer[1]+1;
1037           end; {if}
1038         end; {else}
1039       end; {PointNum=1}
1040       if (PointNum<>1) and (Pointer[PointNum]<NumData) then
1041       begin
```

```
1042          if ColorOfPlot[Pointer[PointNum]]=Greenn
1043            then PlotColor(1,Pointer[PointNum])
1044            else PlotColor(2,Pointer[PointNum]);
1045          Pointer[PointNum]:=Pointer[PointNum]+1;
1046          PlotColor(2,Pointer[PointNum])
1047        end; {PointNum<>1}
1048      end; {case 'R'}
1049      'T': begin
1050        Str(TimeVec[Pointer[PointNum]]*1440:7:0,Short);
1051        Str(RateVec[Pointer[PointNum]]:5:0,Long);
1052        Display('Time='+Short+'min  Q='+Long);
1053        end;
1054      end; {of cases}
1055      PlotColor(3,Pointer[PointNum]);
1056    until Cmd='0';
1057    ColorOfPlot[Pointer[PointNum]]:=Redd;
1058  end; {Procedure MovePointer1}
1059
1060  Procedure MovePointer2; {Move the first or last pointer for a section}
1061  var
1062    Stop5: boolean;
1063    Cmd: char;
1064  begin
1065    Display('L=left, R=right, 0=OK, T=Time');
1066    repeat
1067      repeat
1068        Read(kbd,Cmd); Cmd:=UpCase(Cmd);
1069      until (Cmd='0') or (Cmd='L') or (Cmd='R') or (Cmd='T');
1070      case Cmd of
1071        'L': begin
1072          if Pointer[PointNum]>1 then
1073          begin
1074            if (PointNum=1) and (Pointer[1]>2) then
1075            begin
1076              PlotColor(1,Pointer[1]);
1077              Pointer[1]:=Pointer[1]-1;
1078              PlotColor(2,Pointer[1]);
1079            end;
1080            if PointNum=2 then
1081            begin
1082              Pointer[2]:=Pointer[2]-1;
1083              PlotColor(2,Pointer[2]);
1084            end;
1085          end; {if Pointer[PointNum]>1}
1086        end; {case 'L'}
1087        'R': begin
1088          if (PointNum=1) and (Pointer[1]<NumData) then
1089          begin
1090            if Pointer[1]<NumData-5
1091            then begin
1092              for I:=Pointer[1]+1 to Pointer[1]+5 do
1093              begin PlotColor(2,I); Pointer[1]:=Pointer[1]+1; end;
1094            end
1095            else begin
1096              PlotColor(2,Pointer[1]);
1097              Pointer[1]:=Pointer[1]+1;
1098              PlotColor(2,Pointer[1])
1099            end; {of if-then-else}
1100          end; {PointNum=1}
```

```
1101          if (PointNum=2) and (Pointer[2]<NumData) then
1102          begin
1103            Stop:=false; Stop5:=false;
1104            if Pointer[2]=(Pointer[1]-1) then Stop:=true else Stop:=false;
1105            if Pointer[2]>=(Pointer[1]-5) then Stop5:=true else
    Stop5:=false;
1106            if not Stop5 then
1107            begin
1108              for I:=Pointer[PointNum] to Pointer[PointNum]+4 do
1109                PlotColor(1,I);
1110              Pointer[PointNum]:=Pointer[PointNum]+5;
1111            end;
1112            if not Stop then
1113            begin
1114              PlotColor(1,Pointer[2]);
1115              Pointer[PointNum]:=Pointer[PointNum]+1;
1116              PlotColor(2,Pointer[PointNum])
1117            end;
1118          end;
1119        end; {case 'R'}
1120        'T': begin
1121          Str(TimeVec[Pointer[PointNum]]*1440:7:0,Short);
1122          Str(RateVec[Pointer[PointNum]]:5:0,Long);
1123          Display('Time='+Short+'min  Q='+Long);
1124        end; {case 'T'}
1125      end; {of cases}
1126    until Cmd='0';
1127  end; {Procedure MovePointer2}
1128
1129  Procedure MarkSection; {Indicate a section of data on the graph}
1130  begin
1131    PlotColor(2,2);
1132    PointNum:=1; Pointer[1]:=2;
1133    Display('Set a marker at right end');
1134    Delay(1000); MovePointer2;
1135    Display('First marker set'); Delay(1000);
1136    Display('Set a marker at left end'); PlotColor(2,1);
1137    Delay(1000); PointNum:=2; Pointer[2]:=1;
1138    MovePointer2;
1139  end; {Procedure MarkSection}
1140  {#----- End indication of selected data -----#}
1141
1142  {#----- Curve fitting part 2 -----#}
1143  Procedure CalcTrans; {Mark a section of the data, and calculate the
1144  transmissivity from that section, assuming a simple, confined
1145  aquifer.}
1146  var
1147    EndNow, Error, TooMany: boolean;
1148    Factor: real;
1149  begin
1150    GraphColorMode; PlotData;
1151    PlotLinDdRL; PlotLogTimeRL;
1152    repeat
1153      EndNow:=true; TooMany:=false;
1154      MarkSection; Error:=false;
1155      Pointer1:=Pointer[1]; Pointer2:=Pointer[2];
1156      Q:=RateVec[Pointer2];
1157      if Q=0
1158      then begin
```

```
1159       Display('Recovery; What Q? ');
1160       Q:=ReadReal(0); {If rate=0 then a rate for the calculation is
     needed}
1161     end
1162     else for I:=Pointer2+1 to Pointer1 do
1163       if Q<>RateVec[I] then Error:=true;
1164     if not Error
1165     then begin
1166       TempInt:=Pointer1-Pointer2+1;
1167       if TempInt>=100 then
1168       begin
1169         TooMany:=true; Display('Selecting subset of data');
1170         Factor:=TempInt/100; delay(1000);
1171       end;
1172       if TempInt<=100 then
1173       for I:=1 to TempInt do
1174       begin
1175         Vec1[I]:=LogTimeMin[Pointer2+I-1];
1176         Vec2[I]:=DdVec[Pointer2+I-1];
1177       end
1178       else begin
1179         for I:=1 to 100 do
1180         begin
1181           Vec1[I]:=LogTimeMin[Pointer2+round((I-1)*Factor)];
1182           Vec2[I]:=DdVec[Pointer2+round((I-1)*Factor)];
1183         end;
1184       end; {if-then-else}
1185       if TooMany then TempInt:=100;
1186       LinReg(Vec1, Vec2, TempInt);
1187       Trans:=0.183*Q/Slope;
1188       Str(Trans:10:2,Short);
1189       Display('Transmissivity ='+Short);
1190     end
1191     else begin
1192       Display('ERROR: Change in Q!');
1193     end; {else}
1194     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1195     Display('Another Transmissivity?');
1196     if Response2('YN')='Y' then EndNow:=false;
1197     if not EndNow then PlotData;
1198   until EndNow;
1199   TextMode; TextColor(green);
1200 end; {of Procedure CalcTrans}
1201
1202 Procedure RunTheisFit; {Control the calling of THEISFIT}
1203 var
1204   ChangeInQ, TooMany, ZeroQ: boolean;
1205   Factor: real;
1206 begin
1207   GraphColorMode; PlotData;
1208   PlotLinDdRL; PlotLogTimeRL;
1209   MarkSection; TooMany:=false; ChangeInQ:=false;
1210   Pointer1:=Pointer[1]; Pointer2:=Pointer[2];
1211   Q:=RateVec[Pointer2];
1212   if Q>0 then ZeroQ:=false else ZeroQ:=true;
1213   for I:=Pointer2+1 to Pointer1 do
1214     if Q<>RateVec[I] then ChangeInQ:=true;
1215   TextMode; TextColor(green);
```

```
1216   if not ChangeInQ and not ZeroQ
1217   then begin
1218     TempInt:=Pointer1-Pointer2+1;
1219     if TempInt>=100 then
1220     begin
1221       TooMany:=true; writeln('Selecting a subset of the data.');
1222       Factor:=TempInt/100;
1223       writeln('The record numbers below are those used in TheisFit.');
1224     end;
1225     if TempInt<=100 then
1226     for I:=1 to TempInt do
1227     begin
1228       Vec1[I]:=TimeVec[Pointer2+I-1];
1229       Vec2[I]:=DdVec[Pointer2+I-1];
1230     end
1231     else begin
1232       for I:=1 to 100 do
1233       begin
1234         Vec1[I]:=TimeVec[Pointer2+round((I-1)*Factor)];
1235         Vec2[I]:=DdVec[Pointer2+round((I-1)*Factor)];
1236         write(Pointer2+round((I-1)*Factor):4);
1237       end;
1238       writeln;
1239     end; {if-then-else}
1240     if TooMany then TempInt:=100;
1241     TheisFit(Vec1,Vec2,Distance,Q,TempInt);
1242     GraphColorMode; PlotData; PlotLinDdRL; PlotLogTimeRL;
1243     for I:=Pointer2 to Pointer1 do PlotColor(2,I);
1244     for I:=1 to NumData do
1245     begin
1246       u:=sqr(Distance)*StorCoef/(4*Trans*TimeVec[I]);
1247       TrialDdVec[1]:=Q*WellFunc(u)/(4*Pi*Trans);
1248       Pointer[1]:=I; PlotColor2(3,1,1);
1249     end;
1250     Display('The fitted data are in yellow');
1251   end {if data OK}
1252   else begin
1253     TextMode; TextColor(green);
1254     if ChangeInQ then
1255       writeln('Change in discharge rate!  TheisFit not called.');
1256     if ZeroQ then writeln('Discharge rate <=0! TheisFit not called.');
1257     writeln('Press any key to continue.');
1258   end; {else}
1259   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1260   TextMode; TextColor(green);
1261 end; {of procedure RunTheisFit}
1262
1263 Procedure Storage {Calculate S by Neuton's Method}
1264 (Drawdown, R, Q, Time: real);
1265   var
1266     Finished: boolean;
1267     Ch: char;
1268     Iterat, F: byte;
1269     Wm, WellFunction, Wu, W0: real;
1270     U, U2, U3: real;
1271     Long: String[80]; Short: String[20];
1272   const
1273     Tolerance=0.001;
1274
```

```
1275   function D3 {Derivitive function}
1276   : real;
1277   var D1, D2, TempVal: real;
1278   begin
1279     D1:=(U*-EXP(-U)-EXP(-U))/U2;
1280     TempVal:=(C8+C9*U+U2)*(C7+2*U)-(C6+C7*U+U2)*(C9+2*U);
1281     D2:=TempVal/((C8+C9*U+U2)*(C8+C9*U+U2));
1282     D3:=1/(U*EXP(U))*D2+(C6+C7*U+U2)/(C8+C9*U+U2)*D1
1283   end;
1284
1285   function D4 {Derivitive function}
1286   : real;
1287   begin
1288     D4:=-1/U+C1+2*C2*U+3*C3*U2+4*C4*U3+5*C5*U2*U2
1289   end;
1290
1291 begin {# Main part of Storage}
1292   Display('Enter Transmissivity '); Trans:=ReadReal(0);
1293   ConvergFail:=false; Iterat:=0; F:=0;
1294   WellFunction:=Drawdown*4*Pi*Trans/Q; U:=0.01;
1295   Finished:=false;
1296   while not Finished and not ConvergFail do
1297   begin
1298     U2:=sqr(u); U3:=U*U2; Iterat:=Succ(Iterat);
1299     if Iterat>30 then ConvergFail:=true;
1300     Wu:=WellFunc(u);
1301     Wm:=Wu-WellFunction;
1302     W0:=abs(WellFunction-Wu); if W0<WellFunction*Tolerance then
    finished:=true;
1303     if not Finished then
1304     begin
1305       if U<1 then U:=U-Wm/D4 else U:=U-Wm/D3;
1306       if (U<=0) and (F=0) then
1307       begin
1308         U:=0.0001; F:=1;
1309       end
1310       else if (U<=0) and (F=1) then
1311         begin
1312           U:=1E-10; F:=2;
1313         end
1314         else if (U<=0) and (F=2) then U:=9.999999e-21;
1315       if U>80 then U:=80
1316     end; {if not Finished}
1317   end; {while not Finished}
1318   if ConvergFail=false then
1319   begin
1320     StorCoef:=U*4*Trans*Time/Sqr(R);
1321     Str(StorCoef:10:8,Short);
1322     Display('Storage ='+Short);
1323   end
1324   else begin
1325     Display('STORAGE failed to converge?')
1326   end;
1327 end; {of Procedure Storage}
1328
1329 Procedure RunStorage; {To calculate Storage Coefficent at a given time}
1330 var
1331   EndNow, ZeroQ: boolean;
1332 begin
```

```
1333   GraphColorMode; PlotData;
1334   PlotLinDdRL; PlotLogTimeRL;
1335   repeat
1336     EndNow:=true;
1337     Display('Please set a pointer'); Delay(1200);
1338     PlotColor(2,1);
1339     PointNum:=1; Pointer[1]:=1; MovePointer1;
1340     Q:=RateVec[1];
1341          if  RateVec[Pointer[1]]>0  then  ZeroQ:=false  else ZeroQ:=true;
1342     if not ZeroQ then
1343     begin
1344       for I:=1 to Pointer[1] do
1345       begin
1346         if Q<>RateVec[I] then
1347         begin
1348           Str(TimeVec[I]:8:3,Short); Display('Rate change: time '
    +Short);
1349           Q:=RateVec[I]; Delay(2000);
1350         end;
1351       end; {for I:=1 to Pointer}
1352       Storage(DdVec[Pointer[1]], Distance,
1353       RateVec[Pointer[1]], TimeVec[Pointer[1]]);
1354       Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1355       Display('Another storage?');
1356       if Response2('YN')='Y'
1357         then begin EndNow:=false; PlotData; end;
1358         end
1359     else begin
1360       TextMode; TextColor(green);
1361            writeln('Discharge rate <=0! Cannot procede with analysis.');
1362       writeln('Press any key to continue.');
1363       Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1364     end; {if-then-else}
1365   until EndNow;
1366   TextMode; TextColor(green);
1367 end; {of procedure RunStorage}
1368
1369 Procedure Leakage {Calculate leak. coef. T and S are known}
1370 (Time, Drawdown, Q, R: real);
1371 var
1372   Finished: boolean;
1373   LowMark, HighMark: real;
1374   u, TrialDd, Trans, StorCoef, Lc, Tcl, Kv: real;
1375   ItNum: integer;
1376   Short: ShortString;
1377
1378 begin {# Main part of procedure Leakage}
1379   ItNum:=0;
1380   write('Transmissivity? '); Trans:=ReadReal(1);
1381   write('Storage coefficient? '); StorCoef:=ReadReal(1);
1382   writeln('Calculating'); writeln;
1383   u:=Sqr(R)*StorCoef/(4*Trans*Time);
1384   LowMark:=1e-15; HighMark:=1; Finished:=false;
1385   repeat
1386     ItNum:=ItNum+1;
1387     Num:=exp((ln(LowMark)+ln(HighMark))/2);
1388     TrialDd:=(Q*LeakFunc(u,Num)/(4*Pi*Trans));
1389     if (TrialDd>Drawdown)
1390       then Lowmark:=Num
```

```
1391        else Highmark:=Num;
1392      if abs(TrialDd-Drawdown)<0.0001 then Finished:=true;
1393    until Finished or (ItNum>25);
1394    if Finished
1395    then begin
1396      Lc:=Num/R;
1397      writeln('Final inverse leakage coefficient =',Lc:10:8);
1398      writeln;
1399      write('Thickness of confining layer? '); Tcl:=ReadReal(2);
1400      Kv:=Trans*Tcl*Sqr(Lc);
1401      writeln('Hydraulic conductivity of aquitard =',Kv:10:8);
1402    end
1403    else
1404      writeln('No leakage calculated.');
1405 end; {Procedure Leakage}
1406
1407 Procedure RunLeakage; {Calculate leak. coef. and K of the confining bed}
1408 var
1409   ZeroQ, RateChange: boolean;
1410 begin
1411   GraphColorMode; PlotData; RateChange:=false;
1412   PlotLinDdRL; PlotLogTimeRL;
1413   Display('Please set a pointer'); Delay(1200);
1414   PlotColor(2,1);
1415   PointNum:=1; Pointer[1]:=1; MovePointer1;
1416   Q:=RateVec[1];
1417   if Q>0 then ZeroQ:=false else ZeroQ:=true;
1418   if not ZeroQ then
1419   begin
1420     for I:=1 to Pointer[1] do
1421     begin
1422       if Q<>RateVec[I] then
1423       begin
1424           Str(TimeVec[I]:8:3,Short); Display('Rate change: time '+Short);
1425         Q:=RateVec[I]; Delay(100); RateChange:=true
1426       end;
1427     end; {for I:=1 to Pointer}
1428     if RateChange then delay(3000); {Allow user to read message}
1429     TextMode; TextColor(green);
1430     Leakage(TimeVec[Pointer[1]], DdVec[Pointer[1]],
1431       RateVec[Pointer[1]], Distance);
1432   end
1433   else begin
1434     TextMode; TextColor(green);
1435     writeln('Discharge rate <= 0! Cannot procede with this analysis.');
1436   end; {if-then-else}
1437   writeln('Press any key to continue.');
1438   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1439 end; {of procedure RunLeakage}
1440 {#----- End Curve fitting part 2 -----#}
1441
1442 Procedure RunOption; {Prepare for option requiring selected data}
1443 var
1444   Error, NotFarEnough, ChangeInQ, ZeroQ, Valid: boolean;
1445   NumOfPointers: integer;
1446 begin
1447   Valid:=true; ChangeInQ:=false; ZeroQ:=false; ConvergFail:=false;
1448   if WellType=Pumped then
1449   begin
```

```
1450      writeln('Pumped well data; invalid!'); Delay(2000);
1451      Valid:=false;
1452    end
1453    else begin
1454      GraphColorMode; PlotData; PlotLinDdRL; PlotLogTimeRL;
1455      repeat
1456        Error:=false;
1457         Display('How many pointers? '); NumOfPointers:=round(ReadReal(0));
1458        if (NumOfPointers>30) or (NumOfPointers>NumData)
1459          then begin Display('Sorry, too many'); Delay(2000); Error:=true;
    end;
1460        if NumOfPointers<3
1461          then begin Display('Sorry, not enough'); Delay(2000);
    Error:=true; end;
1462      until Error=false;
1463      PointNum:=1;
1464      begin
1465        PlotColor(2,1); Pointer[PointNum]:=1;
1466        repeat
1467          NotFarEnough:=false; MovePointer1;
1468          if (Pointer[PointNum]-1)<(NumOfPointers-PointNum) then
1469          begin
1470            NotFarEnough:=true; Display('Not far enough'); Delay(1000);
1471          end;
1472          PlotColor(2,Pointer[1]);
1473          ColorOfPlot[Pointer[PointNum]]:=Redd;
1474        until NotFarEnough=false;
1475      end;
1476      for PointNum:=2 to NumOfPointers do
1477      begin
1478        Pointer[PointNum]:=round(Pointer[1]-((PointNum-1)/
    (NumOfPointers-1))*
1479          (Pointer[1]-1));
1480        PlotColor(2,Pointer[PointNum]);
1481        ColorOfPlot[Pointer[PointNum]]:=Redd;
1482      end;
1483      for PointNum:=2 to NumOfPointers do
1484      begin MovePointer1; PlotColor(2,Pointer[PointNum]) end;
1485      Q:=RateVec[1]; ChangeInQ:=false;
1486           if  RateVec[Pointer[1]]>0  then  ZeroQ:=false  else ZeroQ:=true;
1487      for I:=1 to Pointer[1] do
1488        if Q<>RateVec[I] then ChangeInQ:=true;
1489      if not ChangeInQ and not ZeroQ
1490      then begin
1491        for I:=1 to NumOfPointers do
1492        begin
1493          Vec1[I]:=TimeVec[Pointer[NumOfPointers-I+1]];
1494          Vec2[I]:=DdVec[Pointer[NumOfPointers-I+1]];
1495        end;
1496        TextMode; TextColor(green);
1497        if AqType=Leaky
1498          then LeakyFit(Vec1,Vec2,Distance,Q,NumOfPointers);
1499        if AqType=Confined
1500        then begin
1501          if (AqExtent=Strip) or (AqExtent=SemiStrip) then
1502            if Solution=FirstSol
1503              then StripFit(Vec1,Vec2,Distance,Q,NumOfPointers)
1504              else StripFit2(Vec1,Vec2,Distance,Q,NumOfPointers);
1505          if (AqExtent=Bounded) then
```

```
1506          Case Option of
1507            '1': BoundFit(Vec1,Vec2,Distance,Q,NumOfPointers);
1508            '2': BoundFit2(Vec1,Vec2,Distance,Q,NumOfPointers);
1509          end; {cases}
1510          if AqExtent=SemiBounded then
1511            BoundFit3(Vec1,Vec2,Distance,Q,NumOfPointers);
1512        end; {Confined aquifer}
1513        if (AqType=Unconfined) and (AqExtent=Infinite) then
1514        Case Option of
1515          '1': UnconfinedFit(Vec1,Vec2,Distance,Q,NumOfPointers);
1516          '2': UnconfinedFit2(Vec1,Vec2,Distance,Q,NumOfPointers);
1517        end; {cases}
1518      end
1519      else begin
1520        TextMode; TextColor(green);
1521        if ChangeInQ then
1522          writeln('ERROR: Change in Q!  The operation was not
    attempted.');
1523        if ZeroQ then
1524          writeln('ERROR: Q<=0!  The operation was not attempted.');
1525        writeln('Press any key to continue.');
1526        Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1527      end; {else}
1528      if not ChangeInQ and not ZeroQ and not ConvergFail then
1529      begin  {Display best fit data}
1530        if AqType=Leaky then
1531        for I:=1 to NumOfPointers do
1532        begin
1533          u:=sqr(Distance)*StorCoef/(4*Trans*Vec1[I]);
1534          TrialDdVec[I]:=Q*LeakFunc(u,Distance*Lc)/(4*Pi*Trans);
1535        end;
1536        GraphColorMode; PlotData; PlotLinDdRL; PlotLogTimeRL;
1537        for I:=1 to NumOfPointers do PlotColor(2,Pointer[I]);
1538        for I:=1 to NumOfPointers do
1539          PlotColor2(3,I,NumOfPointers-I+1);
1540        Display('The fitted data are in yellow');
1541        Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
1542        TextMode; TextColor(green);
1543      end; {display of fitted data}
1544    end; {if Valid}
1545 end; {Procedure RunOption}
1546
1547 {#----- Menu section -----#}
1548 Procedure UnconfinedMenu;
1549 const
1550    ValidResponse: set of char=['1','2','R'];
1551
1552    procedure DisplayMenu;
1553    begin
1554      ClrScr; writeln('       ANALYZE, Unconfined aquifer menu');
1555      writeln; writeln('Press the indicated number key.');
1556      writeln('1: Calculate T, S, and saturated thickness.');
1557      writeln('2: Calculate saturated thickness.'
1558        ,'  T and S are required.');
1559      writeln('R: Return to ANALYZE main menu.');
1560    end; {sub procedure DisplayMenu}
1561
1562 begin
1563    DisplayMenu;
```

```
1564   repeat
1565     repeat read(kbd,Option); until UpCase(Option) in ValidResponse;
1566     Option:=UpCase(Option);
1567     if Option<>'R' then begin RunOption; DisplayMenu; end;
1568   until Option='R';
1569 end; {Procedure UnconfinedMenu}
1570
1571 Procedure ConfinedMenu;
1572 const
1573   ValidResponse: set of char=['1','2','3','R'];
1574
1575   procedure DisplayMenu;
1576   begin
1577     ClrScr; writeln('       ANALYZE, Confined aquifer menu');
1578     writeln; writeln('Press the indicated number key.');
1579     writeln('1: Calculate Transmissivity by linear regression of a ',
1580       'segment of the');
1581     writeln('   data.  ie. A best fit straight line is found, and ',
1582       'transmissivity');
1583     writeln('   calulated from it.');
1584     writeln('2: Calculate Storage from one datum.'
1585       ,'  Transmissivity is required.');
1586     writeln('3: Run TheisFit (to produce a best fit transmissivity'
1587       ,' and storage).');
1588     writeln('R: Return to ANALYZE main menu.');
1589   end; {sub procedure DisplayMenu}
1590
1591 begin
1592   DisplayMenu;
1593   repeat
1594     repeat read(kbd,Option); until UpCase(Option) in ValidResponse;
1595     Option:=UpCase(Option);
1596     case Option of
1597       '1': CalcTrans;
1598       '2': begin
1599         if Distance<=0
1600         then begin
1601           writeln('Option not available.  Recorded distance <=0!');
1602           delay(4000);
1603         end
1604         else RunStorage;
1605       end;
1606       '3': begin
1607         if Distance<=0
1608         then begin
1609                 writeln('Option not available.  Recorded distance <=0!');
1610           delay(4000);
1611         end
1612         else RunTheisFit;
1613       end;
1614     end; {cases}
1615     if Option<>'R' then DisplayMenu;
1616   until Option='R';
1617 end; {Procedure ConfinedMenu}
1618
1619 Procedure LeakyMenu;
1620 const
1621   ValidResponse: set of char=['1','2','R'];
```

```
1622
1623   procedure DisplayMenu;
1624   begin
1625     ClrScr; writeln('       ANALYZE, Leaky aquifer menu');
1626     writeln; writeln('Press the indicated number key.');
1627     writeln('1: Calculate Leakage from one datum.'
1628       ,'  Both transmissivity and storage');
1629     writeln('   coefficient are required.');
1630     writeln('2: Run LeakyFit to produce a best fit transmissivity,'
1631       ,' storage, and');
1632     writeln('   inverse leakage coefficient.  Guesses for parameters'
1633       ,' are required.');
1634     writeln('R: Return to ANALYZE main menu.');
1635   end; {sub procedure DisplayMenu}
1636
1637 begin
1638   DisplayMenu;
1639   repeat
1640     repeat read(kbd,Option); until UpCase(Option) in ValidResponse;
1641     Option:=UpCase(Option);
1642     case Option of
1643       '1': RunLeakage;
1644       '2': RunOption;
1645     end; {cases}
1646     if Option<>'R' then DisplayMenu;
1647   until Option='R';
1648 end; {Procedure LeakyMenu}
1649
1650 Procedure BoundedMenu;
1651 {A single boundary, and a single 'semi' boundary}
1652 const
1653   ValidResponse: set of char=['1','2','3','R'];
1654
1655   procedure DisplayMenu;
1656   begin
1657     ClrScr; writeln('       ANALYZE, Single boundary menu');
1658     writeln; writeln('Press the indicated number key.');
1659     writeln('1: Fit a bounded aquifer to the data.  Transmissivity,'
1660       ,' storage, and');
1661     writeln('   distance to one discharge boundary image well, are all'
1662       ,' evaluated.');
1663     writeln('2: Fitting of a bounded aquifer to the data; given '
1664       ,'values for T and S.');
1665     writeln('3: Fitting of a semi bounded aquifer to the data; given '
1666       ,'values for T and S.');
1667     writeln('R: Return to ANALYZE main menu.');
1668   end; {sub procedure DisplayMenu}
1669
1670 begin
1671   DisplayMenu;
1672   repeat
1673     repeat read(kbd,Option); until UpCase(Option) in ValidResponse;
1674     Option:=UpCase(Option);
1675     case Option of
1676       '1': begin
1677         AqExtent:=Bounded; RunOption;
1678       end;
1679       '2': begin
1680         AqExtent:=Bounded; RunOption;
```

```
1681       end;
1682       '3': begin
1683         AqExtent:=SemiBounded; RunOption;
1684       end;
1685     end; {cases}
1686     if Option<>'R' then DisplayMenu;
1687   until Option='R';
1688 end; {Procedure BoundedMenu}
1689
1690 Procedure StripMenu;
1691 {A strip aquifer, and a 'semi' strip aquifer.}
1692 const
1693   ValidResponse: set of char=['1','2','3','R'];
1694
1695   procedure DisplayMenu;
1696   begin
1697     ClrScr; writeln('       ANALYZE, Strip aquifer menu');
1698     writeln('     Simple strip');
1699     writeln; writeln('Press the indicated number key.');
1700     writeln('1: Fit a strip aquifer to the data.  Both transmissivity'
1701       ,' and storage');
1702     writeln('   coefficient are required.');
1703     writeln;
1704     writeln('     Semi-strip');
1705     writeln('2: Fit a semistrip aquifer to the data.  Two
    transmissivities,'
1706       ,' and storage');
1707     writeln('   are required.');
1708     writeln('3: Fit a semistrip aquifer to the data.  Transmissivity ',
1709       ' and storage');
1710     writeln('inside strip are required, T outside of strip ',
1711       'is calculated.');
1712     writeln;
1713     writeln('R: Return to ANALYZE main menu.');
1714   end; {sub procedure DisplayMenu}
1715
1716 begin
1717   DisplayMenu;
1718   repeat
1719     repeat read(kbd,Option); until UpCase(Option) in ValidResponse;
1720     Option:=UpCase(Option);
1721     case Option of
1722       '1': begin
1723         AqExtent:=Strip; Solution:=FirstSol; RunOption;
1724       end;
1725       '2': begin
1726         AqExtent:=SemiStrip; Solution:=FirstSol; RunOption;
1727       end;
1728       '3': begin
1729         AqExtent:=SemiStrip; Solution:=SecondSol; RunOption;
1730       end;
1731     end; {cases}
1732     if Option<>'R' then DisplayMenu;
1733   until Option='R';
1734 end; {Procedure StripMenu}
1735
1736 Procedure MainMenu;
1737 const
1738   ValidResponse: set of char=['1','2','3','4','5','E'];
```

```
1739
1740  procedure displayMenu;
1741  begin
1742    ClrScr; writeln('       ANALYZE, Main menu'); writeln;
1743    writeln('Press the key indicating the required aquifer type.');
1744    writeln('1: A simple, infinite, confined aquifer.');
1745    writeln('2: An infinite leaky confined aquifer.');
1746    writeln('3: A confined aquifer with a single boundary or a semi'
1747      ,' boundary.');
1748    writeln('4: A confined strip aquifer or a semistrip aquifer.');
1749    writeln('5: An infinite unconfined aquifer.');
1750    writeln('E: Exit.');
1751  end; {sub procedure DisplayMenu}
1752
1753 begin
1754  DisplayMenu;
1755  repeat
1756    repeat read(kbd,Option); until UpCase(Option) in ValidResponse;
1757    Option:=UpCase(Option);
1758    case Option of
1759      '1': begin
1760        AqExtent:=Infinite; AqType:=Confined;
1761        ConfinedMenu;
1762      end;
1763      '2': begin
1764        if Distance<=0
1765        then begin
1766          writeln('Option not available.  Recorded distance <=0!');
1767          delay(4000);
1768        end
1769        else begin
1770          AqExtent:=Infinite; AqType:=Leaky;
1771          LeakyMenu;
1772        end; {if-then-else}
1773      end;
1774      '3': begin
1775        if Distance<=0
1776        then begin
1777          writeln('Option not available.  Recorded distance <=0!');
1778          delay(4000);
1779        end
1780        else begin
1781          AqType:=Confined;
1782          BoundedMenu;
1783        end; {if-then-else}
1784      end;
1785      '4': begin
1786        if Distance<=0
1787        then begin
1788          writeln('Option not available.  Recorded distance <=0!');
1789          delay(4000);
1790        end
1791        else begin
1792          AqType:=Confined;
1793          StripMenu;
1794        end; {if-then-else}
1795      end;
1796      '5': begin
1797        if Distance<=0
```

```
1798          then begin
1799            writeln('Option not available.  Recorded distance <=0!');
1800            delay(4000);
1801          end
1802          else begin
1803            AqType:=UnConfined; AqExtent:=Infinite;
1804            UnconfinedMenu;
1805          end; {if-then-else}
1806        end;
1807      end; {cases}
1808      if Option<>'E' then DisplayMenu;
1809    until Option='E';
1810  end; {Procedure MainMenu}
1811  {#----- End of menu section -----#}
1812  {#----- End of procedures and functions -----#}
1813
1814  begin {# Main part of program}
1815    clrscr; TextColor(green); Finished:=false;
1816    writeln('                    ANALYZE: version 1.1');
1817    writeln('           An aid in analysis of aquifer test data.');
1818    writeln;
1819    writeln('   This program is not capable of calculating aquifer
      properties');
1820    writeln('on it''s own.  It is a tool which, like any tool, must be
      used by');
1821    writeln('a person competent in the use of that tool.  Any person not
      having');
1822    writeln('a good understanding of hydrogeology and using this program,
      must');
1823    writeln('not take the answers it gives seriously.');
1824    writeln;
1825    repeat
1826      ReadTestDataFile(TimeMin, DdVec, RateVec,
1827        TestType, WellType, Distance, NumData);
1828      if FileName<>'x' then
1829      begin
1830        for I:=1 to NumData do TimeVec[I]:=TimeMin[I]/1440;
1831        ViewAlterData             {Summary of data read from file}
1832          (TimeMin, DdVec, RateVec,
1833          TestType, WellType, Distance, NumData);
1834        writeln;
1835        if WellType=Pumped then
1836          writeln('WARNING: Pumped well data, see notes.');
1837        Remove0;                  {Remove times (and drawdowns) <=0, adjust
      NumData}
1838        LoadTimeLog;              {Load log of time vectors}
1839        GetMaxMin;                {Get the maximum and minimum times and
      drawdowns}
1840        writeln('Drawdown ranges from ',MinDd:7:3,' to ',MaxDd:7:3,'m.');
1841        writeln('Time ranges from ',MinTime:7:3,' to '
      ,MaxTime:7:3,'min.');
1842        writeln; writeln('Press any key to continue');
1843        Ch:='x'; repeat Read(Kbd,Ch) until Ch<>'x';
1844        GraphMaxMin;
1845        MainMenu;
1846        write('Do you want to ');
1847        if CapOptions('Return to primary menu, or go Load another
      file? ')=1 then
1848          Finished:=true;
```

```
1849     end {if FileName<>'x'}
1850     else Finished:=true;
1851   until Finished;
1852   ChainTo('GWMENU.CHN',IOCode);
1853   if IOCode<>0
1854   then begin
1855     writeln('Unable to find program GWMENU.CHN!');
1856   end;
1857 end. {# of program ANALYZE}
```

15. LISTING OF FILE BOUNDFIT.PRC

```
  1
  2 {#----- Procedure BoundFit -----#}
  3 OVERLAY Procedure BoundFit {T, S, and distance of image well calculated}
  4 (TimeVec, DdVec: SmallVec; R, Q: real; NumOfPointers: integer);
  5 var
  6   Improvement: Boolean;
  7   I, J: integer;
  8   CurrentRms, OldRms, TrialRms, u: Real;
  9   Guessed, Current, Trial, CurrentInc, TrialInc: SmallVec;
 10
 11   procedure Test; {Check quality of fit}
 12   begin
 13     for I:=1 to NumOfPointers do
 14       TrialDdVec[I]:=CalcDd(R, TimeVec[I], Q, Trial[1], Trial[2]);
 15     for I:=1 to NumOfPointers do
 16       TrialDdVec[I]:=TrialDdVec[I]+
 17       CalcDd(Trial[3], TimeVec[I], Q, Trial[1], Trial[2]);
 18     TrialRms:=RmsError(DdVec, TrialDdVec, NumOfPointers);
 19   end; {sub procedure Test}
 20
 21   procedure AdjustAll; {Adjust all three unknowns}
 22   var
 23     Change: array[1..3] of boolean;
 24     I, J, K, L: integer;
 25     LowestRms: real;
 26     Best: array[1..3] of real;
 27   begin
 28     LowestRms:=CurrentRms;
 29     for I:=1 to 3 do
 30     begin
 31       for J:=1 to 3 do
 32       begin
 33         for K:=1 to 3 do
 34         begin
 35           case I of
 36             1: Trial[1]:=Current[1]*CurrentInc[1];
 37             2: Trial[1]:=Current[1];
 38             3: Trial[1]:=Current[1]/CurrentInc[1];
 39           end;
 40           case J of
 41             1: Trial[2]:=Current[2]*CurrentInc[2];
 42             2: Trial[2]:=Current[2];
 43             3: Trial[2]:=Current[2]/CurrentInc[2];
 44           end;
 45           case K of
 46             1: Trial[3]:=Current[3]*CurrentInc[3];
 47             2: Trial[3]:=Current[3];
```

```
48            3: Trial[3]:=Current[3]/CurrentInc[3];
49          end;
50          Test;
51          if TrialRms<LowestRms*0.999 then
52          begin
53            LowestRms:=TrialRms;
54            writeln('Improved values');
55            for L:=1 to 3 do
56            begin
57              Best[L]:=Trial[L];
58              case L of
59                1: write('Transmissivity =');
60                2: write('Storage coefficient =');
61                3: write('Distance to image well =');
62              end; {of cases}
63              writeln(Trial[L]:13:6);
64              if Trial[L]<>Current[L] then Change[L]:=true else
65                Change[L]:=false;
66            end; {for}
67            writeln('RMS error =',TrialRms:8:6); writeln;
68          end; {if Trial}
69        end; {for K}
70      end; {for J}
71    end; {for I}
72    if LowestRms<>CurrentRms
73    then begin
74      CurrentRms:=LowestRms;
75      for I:=1 to 3 do
76      begin
77        if Change[I]=false then CurrentInc[I]:=sqrt(CurrentInc[I]);
78        Current[I]:=Best[I];
79      end {for}
80    end; {if}
81  end; {sub procedure AdjustAll}
82
83  procedure CalcBounded; {Control the curve fitting}
84  begin
85    Current[1]:=Guessed[1]; Trial[1]:=Guessed[1];
86    Current[2]:=Guessed[2]; Trial[2]:=Guessed[2];
87    Current[3]:=Guessed[3]; Trial[3]:=Guessed[3];
88    Test;
89    CurrentRms:=TrialRms;
90    writeln('CurrentRms = ',CurrentRms:8:4); OldRms:=CurrentRms;
91    for I:=1 to 3 do
92    begin CurrentInc[I]:=2; TrialInc[I]:=CurrentInc[I]; end;
93    repeat
94      AdjustAll;
95      if OldRms=CurrentRms then
96      begin
97        writeln('  Current factors: ');
98        for I:=1 to 3 do
99        begin
100         case I of
101           1: write('Transmissivity: ');
102           2: write('Storage coefficient: ');
103           3: write('Image well distance: ');
104         end; {of cases}
105         CurrentInc[I]:=sqrt(CurrentInc[I]);
```

```
106           writeln(CurrentInc[I]:8:6,'  ');
107         end; {for}
108         writeln; writeln;
109       end {if}
110       else
111         OldRms:=CurrentRms;
112     until (CurrentRms<0.005) or
113       ((CurrentInc[1]<1.03) and (CurrentInc[2]<1.03) and
114       (CurrentInc[3]<1.03));
115   end; {sub procedure CalcBounded}
116
117 begin {# Main part of procedure BoundFit}
118   TextColor(Green); ClrScr;
119   writeln('Please enter guesses for the following:-');
120   writeln;
121   write('Transmissivity? '); Guessed[1]:=ReadReal(2);
122   write('Storage Coeficient? '); Guessed[2]:=ReadReal(2);
123   write('Distance of the image well? '); Guessed[3]:=ReadReal(2);
124   writeln('Calculating');
125   CalcBounded;
126   writeln; writeln('Best fit values are:');
127   for I:=1 to 3 do
128   begin
129     case I of
130       1: writeln('Transmissivity = ',Current[I]:9:2);
131       2: writeln('Storage coefficient = ',Current[I]:9:7);
132       3: writeln('Distance to image well = ',Current[I]:8:1);
133     end; {of cases}
134   end;
135   writeln('Final root-mean-square error is ',CurrentRms:9:5);
136   for I:=1 to 3 do Trial[I]:=Current[I]; Test; {Get best TrialDdVec
    data}
137   writeln('Press any key to continue.');
138   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
139 end; {END OVERLAY Procedure BoundFit}
140
141 OVERLAY Procedure BoundFit2 {T and S given, distance to image well
    found}
142 (TimeVec, DdVec: SmallVec; R, Q: real;
143   NumofPointers: integer);
144 var
145   Change, Finished: boolean;
146   ItNum, Best: byte;
147   I, J: integer;
148   Distance, Factor, u, CurrentRms, BestRms: real;
149   TempR: array [1..3] of real;
150
151   procedure Test; {Check quality of fit}
152   begin
153     u:=Sqr(R)*StorCoef/(4*Trans*TimeVec[I]);
154     TrialDdVec[I]:=(Q*WellFunc(u)/(4*Pi*Trans));
155     u:=Sqr(TempR[J])*StorCoef/(4*Trans*TimeVec[I]);
156     TrialDdVec[I]:=TrialDdVec[I]+(Q*WellFunc(u)/(4*Pi*Trans));
157   end; {sub procedure Test}
158
159 begin
160   ItNum:=0;
161   writeln('Calculation of distance to an image well; T and S given.');
162   writeln;
```

```
163   write('Transmissivity? '); Trans:=ReadReal(1);
164   write('Storage coefficient? '); StorCoef:=ReadReal(1);
165   writeln('Calculating'); writeln;
166   Distance:=100; Factor:=2; Finished:=false; BestRms:=1000;
167   repeat
168     ItNum:=ItNum+1; Change:=false;
169     for J:=1 to 3 do
170     begin
171       for I:=1 to NumOfPointers do
172       begin
173         case J of
174           1: TempR[J]:=Distance/Factor;
175           2: TempR[J]:=Distance;
176           3: TempR[J]:=Distance*Factor;
177         end; {of cases}
178         Test;
179       end; {for I}
180       CurrentRms:=RmsError(DdVec, TrialDdVec, NumOfPointers);
181       if CurrentRms<BestRms then
182       begin
183         Best:=J;
184         Change:=true;
185         writeln('Improved value = ',TempR[J]:10:2,'m, new RMS ='
186           ,CurrentRms:7:5);
187         BestRms:=CurrentRms;
188       end; {if improvement}
189     end; {for J}
190     if Change
191       then Distance:=TempR[Best]
192       else begin
193         Factor:=Sqrt(Factor);
194         writeln('Iteration ',ItNum,',  New factor =',Factor:8:4);
195     end; {if-then-else}
196     if Factor<1.001 then Finished:=true;
197   until Finished or (ItNum>25);
198   if Finished
199   then begin
200     writeln('Best fit image well distance was ',Distance:8:2);
201     writeln('Final root-mean-square error is ',BestRms:9:5);
202     J:=1; TempR[1]:=Distance; Test;
203     writeln('Press any key to continue.');
204     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
205   end
206   else begin
207     writeln('No image well distance calculated - failed to converge.');
208     writeln('Press any key to continue.');
209     Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
210   end;
211 end; {END OVERLAY Procedure BoundFit2}
212
213 OVERLAY Procedure BoundFit3  {T and S are given, T2 and image well
    found}
214 (TimeVec, DdVec: SmallVec; R, Q: real; NumofPointers: integer);
215 var
216   Improvement: Boolean;
217   I, J: integer;
218   CurrentRms, OldRms, TrialRms, ThisQ, u: Real;
219   Guessed, Current, Trial, CurrentInc, TrialInc: SmallVec;
220
```

```
221  procedure Test; {Check quality of fit}
222  begin
223    ThisQ:=Q;
224    for I:=1 to NumOfPointers do
225      TrialDdVec[I]:=CalcDd(R, TimeVec[I], Q, Trans, StorCoef);
226    ThisQ:=Q*-(Trial[1]/Trans-1)/(Trial[1]/Trans+1);
227    for I:=1 to NumOfPointers do
228      TrialDdVec[I]:=TrialDdVec[I]+
229      CalcDd(Trial[2], TimeVec[I], Q, Trans, StorCoef);
230    TrialRms:=RmsError(DdVec, TrialDdVec, NumOfPointers);
231  end; {sub procedure Test}
232
233  procedure Adjust2; {Adjust the two unknowns}
234  var
235    Change: array[1..2] of boolean;
236    I, J, K, L: integer;
237    LowestRms: real;
238    Best: array[1..3] of real;
239  begin
240    LowestRms:=CurrentRms;
241    for I:=1 to 3 do
242    begin
243      for J:=1 to 3 do
244      begin
245        case I of
246          1: Trial[1]:=Current[1]*CurrentInc[1];
247          2: Trial[1]:=Current[1];
248          3: Trial[1]:=Current[1]/CurrentInc[1];
249        end;
250        case J of
251          1: Trial[2]:=Current[2]*CurrentInc[2];
252          2: Trial[2]:=Current[2];
253          3: Trial[2]:=Current[2]/CurrentInc[2];
254        end;
255        Test;
256        if TrialRms<LowestRms*0.999 then
257        begin
258          LowestRms:=TrialRms;
259          writeln('Improved values');
260          for L:=1 to 2 do
261          begin
262            Best[L]:=Trial[L];
263            case L of
264              1: write('T2 =');
265              2: write('Distance to image well =');
266            end; {of cases}
267            writeln(Trial[L]:13:6);
268            if Trial[L]<>Current[L] then Change[L]:=true
269              else Change[L]:=false;
270          end; {for L}
271          writeln('RMS error =',TrialRms:8:6); writeln;
272        end; {if Trial}
273      end; {for J}
274    end; {for I}
275    if LowestRms<>CurrentRms
276    then begin
277      CurrentRms:=LowestRms;
278      for I:=1 to 2 do
279      begin
```

```
280         if Change[I]=false then CurrentInc[I]:=sqrt(CurrentInc[I]);
281         Current[I]:=Best[I];
282       end {for}
283     end; {if}
284   end; {sub procedure Adjust2}
285
286   procedure CalcBounded; {Control the curve fitting}
287   begin
288     Current[1]:=Guessed[1]; Trial[1]:=Guessed[1];
289     Current[2]:=Guessed[2]; Trial[2]:=Guessed[2];
290     Test;
291     CurrentRms:=TrialRms;
292     writeln('CurrentRms = ',CurrentRms:8:4); OldRms:=CurrentRms;
293     for I:=1 to 2 do
294     begin CurrentInc[I]:=2; TrialInc[I]:=CurrentInc[I]; end;
295     repeat
296       Adjust2;
297       if OldRms=CurrentRms then
298       begin
299         writeln('  Current factors: ');
300         for I:=1 to 2 do
301         begin
302           case I of
303             1: write('T2: ');
304             2: write('Image well distance: ');
305           end; {of cases}
306           CurrentInc[I]:=sqrt(CurrentInc[I]);
307           writeln(CurrentInc[I]:8:6,'  ');
308         end; {for}
309         writeln; writeln;
310       end {if}
311       else
312         OldRms:=CurrentRms;
313     until (CurrentRms<0.002) or
314       ((CurrentInc[1]<1.01) and (CurrentInc[2]<1.01));
315   end; {sub procedure CalcBounded}
316
317 begin {# Main part of procedure BoundFit3}
318   TextColor(Green); ClrScr;
319   writeln('Please enter the known values for:');
320   write('Transmissivity? '); Trans:=ReadReal(2);
321   write('Storage Coeficient? '); StorCoef:=ReadReal(2);
322   writeln('Please enter guesses for the following:-');
323   writeln;
324   write('Transmissivity on the far side of the boundary? ');
325   Guessed[1]:=ReadReal(2);
326   write('Distance of the image well? '); Guessed[2]:=ReadReal(2);
327   writeln('Calculating');
328   CalcBounded;
329   writeln; writeln('Best fit values are:');
330   for I:=1 to 2 do
331   begin
332     case I of
333       1: writeln('T2 = ',Current[I]:9:2);
334       2: writeln('Distance to image well = ',Current[I]:8:1);
335     end; {of cases}
336   end;
337   writeln('Final root-mean-square error is ',CurrentRms:9:5);
338    for I:=1 to 2 do Trial[I]:=Current[I]; Test; {Get best TrialDdVec data}
```

```
339   writeln('Press any key to continue.');
340   Ch:='x'; repeat read(kbd,Ch) until Ch<>'x';
341 end; {END OVERLAY Procedure BoundFit3}
342 {#----- End of the BoundFit procedures -----#}
343 {#----- End of Include file BoundFit.prc -----#}
```

Appendix A

DISK DATA FILE FORMAT

All the programs described in this book are capable of reading and/or writing data to/from a disk file with one of two standard formats. Discharge test data files in the ASCII format are indicated by a file name extension of '.WTD' (Well Test Data). Such files may also be produced by an ASCII editor such as that in Turbo Pascal, although program DTDHA is specifically designed to allow ease of entry of data from the keyboard.

The second form in which data may be stored on disk file is in Turbo Pascal machine language floating point numerical code. Data files in this format have the file name extension '.FTD' (Fast Test Data). While this form has the advantage of being faster to read and write, it has the disadvantage of not being so transportable. For example an FTD file written by a computer with a 8087 maths co-processor cannot be read by another computer without the co-processor because Turbo uses a different numerical format in each case.

In any case, the data are in the form of the same series of numbers, it is the coding of the numbers that varies. (Actually there is an exception in the way in which the four descriptive codes at the beginning of the file are stored in a Fast file. As this consists of records which themselves consist of three numbers each, the first two records are used to store the four descriptive codes.) An example of an ASCII file is given below.

```
1            2                  1100.000          18
        1.000          0.000       -514.000
        2.000          0.000       -514.000
        3.414          0.000       -514.000
        5.414          0.000       -514.000
        8.243          0.001       -514.000
       12.243          0.002       -514.000
       17.899          0.004       -514.000
       25.899          0.002       -514.000
       37.213         -0.000       -514.000
```

53.212	-0.002	-514.000
75.839	-0.005	-514.000
107.839	-0.012	-514.000
153.092	-0.015	-514.000
217.090	-0.025	-514.000
307.597	-0.030	-514.000
435.592	-0.045	-514.000
616.605	-0.065	-514.000
872.595	-0.080	-514.000

The data are arranged in columns and rows. The first row has four data, and all subsequent rows have three data each. All columns are 14 characters in width, so the first column starts at position 1 and goes to 14, the others are 15-28, 29-42, and 43-56 respectively.

In the first row:

The first item is a code number representing the type of test that the data apply to, one indicates a discharge test, two a recovery test, and three a simulation.

The second item is a code for the type of well, one indicates a pumped well, and two indicates a piezometer or observation well.

The third item is either the distance from the pumped well, if the data apply to a piezometer; or the effective diameter of the pumped well, if the data apply to a pumped well.

The third item is the number of time/drawdown/discharge rate sets that are to follow.

In all rows after the first:

The first item is the time in minutes for the drawdown and discharge rate which are to follow.

The second item is the drawdown.

The third item is the discharge rate as it was at the given time.

The actual data in this example are artificial. The negative discharge rates indicate that there was recharge rather than discharge.

Appendix D explains how WTD data files can be produced using the Lotus 1-2-3 spread sheet program.

Appendix B

THE USE OF TURBO PASCAL VERSION 3.XX

The programs of this book were written and compiled using Turbo Pascal versions 3.01A (without 8087 support) and 3.02A (with 8087 support). Version 3.x had several limitations regarding the size of a program and the practicalities of compilation which made the use of overlays and chaining necessary.

1. MAJOR SEGMENTS OF SOURCE AND OBJECT CODE

1.1. Include files

Editing a program of greater than 1000 lines can become a little slow because of the time taken to move from one part to another with the Turbo Pascal editor. If some part of the program is known to be debugged then that part may be used in the form of an include file where it occupies only one line of the source code file.

In the GW set of programs there is a certain amount of code common to all the programs. All the programs contain the Include file FIRST.SEG, and almost all contain either READ.PRC, SAVE.PRC or READSAVE.PRC.

A more compelling reason for using include files is that the Turbo editor is limited to a maximum of 64 kilobytes of code in memory at one time, several of the programs in this book go beyond that limit.

Include files allow code to be kept out of sight on disk until it is required at compile time, when the Turbo compiler brings it into memory.

1.2. Overlays

Just as Turbo Pascal version 3.x is limited to a maximum of 64k of source code in memory at one time, so it is limited to 64k of object code at one time. There are two ways around this limit, the use of overlays or the use of chain files. Any blocks of code that will not be required simultaneously can be placed in an overlay, and Turbo will bring them from disk into memory as they are required at run time. This technique has been used in programs DTDHA and ANALYZE. One disadvantage with the use of overlays is that during debugging it is necessary to compile all of the code every time one wishes to do a test run.

1.3. Chain files

Chain files are virtually separate compiled programs that can be called from either other Chain files, or from a Com file. All Chain files in a set must relate to one Com file; it is the Com file that contains the Turbo library of standard functions and procedures. The disadvantage in the use of Chain files is that all procedures and functions (other than the Turbo library) required by a program must be included in each Chain file. The sharing of data between Chain files is possible, but was not done by the GW set of programs.

GWSTART is the Com file of this set, all other Pascal language files of this book are compiled as Chain files.

2. CONVENTIONS USED

2.1. Indentation of source code

The indenting of the source code in this book follows the rules:

1/ Code between Begin and End, or between Repeat and Until, etc. is indented two spaces.

2/ Code between 'If x Then Begin' and 'End' and between 'Else Begin' and 'End' is normally indented two spaces, eg.

```
if X=Y
then begin
  Z:=0;
end
else begin
  Z:=1;
end;
```

2.2. Capitalisation of key words

All variable names, type names, procedure names, etc. which are declared in the programs are capitalised. Names of standard Turbo Pascal functions and procedures are not capitalised, unless they consist of two or more root words, in which case capitalisation makes them clearer. eg. 'write' and 'begin' are not capitalised, while 'GotoXY' and 'TextColor' are. There are one or two exceptions to the capitalisation of standard key words consisting of two or more root words, eg. the standard Pascal procedure name 'writeln' is not capitalised in these programs.

Appendix C

ERROR MESSAGES

Two types of error message are possible. The first type is that given due to the GW set of programs detecting a problem, or potential problem, with the data and objecting to it. This type of error message is very specific to the operation being carried out at the time the message arose. In most cases the message should be self explanatory (eg. an invalid entry of alphabetical characters when numerical were expected), but in some cases the notes on the program will have to be referred to (eg. if the Sternberg procedure objects to the form of your data).

The second type of error message is due to Turbo Pascal detecting a problem missed by the programs (or outside of the realm of the programs) and causing a return to DOS. The second type of error message should be rare and the errors are explained in the Turbo Pascal manual. For those who do not have a Turbo Pascal manual and are using the compiled programs as they were supplied on disk, the Turbo Pascal error numbers and their explanations will be given here. Comments in round brackets are mine.

1. RUN TIME ERRORS

01 Floating point overflow.

02 Division by zero attempted.

03 Sqrt argument error. The argument passed to the Sqrt function was negative.

04 Ln argument error. The argument passed to the natural logarithm function was zero or negative.

10 String length error.

11 Invalid string index.

90 Index out of range. The index expression of an array subscript was out of range.

91 Scalar or subrange out of range. The value assigned to a scalar or a subrange variable was out of range -32768 ... 32767.

92 Out of integer range. The real value passed to Trunc or Round was not within the Integer range -32768 ... 32767.

F0 Overlay file not found. (This message will be given if an overlay file which Turbo attempted to load into memory was not on the current drive or directory, and also not on the DOS path. Please refer to the Preliminary section, and/or to your DOS manual.)

FF Heap/stack collision. (This message will be given if your computer's memory is insufficient for the current demands upon it. If it occurs you may be able to reduce memory demands by one or more of the following:

1/ Reduce the number of memory resident programs in your computer.

2/ Change your CONFIG.SYS file to produce a smaller RAM disk, or no RAM disk at all.

If you do not have any memory resident programs or RAM disk then you may need to purchase more memory. On a PC XT, or AT there is no point in going beyond 640 kilobytes for the purposes of these programs.)

2. INPUT/OUTPUT ERROR MESSAGES

01 File does not exist.
02 File not open for input.
03 File not open for output.
04 File not open.
10 Error in numeric format.
20 Operation not allowed on a logical device.
21 Not allowed in direct mode.
22 Assign to std files not allowed.
90 Record length mismatch.
91 Seek beyond end-of-file.
99 Unexpected end of file. (This error may be encountered under one of the following conditions:
1/ You have produced an ASCII data file by some method other than with the programs of the GW group, and the number of records as indicated by the forth value on the first line of the file is greater than the actual number of records.
2/ A computer without a 8087 maths co-processor attempted to read a FTD data file produced by a computer with a 8087, or vice versa.)
F0 Disk write error. Disk full while attempting to expand a file. (This would be very frustrating if it occurred after spending some time on typing data into the computer. Either make sure that the disk to which you intend to save your data has plenty of spare space, or periodically save the data while entering it.)
F1 Directory is full. (The directory on a disk has room for only a certain number of files, you cannot place more files in a particular disk than this number. The exact number is dependent upon the operating system.)
F2 File size overflow.
F3 Too many open files.
FF File disappeared. An attempt was made to Close a file which was no longer present in the disk directory, e.g. because of an unexpected disk change.

Appendix D

CONVERTING A DATA FILE FROM LOTUS 1-2-3 TO WTD FORMAT

The data must be placed in the Lotus spread sheet in the format given in Appendix A with the column width for all four columns set at 14. Take especial care that the (first and only) figure in the forth column is the number of rows of time/drawdown/discharge rate data that follow.

The print command is given and the following points taken care of:

1/ The four columns of data are all highlighted for the full length of the first three columns;

2/ The width of the margin must be set to zero;

3/ Output must be to disk rather than to printer.

4/ The file name given must have the extension '.WTD'.

It does not seem to be possible to stop Lotus from paginating the data, ie. leaving a few blank lines in every 66 file lines. These blank lines must be removed with an ASCII editor program. With your editor, go through the file and carefully erase each blank line. If you use the Turbo Pascal editor, which is ideal for the purpose, then place the cursor on each blank line in turn and hold down Ctrl while pressing 'Y'. Save the data again before quitting Turbo.

The data should now be in a form readable by the GW set of programs.

Lotus 1-2-3 is a trademark of Lotus Development Corporation.

Epilogue

Some readers may wonder why there are no flow diagrams in this book. This omission was quite deliberate as I felt that they would not justify the space required to print them and the time taken to generate them. When producing flow diagrams it is difficult to know when to stop; a complete diagram could easily become as large as the program itself, and also equal in complexity. The flow of a part of a program having particular interest can be traced from the code with a little effort. It is left to the reader to produce special purpose flow diagrams as required.

Similar comments could be made about the relatively small number of remarks in the code. Remarks can easily take up as much memory as the source code itself, and several of these programs already push the memory requirements to the maximum. Also, every time a program is modified and listed on a printer the remarks are listed with it, making for unnecessary bulk. It is hoped that the explanations in the text of this book will fill the gap.

Turbo Pascal version 4 came onto the Australian market a short time before this book was finished. What I have seen of version 4 is only enough to discover that the source code of these programs will not compile unmodified using the new version. I still await the delivery of my copy of version 4 so that I can see the amount of modification required; I do not expect it to be very great.

At the present moment it is my intention to go on developing the programs of this book, but exactly what form that development will take I cannot be sure. A simple conversion to suit T-P version 4 is a possible first step. Complete rewriting in the C language (Turbo C?) is not out of the question.

Index

proc. indicates a procedure in the file which is named in capitals
func. indicates a function in the file which is named in capitals

This Page Intentionally Left Blank