LBNL-3698E

iTOUGH2 Universal Optimization Using the PEST Protocol

User's Guide

Stefan Finsterle

Earth Sciences Division
Lawrence Berkeley National Laboratory
University of California
Berkeley, CA 94720

July 2010

This work was supported, in part, by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Wind and Geothermal Technologies, of the U.S. Department of Energy, and as part of the Subsurface Science Scientific Focus Area funded by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Resources under Contract Number DE-AC02-05CH11231.

DISCLAIMER

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PAGE INTENTIONALLY LEFT BLANK

1. INTRODUCTION

iTOUGH2 (http://www-esd.lbl.gov/iTOUGH2) is a computer program for parameter estimation, sensitivity analysis, and uncertainty propagation analysis [*Finsterle*, 2007a, b, c]. iTOUGH2 contains a number of local and global minimization algorithms for automatic calibration of a model against measured data, or for the solution of other, more general optimization problems (see, for example, *Finsterle* [2005]). A detailed residual and estimation uncertainty analysis is conducted to assess the inversion results. Moreover, iTOUGH2 can be used to perform a formal sensitivity analysis, or to conduct Monte Carlo simulations for the examination for prediction uncertainties. iTOUGH2's capabilities are continually enhanced.

As the name implies, iTOUGH2 is developed for use in conjunction with the TOUGH2 forward simulator for nonisothermal multiphase flow in porous and fractured media [*Pruess*, 1991]. However, iTOUGH2 provides FORTRAN interfaces for the estimation of user-specified parameters (see subroutine USERPAR) based on user-specified observations (see subroutine USEROBS). These user interfaces can be invoked to add new parameter or observation types to the standard set provided in iTOUGH2. They can also be linked to non-TOUGH2 models, i.e., iTOUGH2 can be used as a universal optimization code, similar to other model-independent, nonlinear parameter estimation packages such as PEST [*Doherty*, 2008] or UCODE [*Poeter and Hill*, 1998]. However, to make iTOUGH2's optimization capabilities available for use with an external code, the user is required to write some FORTRAN code that provides the link between the iTOUGH2 parameter vector and the input parameters of the external code, and between the output variables of the external code and the iTOUGH2 observation vector. While allowing for maximum flexibility, the coding requirement of this approach limits its applicability to those users with FORTRAN coding knowledge.

To make iTOUGH2 capabilities accessible to many application models, the PEST protocol [*Doherty*, 2007] has been implemented into iTOUGH2. This protocol enables communication between the application (which can be a single "black-box" executable or a script or batch file that calls multiple codes) and iTOUGH2. The concept requires that for the application model:

(1) Input is provided on one or more ASCII text input files;

(2) Output is returned to one or more ASCII text output files;

(3) The model is run using a system command (executable or script/batch file); and

(4) The model runs to completion without any user intervention.

For each forward run invoked by iTOUGH2, select parameters cited within the application model input files are then overwritten with values provided by iTOUGH2, and select variables cited within the output files are extracted and returned to iTOUGH2. It should be noted that the core of iTOUGH2, i.e., its optimization routines and related analysis tools, remains unchanged; it is only the communication format between input parameters, the application model, and output variables that are borrowed from PEST. The interface routines have been provided by *Doherty* [2007]. The iTOUGH2-PEST architecture is shown in Figure 1.

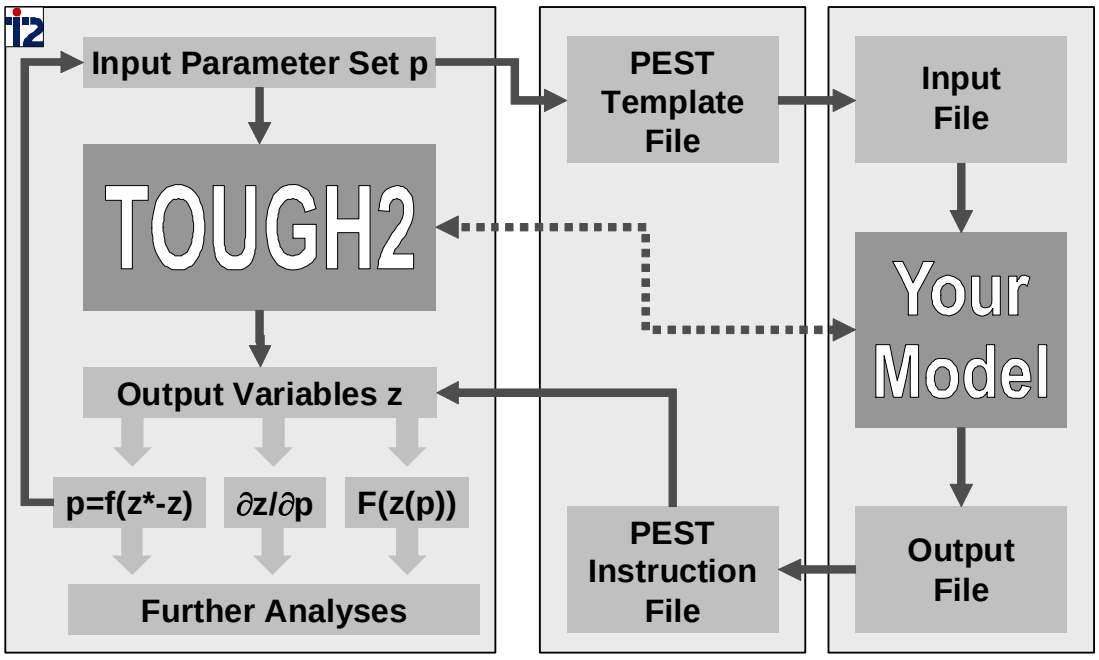


Figure 1. iTOUGH2-PEST architecture.

This manual contains installation instructions for the iTOUGH2-PEST module, and describes the PEST protocol as well as the input formats needed in iTOUGH2. Examples are provided that demonstrate the use of model-independent optimization and analysis using iTOUGH2.

2. INSTALLATION AND EXECUTION

Compilation and installation of iTOUGH2 is described in Section 5 of *Finsterle* [2007a] as well as in read-me files distributed with the code. To make the PEST protocol available in iTOUGH2, the following steps have to be performed:

(1) Edit file *it2stubs.f* and rename subroutines `INPEST`, `INITIALIZEPEST`, `UPDATEPEST`, `OBSERVATPEST`, and `FINALIZEPEST` (e.g., by adding an "`x`" at the end of the subroutine name); recompile *it2stubs.f*.

(2) Adjust maximum array dimensions in file *maxsize.inc*. If iTOUGH2 is intended to be used for non-TOUGH2 models only, memory can be saved by minimizing TOUGH2-related parameters, specifically `MAXEL`, `MAXCON`, `MAXSS`, `MAXR`, `MAXTIMES`, etc. A run-time error message will be issued if arrays are insufficiently dimensioned.

(3) Compile file *it2pest.f* and *mio.f90* and link them to the standard iTOUGH2 object files to create the executable. (On Unix platforms, you may use the script `it2make` with option `-pest`, or you may edit file *Makefile*, assign object files *it2pest.$(EXO)* and *mio.$(EXO)* to the environment variable `SPECIAL`.)

(4) Even if iTOUGH2-PEST is to be used for non-TOUGH2 models only, an equation-of-state (EOS) module needs to be selected and linked to the code, even though the TOUGH2 simulation part will be bypassed during execution of iTOUGH2-PEST. There are two options:

   a. Any regular EOS module can be selected for compilation and linking. During execution, a dummy TOUGH2 input file needs to be provided, with the keyword `PEST` in the first line.

   b. The special EOS module *eospest.f* can be compiled and linked, in which case no TOUGH2 input file is needed during execution.

Like any iTOUGH2 run, iTOUGH2-PEST reads in two main input files: an iTOUGH2 and a TOUGH2 input file. The latter is required even if no TOUGH2 forward model is used, in which case a dummy TOUGH2 input file must be provided, containing the word `PEST` in Columns 1–4 of the first line.

On Unix platforms, the script file `itough2` is used, which copies all input and data files to a temporary directory. The PEST template and instruction files, as well as the executable or script file used to run the forward model, also need to be copied to the temporary directory. This is achieved by simply adding lines with the statement "`FILE: filename`" (one line for each PEST file) anywhere in the iTOUGH2 input file.

3. THE PEST PROTOCOL

The PEST protocol is described in full in *Doherty* [2008]. (Note that programs of the USGS JUPITER suite [*Banta et al.*, 2008] use the same protocol.) The following subsections provide a brief overview of the key concepts and features. For a detailed description of the interface, the reader is referred to Section 3 of *Doherty* [2008].

3.1 General Concept

For iTOUGH2 to be model-independent, the communication between the optimization routines and the forward model must be general and defined with a clear, simple interface, across which parameter values (updated by iTOUGH2) and resulting model output at the calibration points (calculated by the forward model) are exchanged. Moreover, iTOUGH2 must be able to call the forward model (or series of forward models) using a simple system command. Finally, the forward model must run without user intervention. These conditions can be described in more detail as follows:

(1) The input file or files that contain the parameters to be estimated or modified by iTOUGH2 must be ASCII text files. Note, however, that models that receive their input directly from the user through keyboard entry can also be used, as keyboard inputs can be typed ahead of time into a file, and the model can be directed to look to this file for its input using "`< filename`" on the model command line, which indicates redirection of standard input from the keyboard to the specified file.

(2) A so-called template file (see detailed description in Section 3.2) is used to identify which input variables are subject to parameter estimation.

(3) The output file that contains the model calculated values of the observable variables that will be compared to the corresponding measured data needs to be an ASCII text file. Note, however, that models that use screen output can also be used by using "`> filename`" on the model command line, which indicates redirection of standard output from the screen to the specified file.

(4) A so-called instruction file (see detailed description in Section 3.3) is used to extract output variables at calibration points.

(5) The code must be capable of being run using a system command. The code itself can be written in any programming language; no access to the source code is required.

(6) This system command can be (a) the name of an executable (e.g., `model.exe`), (b) an executable with arguments (e.g., "`model.exe < input > output`"; or "`./model –i input –o output`") or (c) a batch or script file that executes multiple, linked programs (e.g. "`run-models.bat`" or "`cat run-models | sh`").

(7) The code must be capable of being run to completion without interactive user intervention (see (1) above for comment about keyboard input).

3.2 Template File

iTOUGH2 requests a model evaluation each time a new parameter vector is proposed. The updated parameter values need to be written to the model input files which hold them. A template file is used to identify those parameters in the input file that iTOUGH2 is allowed to vary. A template file is a replica of a model input file except that the space occupied by each variable parameter is replaced by a sequence of characters which identify the space occupied by that parameter. A template file is required for each input file that contains one or more parameters to be adjusted for model calibration, sensitivity analysis, or uncertainty propagation analysis.

To construct a template file, a copy of the model input file should be made, and each space occupied by a parameter needs to be replaced by a set of characters that both identify the parameter and define its location (i.e., position and width) in the input file. A parameter is identified by a unique name of up to twelve characters in length. A given parameter can be referenced once or many times, i.e., a single parameter value to be estimated may refer to multiple input variables. For each simulation run invoked by iTOUGH2, the template is copied to the model input file, and each parameter space is replaced with the appropriate parameter value.

A model input file can be of any length. However, a line cannot be longer than 2000 characters. The same applies to template files. It is suggested that template files be provided with the extension "*.tpl*" in order to distinguish them from other types of files. Figure 2 shows a simple input file, and Figure 3 is the corresponding template file. The meaning and purpose of each of the numbers in the input file are of no relevance here, as they are also unknown to iTOUGH2. We assume that the first three real values (in rows 2–4) hold three coefficients that we would like to estimate using iTOUGH2.

Figure 3 shows, the first line of a template file must contain the letters "ptf" (which stands for "PEST template file"), followed by a space, followed by a single character. The character following the space is the "parameter delimiter". In a template file, a "parameter space" is identified as the set of characters between and including a pair of parameter delimiters. When iTOUGH2 writes a model input file based on a template file, it replaces all characters between and including these parameter delimiters by a number representing the current value of the parameter that owns the space; that parameter is identified by name within the parameter space, between the parameter delimiters.

The parameter delimiter can be any (special) character *except* [a-z], [A-Z] and [0-9]. Moreover, the parameter delimiter character must not appear anywhere within the template file except in its capacity as a parameter delimiter.

All parameters are referenced by name. The parameter names in template files (where the locations of parameters on model input files are identified) must be identical to those in the > PARAMETER block of the iTOUGH2 input file (which is the equivalent of the PEST control file). Parameter names can be from one to twelve characters in length, any characters being legal except for the space character and the parameter delimiter character. Parameter names are case-insensitive. Each parameter space is defined by two parameter delimiters; the name of the parameter to which the space belongs must be written between the two delimiters. The minimum allowable parameter space width is thus three characters: one character each for the left and right delimiters and one for the parameter name.

Section 3.3 of *Doherty* [2008] contains more details about the template file. The PEST utility program TEMPCHEK can be used to check whether the template file obeys the PEST protocol, and

to generate a model input file from a template file given a set of parameter values (see Section 10.1 of *Doherty* [2008]).

```
2
 0.5000000E+00
 2.0000000E+00
 1.5000000E+00
5
0.25
0.50
1.00
1.50
2.00
```

Figure 2. Example model input file.

```
ptf #
2
#coeff0      #
#coeff1      #
#coeff2      #
5
0.25
0.50
1.00
1.50
2.00
```

Figure 3. Template file corresponding to model input file of Figure 2.

3.3 Instruction File

The instruction file directs the interface to identify and extract from one or multiple ASCII output files those "observable" values for which a corresponding measured value (or target value) is available. Since output files often change from run to run, the simple template concept used for locating variables in an input file (see Section 3.2) cannot be applied; instead, a relatively small set of basic search directives for locating fields in the output file are provided in the instruction file.

Details about the concept and search directives of an instruction file can be found in *Doherty* [2008]. Here, we provide a simple example (Figure 5) and summarize the search directives (Table 1). The basic concept is that each output file is parsed line by line from top to bottom, until a *Primary Marker* is identified. *Line advances* and *Secondary Markers* are used to find further reference points and eventually to locate an observation.

```
Simulation Output File
======================
Iteration No. 1    Time = 0.2 years
...
Iteration No. 5    Time = 1.0 years
    Depth          Pressure
    1.00           1.21072
    2.00           1.51313
    3.00           2.07536
    4.00           2.95097
    5.00           4.19023
    6.00           5.87513
    7.00           8.08115
```

Figure 4. Example model output file.

```
pif @
@Iteration@  @1.0 years@
l2 [pres1]21:27
l1 (pres2)11:25
l1 t20   !pres3!
@ 4.00 @ !pres4!
l1 w w   !pres5!
l2 !dum! !pres7!
```

Figure 5. Example instruction file related to the output file of Figure 4.

The first line of an instruction file must begin with the three letters "pif" (which stand for "PEST instruction file"), followed, after a single space, by a single character, the marker delimiter. The role of the marker delimiter in an instruction file is to define the extent of a marker; a marker delimiter must be placed just before the first character of a text string comprising a marker and immediately

after the last character of the marker string. The text between a pair of marker delimiters is not interpreted as a list of search instructions, but is used as a marker.

A ***Marker Delimiter*** must not be one of the characters A–Z, a–z, 0–9, !, [,], (,), :, or &. The marker delimiter must not occur within the text of any markers.

Each observation must be provided with a unique name. An ***Observation Name*** must be 20 characters or less in length. These 20 characters can be any ASCII characters except for [,], (,), or the marker delimiter character. These same observation names must also be cited in the `> OBSERVATION` block of the iTOUGH2 input file.

Each observation name is unique and thus may occur only once in all instruction files. There is one observation name, however, to which these rules do not apply—the dummy observation name "`dum`"—which is simply a mechanism for model output file navigation. It may occur many times, if necessary, in an instruction file. While "`dum`" is used to located a field in an output file, the value is not extracted and passed to iTOUGH2 as an observation

If a number of instruction items appear on a single line of an instruction file (see Figure 5), these items must be separated from each other by at least one space. Instructions pertaining to a single line on a model output file are written on a single line of an instruction file. Thus, the start of a new instruction line signifies that at least one new model output file line must be read. However, if the first instruction on the new line is the character "`&`", the new instruction line is simply a ***Continuation*** of the old one.

Unless it is a continuation of a previous line, each instruction line must begin with either of two instruction items: a primary marker or a line advance item. The ***Primary Marker*** is a string of characters, bracketed by a pair of marker delimiter characters. If a marker is the first item on an instruction line, then it is a primary marker; if it occurs later in the line, following other instruction items, it is a secondary marker. On encountering a primary marker in an instruction file, the model output file is read line by line, searching for the string between the marker delimiter characters. When the string is found, the "cursor" is placed at the last character of the string; further instructions pertain to the parts of the model output file line following the string identified as the primary marker.

The syntax for the ***Line Advance*** item is "`l`*n*" or "`L`*n*" where *n* is the number of lines to advance.

A ***Secondary Marker*** is a marker which does not occupy the first position of an instruction line. It moves the cursor along the current model output file line until it finds the secondary marker string, and places its cursor on the last character of that string. If the secondary marker is not found, the line is advanced.

The ***Whitespace*** instruction is a simple "`w`", separated from its neighboring instructions by at least one blank space. The instruction moves the cursor forward from its current position until it encounters the next blank character, and then moves the cursor forward again until it finds a nonblank character, finally placing the cursor on the blank character preceding this nonblank character.

The ***Tab*** instruction places the cursor at a user-specified character position (i.e., column number) on the model output file line which is currently processed. The instruction syntax is "`t`*n*" where *n* is the column number.

A ***Fixed Observation*** can be found between, and including, columns *n1* and *n2* on the model output file line on which the cursor is currently resting. The instruction item for a fixed observation consists of two parts. The first part consists of the observation name enclosed in square brackets, while the second part consists of the first and last columns from which to read the observation. No space must separate these two parts of the observation instruction.

Semi-Fixed Observations are located by two numbers identifying two column numbers *n1* and *n2*. An observation is read if it is fully contained, starts, or ends somewhere between columns *n1* and *n2* However, reading of an observation fails if no non-blank characters are found in the space between the column numbers, or if more than one space-separated word is found. Note that the width of the observation value can be greater than the difference between the column numbers cited in the semi-fixed observation instruction. The instruction item for a semi-fixed observation consists of two parts. The first part consists of the observation name enclosed in parentheses, while the second part consists of the two column numbers, separated by a colon. There must be no space separating these two parts of the semi-fixed observation instruction.

A ***Non-Fixed Observation*** instruction does not include any column numbers because the number is found using secondary markers and/or other navigational aids such as whitespace and tabs which precede the non-fixed observation on the instruction line. The non-fixed observation is read as a free-format number following the current cursor position. The end of the number is indicated by a blank character, end of line, or the first character of a secondary marker. A non-fixed observation is represented by the name of the observation surrounded by exclamation marks.

It is suggested that instruction files be provided with the extension "*.ins*" in order to distinguish them from other types of files. The instruction file can be checked for syntactical correctness and consistency using the PEST utility program INSCHEK (see Section 10.2 of *Doherty* [2008]).

Table 1. Summary Table of Search Directives in Instruction File

Instruction Item	Description	Example Instruction
General	Output file parsed from top to bottom; lines from left to right. Instruction lines must start either with a Primary Marker, a Line Advance, or the continuation character "&". If a number of instruction items appear on a single line of an instruction file, these items must be separated from each other by at least one space. Instructions pertaining to a single line on a model output file are written on a single line of an instruction file.	`@OUTPUT@ w !obs1!` `& w w !obs2!`
First line	Keyword and marker delimiter	`pif @`
Marker Delimiter	A marker delimiter must not be one of the characters A–Z, a–z, 0–9, !, [,], (,), :, or &. The marker delimiter must not occur within the text bracketed by any markers.	`pif @`

Observation Name	Unique name identifying observation;
	maximum 20 characters long;
	any ASCII characters except for [,], (,), or the marker delimiter character.
		`arg1`
		`y2`
		`obs3`
		`pressure_at_X=4`
		`conc-after-5-year`

Dummy Observation	Dummy observation can be used to navigate line by reading non-fixed observations; however, values are not extracted. The observation name for dummy variables must be `dum`.	`l1 !dum! !dum! !sat!`
Primary Marker	Marker at beginning of instruction line. Bracketed by Marker Delimiter	`@OUTPUT@`
Line Advance	At beginning of instruction line. L*n* advances by *n* lines	`l1` `L56`
Secondary Marker	Marker that does not occupy first instruction item Searches within current line from left to right. Advances to next line of not found.	`@OUTPUT@ @TIME IS@`

Table 1. (cont) Summary Table of Search Directives in Instruction File

Whitespace	Moves cursor forwards from its current position until it encounters the next blank character, and then moves the cursor forward again until it finds a nonblank character, finally placing the cursor on the blank character preceding this nonblank character.	`@DEPTH =@ w w !p!`
Tab	Places cursor at a user-specified character position on current model output file line.	`@DEPTH =@ t56 !p!`
Fixed Observation	Reads observation between columns *n*1 and *n*2. Observation name in brackets; column numbers separated by colon; no spaces.	`l1 [pres]13:25`
Semi-Fixed Observation	Reads observation that is contained, starts, or ends somewhere between columns *n*1 and *n*2. Observation name in parentheses; column numbers separated by colon; no spaces.	`l1 (pres)19:20`

Non-Fixed Observation	Reads observation in free format at current location.
	Observation name between exclamation points.
		`l8 w !pres!`
		`l8 !dum! !dum! !pres!`
		`l5 *=* !sat! *%*`

4. iTOUGH2-PEST INPUT FORMATS

4.1 Introduction

The PEST protocol described in Section 3 provides the interface between the vector of input parameters adjusted by iTOUGH2 and the (non-TOUGH2) forward model (or set of models), and between select output values calculated by this model and the iTOUGH2 vector of observable variables. The usage and format of the template and instruction files needed to facilitate this data exchange between iTOUGH2 and the forward model are identical to those used by PEST and UCODE. However, the actual analysis of these parameters and observations is done using the iTOUGH2 capabilities, i.e., all input and output options, the local and global optimization algorithms, the sensitivity, residual, error, and uncertainty analyses commonly done for TOUGH2 models are performed for the user-provided model (or series of models).

While all relevant commands described in the iTOUGH2 Command Reference [*Finsterle*, 2007b] are available, new commands are needed to identify the universal parameter and universal observations, and to specify the model executable, template and instruction files.

4.2 Generic iTOUGH2-PEST Input File

Figure 6 shows the syntax of the new PEST-related commands in a generic template; only the new or necessary commands are shown; complete examples can be found in Section 5. Unlike in iTOUGH2 applications that use one of the TOUGH2 modules [*Pruess et al.*, 1999] as the forward model, no knowledge about the potential types and properties of the parameters to be adjusted is available to the code if a non-TOUGH2 model is used. Therefore, a generic parameter-selection command `>> PEST` is used in the `> PARAMETER` block to signify that the parameter refers to an unknown, user-supplied forward model, and will be adjusted through the PEST template file (see Section 3.2). The name of the parameter is then supplied through the `>>>> ANNOTATION` command. This (case-insensitive) name has to be identical to the parameter name in the template file. Since the parameter does not refer to a TOUGH2 material name or model region, the third-level command `>>> NONE` must be used. An initial value of the parameter must be given either through the `>>>> GUESS` or `>>>> PRIOR` command. All other fourth-level commands of the `> PARAMETER` block are also available for further parameter specification.

Similarly, since observations are taken from the output file of a model that is unknown to iTOUGH2, the definition of calibration points as points in space and time is not applicable, and a more general (less controllable) approach must be taken. As a first consequence, the `>> TIMES` block (which is essential in regular iTOUGH2 applications) is not needed or used for identifying observations that refer to a non-TOUGH2 model (it is still needed when combining PEST-type and TOUGH-type observations in a single inversion). This does not mean that PEST-type observations do not refer to time, but it is up to the user to identify the observation time through appropriate parsing of the output file using the search directives in the instruction file (see Section 3.3).

```
> PARAMETER
  >> PEST
     >>> NONE
         >>>> NAME  : parameter-name
         >>>> GUESS : initial-parameter-value
         >>>> PRIOR : prior-information-value
              (GUESS or PRIOR is required)
         >>>> other fourth-level commands
         <<<<
     <<<
  <<

> OBSERVATION
  (TIMES block not required)
  >> PEST
     >>> UNIVERSAL/MODEL/NONE (: data-set-name)
         >>>> DATA
              observation-name-1    value-1    (weight-1)
              observation-name-2    value-2    (weight-2)
              observation-name-...  value-...  (weight-...)
         >>>> other fourth-level commands
         <<<<
     <<<
  <<

> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> TEMPLATE: number-of-template-files
              template-file-1.tpl       input-file-1
              ...                       ...
              template-file-ntpl.tpl    input-file-ntpl

         >>>> INSTRUCTION: number-of-instruction-files (NO DELETE)
              instruction-file-1.ins output-file-1
              ...                       ...
              instruction-file-ntpl.ins output-file-nins

         >>>> EXECUTABLE: executable-name  (BEFORE/AFTER)
         >>>> PRECISION : SINGLE/DOUBLE
         >>>> DECPOINT  : NOPOINT/POINT
         <<<<
     <<<
```

Figure 6. Generic iTOUGH2 input file with PEST-related blocks.

The observation type is also unknown; therefore, a generic type >> PEST is used to signify that the observation refers to an unknown user-supplied forward model, and will be read by means of the PEST instruction file.

The third-level command of block `> OBSERVATION` usually is used to identify the spatial point in the model that corresponds to the location where the measurements are taken. Again, iTOUGH2 has no notion of how points in space are defined in the unknown forward model. Therefore, a generic third-level command `>>> UNIVERSAL`, `>>> NONE`, or `>>> MODEL` is used to identify a data set. This data set may or may not consist of data that refer to a particular point in space. Nevertheless, it is recommended to structure data into data sets as some of the *a posteriori* analyses pertain to individual data sets. Each data set can be given a name (either on the third-level command line or through command `>>>> ANNOTATION`). This data set name should not be confused with the observation name used to extract individual values from the non-TOUGH2 output file.

Unique observation names need to be given for each data point. The measured values are provided through command `>>>> DATA` (either following this command line or on an external file; see syntax of command `>>>> DATA` for details). However, unlike in a regular iTOUGH2 data block, where a data line includes the measurement time, the measured value, and (optionally) the standard deviation of the measurement error, a PEST-type data line consists of (1) a unique observation name (consistent with the name given in the instruction file), followed by (2) the measured value, and (3) (optionally) the weight to be attached to the corresponding residual. If no weight is given in the third column, the corresponding iTOUGH2 fourth-level commands (`>>>> WEIGHT`, `>>>> DEVIATION`, `>>>> VARIANCE`, `>>>> RELATIVE`) can be used to assign weights to all observations in the corresponding data set. Note that this format is compatible with the way observation data are provided through the PEST control file (see Section 4.2.7 of *Doherty* [2008]), i.e., standard observation data blocks from a PEST control file can directly be used in iTOUGH2 input files (the name of the observation group given in the fourth column of a PEST data definition block is ignored, with similar functionality provided through other iTOUGH2 features).

The `> COMPUTATION`, `>> OPTION`, `>>> PEST` block is used to (1) identify the executable (or script or batch) file of the forward model, (2) relate the template file(s) to the corresponding input file(s), (3) relate the instruction file(s) to the corresponding output file(s), and (4) to specify the precision with which parameter values are to be written to the input file, and whether the value should include a decimal point (see Section 3.2.6 of *Doherty* [2008]).

On Unix systems, it is suggested that the keyword `FILE:` followed by a file name is provided in the iTOUGH2 input file lines, one line for each file used by the PEST interface. This ensures that the *itough2* script automatically copies all the needed files to the temporary directory.

4.3 PEST-Related iTOUGH2 Commands

The new, PEST-related iTOUGH2 commands are described on the following pages in the standard format of the iTOUGH2 Command Reference [*Finsterle*, 2007b]. They are also available at http://esd.lbl.gov/iTOUGH2/Command/cgmmand.html.

Command

```
>> PEST
```

Parent Command

```
> PARAMETER
```

Subcommand

```
>>> NONE
```

Description

This command signifies that a parameter related to a user-supplied model (i.e., not a TOUGH2 module) be selected. The parameter will be identified and updated using the template file of the PEST interface. An initial guess must be provided for all non-TOUGH2 parameters through commands `>>>> GUESS` or `>>>> PRIOR`. All PEST-related parameters must be specified before any TOUGH2-related are selected.

Example

```
> PARAMETER
  >> PEST
     >>> NONE
         >>>> NAME       : coefficient-A
         >>>> LOGARITHM
         >>>> GUESS      : -3.0
         >>>> RANGE      : -6.0 0.0
         <<<<
     <<<
  <<
```

See Also

```
>> PEST (o), >>> PEST
```

Command

```
>> PEST
```

Parent Command

```
> OBSERVATION
```

Subcommand

```
>>> NONE
>>> MODEL
>>> UNIVERSAL
```

Description

This command selects an (unknown) observation type related to a user-supplied model. The calculated value is identified and extracted from the output files using the search directives of a PEST instruction file. Unique observation names must be provided in the first data column, followed by the measured values, and (optionally) the weight attached to the residual. The case-insensitive observation names must be identical to those used in the instruction file(s). All PEST-related observations must be specified before any TOUGH2-related observations are selected.

Example

```
> OBSERVATION
  >> PEST
     >>> UNIVERSAL
         >>>> ANNOTATION  :   Total Costs
         >>>> DATA
              capital-cost    0.0
              operating-cost  0.0
         >>>> WEIGHT      :   1.00531 [dollar/CHF]
         <<<<

     >>> UNIVERSAL: pumping rates
         >>>> DATA
              pH-after-0-yr   7.2  1.0  pump
              pH-after-1-yr   5.8  0.5  pump
              pH-after-2-yr   3.6  0.5  pump
         <<<<
      <<<
```

See Also

```
>> PEST (p),  >>> PEST
```

Command

```
>>> PEST
```

Parent Command

```
>> OPTION
```

Subcommand

```
>>>> DECPOINT
>>>> EXECUTABLE
>>>> INSTRUCTION
>>>> PRECISION
>>>> TEMPLATE
```

Description

This command invokes fourth-level commands to specify files and options for the PEST interface between iTOUGH2, a user-supplied model, and its input and output files. It is used to (1) identify the executable (or script or batch) file that calls the forward model, (2) relate the template file(s) to the corresponding input file(s), (3) relate the instruction file(s) to the corresponding output file(s), and (4) to specify the precision with which parameter values are to be written to the input file, and whether the value should include a decimal point.

Example

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> TEMPLATE   : 1
              input.tpl    input.txt

         >>>> INSTRUCTION: 2
              cost.ins     cost.out
              pump.ins     pump.out

         >>>> EXECUTABLE : pumpcost.bat
         >>>> PRECISION  : SINGLE
         >>>> DECPOINT   : POINT
         <<<<
     <<<
  <<
```

See Also

```
>> PEST (o),PEST (p)
```

Command

```
>>>> EXECUTABLE: FILE (BEFORE/AFTER)
```

Parent Command

```
>>> PEST
```

Subcommand

-

Description

iTOUGH2 capabilities can be applied to non-TOUGH2 models using the PEST interface [*Doherty*, 2008]. The user-supplied model must be an executable that can be run from a system command prompt. The system command can also be a script or batch file, in which multiple models can be combined. The model must be able to be executed without user interference or intervention. It must get its input through one or multiple ASCII text files, and write its output to one or multiple ASCII text files. Communication between iTOUGH2 and the model's input and output occurs through PEST-style template and instruction files, respectively.

The user-supplied model can be run by itself, before (default) or after (keyword AFTER) a TOUGH2 run. The latter two options are useful for estimating parameters that relate to pre- or postprocessors of TOUGH2, respectively. The name of the executable, script, or batch file is provided following the colon. If the executable name contains spaces, the command line must be in quotes. Under Unix, it is recommended to add the keyword FILE to the command line or on a separate line, so the executable is automatically copied to the temporary directory. Examples include:

```
>>>> EXECUTABLE      : myModel.exe
>>>> EXECUTABLE      : Run-ModelA-and-ModelB.bat
>>>> EXECUTABLE FILE : UnixScript.sh  run AFTER TOUGH2
>>>> EXECUTABLE      : "a.out < input > output"
                 FILE : a.out
```

Example

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE FILE : this-is-not-TOUGH.exe
         >>>> TEMPLATE        : 1
              input.tpl          input.txt
         >>>> INSTRUCTION     : 1
              output.ins         output.txt
```

See Also

>>>> INSTRUCTION, >>>> TEMPLATE

Command

`>>>> TEMPLATE: num-template-files`

Parent Command

`>>> PEST`

Subcommand

-

Description

iTOUGH2 capabilities can be applied to non-TOUGH2 models using the PEST interface [*Doherty*, 2008]. Template files are used to communicate between iTOUGH2 parameters and the input variables of the user-supplied model which must be provided through one or multiple ASCII text files. (Note that if the model expects input from the keyboard, the '<' symbol can be used on the command line (see `>>>> EXECUTABLE`) to redirect standard input from the keyboard to a text file.)

A template file must be provided for each input file that contains a parameter adjusted by iTOUGH2; *num-template-files* is the number of template files provided. Template files are matched to their corresponding input files on the lines following the `>>>> TEMPLATE` command. Under Unix, it is recommended to add separate lines with the keyword `FILE:` followed by the file name, so the files are automatically copied to the temporary directory. This can occur anywhere in the iTOUGH2 input file.

Example

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE FILE : Run-ModelA-and-ModelB.bat
         >>>> TEMPLATE        : 2
              ModelA.tpl        inputA.txt
              ModelB.tpl        inputB.txt
         >>>> INSTRUCTION     : 1
              outputB.ins       outputB.txt
         <<<<
     <<<

copy FILE: ModelA.tpl  to temporary directory
copy FILE: ModelB.tpl  to temporary directory
copy FILE: outputB.ins to temporary directory
```

See Also

`>>>> EXECUTABLE`, `>>>> INSTRUCTION`

Command
`>>>> INSTRUCTION: num-instruction-files (NO DELETE)`

Parent Command
`>>> PEST`

Subcommand
-

Description
iTOUGH2 capabilities can be applied to non-TOUGH2 models using the PEST interface [*Doherty*, 2008]. Instruction files are used to communicate between the output variables of the user-supplied model (which must be provided through one or multiple ASCII text files) and the iTOUGH2 observation vector. (Note that if the model writes output to the screen, the '>' symbol can be used on the command line (see `>>>> EXECUTABLE`) to redirect standard output from the screen to a text file.)

An instruction file must be provided for each output file that contains an observable variable used by iTOUGH2 for model evaluation; *`num-instruction-files`* is the number of instruction files provided. Instruction files are matched to their corresponding output files on the lines following the `>>>> INSTRUCTION` command.

Upon initialization, all output files are deleted unless keyword `NO DELETE` is present.

PEST requires that at least one PEST observation is provided. However, if the external model is a preprocessor to TOUGH2, all observations may be internally taken from TOUGH2 arrays, without the need for an observation that is read from an external output file. Dummy instruction and output files can be automatically generated by providing their respective file names as *dummy.ins* and *dummy.out*. A "measured" value of zero will be generated and returned for comparison to an observation named `dummy` that needs to be defined in the iTOUGH2 input file.

Under Unix, it is recommended to add separate lines with the keyword `FILE:` followed by the file name, so the files are automatically copied to the temporary directory. This can occur anywhere in the iTOUGH2 input file.

Example

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE  : “idratherusetough.exe > nowwhat”
         >>>> TEMPLATE    : 1
              input.tpl     input.txt

         >>>> INSTRUCTION : 2
              output1.ins   nowwhat
              output2.ins   some_results.txt
         <<<<

copy FILE: input.tpl            to temporary directory
copy FILE: output1.ins          to temporary directory
copy FILE: output2.ins          to temporary directory
copy FILE: idratherusetough.exe to temporary directory
```

The following example shows the use of dummy instruction and output files:

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE  : TOUGH-pre-processor
         >>>> TEMPLATE    : 1
              meshgen.tpl   meshgen.txt

         >>>> INSTRUCTION : 1   NO DELETE
              dummy.ins     dummy.out
         <<<<
```

A dummy PEST observation needs to be provided as follows:

```
> OBSERVATION
  >> PEST
     >>> UNIVERSAL
         >>>> DATA
              dummy 0.0 1.0E-20
         <<<<
```

See Also

>>>> EXECUTABLE, >>>> TEMPLATE

Command

```
>>>> DECPOINT: POINT/NOPOINT
```

Parent Command

```
>>> PEST
```

Subcommand

-

Description

iTOUGH2 capabilities can be applied to non-TOUGH2 models using the PEST interface [*Doherty*, 2008]. By selecting the keyword `NOPOINT`, the decimal point in the representation of a parameter in the input file is omitted, potentially increasing the accuracy of a parameter value. However, this should be done with great caution, as fields read by FORTRAN programs that read fields using format specifiers such as "(F6.2)" or "(E8.2)" may insert a decimal point incorrectly if none is specified; for details, see Section 3.2.6 of *Doherty* [2008]. Therefore, `POINT` is the default option.

Example

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE  :     pointnopoint.exe
         >>>> TEMPLATE    : 1
              pointnopoint.tpl  whatsthepoint.in

         >>>> INSTRUCTION : 1
              pointnopoint.ins  pointless.out

         >>>> DECPOINT: NOPOINT
         <<<<
```

See Also

-

Command

```
>>>> PRECISION: SINGLE/DOUBLE
```

Parent Command

```
>>> PEST
```

Subcommand

-

Description

iTOUGH2 capabilities can be applied to non-TOUGH2 models using the PEST interface [*Doherty*, 2008]. The `>>>> PRECISION` command determines whether single or double precision protocol is to be observed in writing parameter values. Unless a parameter space is greater than 13 characters in width, it has no bearing on the precision with which a parameter value is written to a model input file, as this is determined by the width of the parameter space. If keyword `SINGLE` is selected, exponents are represented by the letter "e"; also, if a parameter space is greater than 13 characters in width, only the last 13 spaces are used in writing the number representing the parameter value, any remaining characters within the parameter space being left blank. If keyword `DOUBLE` is selected, up to 23 characters can be used to represent a number and the letter "d" is used to represent exponents; also, the double-precision range of real numbers is available.

Example

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE  : "precise.exe < imsingle.in"
         >>>> TEMPLATE    : 1
              precisely.tpl  imsingle.in

         >>>> INSTRUCTION : 1
              precisely.ins  nodifference.out

         >>>> DECPOINT: SINGLE
         <<<<
```

See Also

-

5. EXAMPLES

The examples in this section are all tutorial. Their purpose is to demonstrate the usage of iTOUGH2-PEST features; they are not designed to be useful, efficient or elegant, and are not intended to be of scientific value.

5.1 Polynomial Fitting Using iTOUGH2-PEST

In this example, iTOUGH2-PEST is used to estimate the coefficients of a polynomial. Consider the simple FORTRAN program of Figure 7. It prompts the user to enter the degree n of a polynomial

$$y(x) = \sum_{i=0}^{n} a_i \cdot x^i \tag{1}$$

The program then expects n+1 coefficients a_i, followed by the range [x_{min}, x_{max}] and number of points m for which the polynomial shall be evaluated. This information is entered by the user through the keyboard, and m pairs of points, x and $y(x)$, are displayed on the screen.

The FORTRAN source code is compiled and linked into an executable named *Polynomial.exe*. This executable needs to be copied to the working directory or added to the command search path. For this program to serve as the forward operator in an iTOUGH2-PEST inversion, the input (i.e., responses to the prompted question marks) must be pre-typed into a text file, here named *Polynomial.in* (see Figure 8). The corresponding screen output, redirected into a text file *Polynomial.out*, is shown in Figure 9.

The template file *Polynomial.tpl* (see Figure 10) is a copy of the input file, with the header line "`ptf #`" added, and the slots for the parameters to be estimated replaced by user-specified parameter names, bracketed by the parameter delimiter #.

Figure 11 shows the instruction file *Polynomial.ins* used to parse through the output file *Polynomial.out* and to find and extract the values $y(x)$. It starts with the keyword `pif @`, where `@` is the marker delimiter. It then searches for the primary marker "`y(x)`", arriving at the line before the output of interest is found. All the following lines start with the line advance command `l1`, which instructs the parser to read and interpret one line at a time.

For demonstration purposes, different search directives are used to identify the values to be extracted (see Table 1). The first value (i.e., the polynomial evaluated at x=x_{min} for the three coefficients a_0, a_1, and a_2 provided by iTOUGH2-PEST) is read as a fixed observation, i.e., the number is expected to be found between columns 18 and 34. The second value, named `y2`, is extracted using the semi-fixed observation format, i.e., it extracts the number that starts or ends within or spans columns 29 to 31. To find the third value, `y3`, two white spaces are bridged, and the next word is read as a free observation. Finally, values `y4` through `y21` are extracted by first reading (and discarding) a dummy observation (the reported x_i value), and then reading and extracting the result `yi` = $y(x_i)$ using the free-observation format.

Next, the PEST control capabilities of iTOUGH2 are selected through the iTOUGH2 input file.

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program polynomial
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C
C --- Evaluates polynomial
C
      dimension c(9)
C
C --- Read degree of polynomial, coefficients, x-value range and
C     number of evaluation points
C
      write(*,*) ' Evaluate Polynomial'
      write(*,*) ' ******************'
      write(*,*)
      write(*,*) ' Degree of polynomial : ?'
      read(*,*)  n
      do i=1,n+1
         write(*,7000) i-1
 7000 format('  Coefficient a',i1,7x,': ?')
         read(*,*) c(i)
      enddo
      write(*,*) ' Range of x: Xmin     : ?'
      read(*,*)  xmin
         write(*,*) '             Xmax     : ?'
      read(*,*)  xmax
      write(*,*) ' Number of points     : ?'
      read(*,*)   m
      dx=(xmax-xmin)/(max(m,2)-1)
C
C --- Calculate and output f(x)
C
      write(*,7001) n
 7001 format(/,/,'  Polynomial of degree',i2,' with coefficients:')
      do i=1,n+1
         write(*,7002) i,c(i)
 7002 format('  a(',i1,') =',f9.5)
      enddo
      write(*,*)
      write(*,*) '               x            y(x)'
      x=xmin
      do j=1,m
         y=c(1)
         do i=2,n+1
            y=y+c(i)*x**(i-1)
         enddo
         write(*,'(2f17.10)') x,y
         x=x+dx
      enddo
      end
```

Figure 7. FORTRAN program that evaluates a polynomial for a given set of coefficients.

2	Degree of polynomial, *n*
5.0	Coefficient a_0
4.0	Coefficient a_1
3.0	Coefficient a_2
-2.0	x_{min}
2.0	x_{max}
21	Number of points, *m*

Figure 8. Text file *Polynomial.in* providing input required by program *Polynomial.exe*.

```
 Evaluate Polynomial
 *******************

 Degree of polynomial : ?
 Coefficient a0       : ?
 Coefficient a1       : ?
 Coefficient a2       : ?
 Range of x: Xmin     : ?
             Xmax     : ?
 Number of points     : ?


 Polynomial of degree 2 with coefficients:
 a(1) =  5.00000
 a(2) =  4.00000
 a(3) =  3.00000

                x             y(x)
   -2.0000000000     9.0000000000
   -1.7999999523     7.5199995041
   -1.5999999046     6.2799992561
   -1.3999998569     5.2799992561
   -1.1999998093     4.5199995041
   -0.9999998212     3.9999997616
   -0.7999998331     3.7199997902
   -0.5999998450     3.6800000668
   -0.3999998569     3.8800001144
   -0.1999998540     4.3200006485
    0.0000001490     5.0000004768
    0.2000001520     5.9200010300
    0.4000001550     7.0800008774
    0.6000001431     8.4800014496
    0.8000001311    10.1200017929
    1.0000001192    12.0000009537
    1.2000001669    14.1200027466
    1.4000002146    16.4800014496
    1.6000002623    19.0800037384
    1.8000003099    21.9200038910
    2.0000002384    25.0000038147
```

Figure 9. Text file *Polynomial.out* with screen output from program *Polynomial.exe*.

```
ptf #
2                    Degree of polynomial, n
# coeff0  #          Coefficient a0, replaced by variable coeff0
# coeff1  #          Coefficient a1, replaced by variable coeff1
# coeff2  #          Coefficient a2, replaced by variable coeff2
-2.0                 xmin
 2.0                 xmax
21                   Number of points, m
```

Figure 10. Template file *Polynomial.tpl*, creating input files *Polynomal.in* for different values of the polynomial coefficients.

```
pif @
@y(x)@
l1 [y1]18:34
l1 (y2)29:31
l1   w w !y3!
l1 !dum! !y4!
l1 !dum! !y5!
l1 !dum! !y6!
l1 !dum! !y7!
l1 !dum! !y8!
l1 !dum! !y9!
l1 !dum! !y10!
l1 !dum! !y11!
l1 !dum! !y12!
l1 !dum! !y13!
l1 !dum! !y14!
l1 !dum! !y15!
l1 !dum! !y16!
l1 !dum! !y17!
l1 !dum! !y18!
l1 !dum! !y19!
l1 !dum! !y20!
l1 !dum! !y21!
```

Figure 11. Instruction file *Polynomial.ins* used to peruse output file *Polynomial.out*, extracting 21 values $y(x)$.

Figure 12 shows a complete iTOUGH2 input file used to perform the polynomial fit. The `> PARAMETER` block defines the three parameters to be estimated. The generic parameter type `PEST` must be selected; it refers to no particular domain, thus the third-level command `>>> NONE`. The parameter names provided through the `>>>> NAME` command must be identical to the names given in the template file *Polynomial.tpl* (see Figure 10). An initial value must be given either through command `>>>> GUESS` or `>>>> PRIOR`. In this example, the `>>>> VALUE` of the polynomial coefficient is directly estimated (rather than its logarithm, or a multiplication factor).

In the `> OBSERVATION` block, the generic observation type `>> PEST` is defined. Since it is unknown at which time a PEST observation was taken, there is no `>> TIMES` command. Similarly, since the locations of PEST observations are unknown, the generic command `>>> UNIVERSAL` has to be selected, optionally followed by a name describing the data set. The measured data points (against which the model is calibrated) are specified after command `>>>> DATA`. Each data point is identified by its name, which must be identical to the observation name used in the instruction file *Polynomial.ins* (see Figure 11), followed by the measured value. Since no weight is given on each individual data line, all residuals of this data set are weighted by the inverse of the standard deviation given after command `>>>> DEVIATION`.

In the `> COMPUTATION` block, the template file *Polynomial.tpl* is attached to the actual input file—*Polynomial.in*—it creates after each parameter update, and which is read by the external program *Polynomial.exe*. Similarly, the instruction file *Polynomial.ins* is attached to the output file *Polynomial.out* it searches after each completion of a *Polynomial.exe* forward run. Finally, the command to be executed for running the external program is provided. The command line includes the redirection of standard input (keyboard) and output (screen) to the input and output text files. Since this command is comprised of multiple words, it has to be surrounded by quotes. Parameters are written to the input file using single precision format and including a decimal point. Three Levenberg-Marquardt iterations are considered sufficient to identify the three coefficients.

When running this sample problem (using iTOUGH2 with an arbitrary EOS module linked to it), the iTOUGH2 input file *Poli* (Figure 12) has to be provided along with a dummy TOUGH2 input file, which needs to have the keyword `PEST` at the beginning of the first line, as shown in Figure 13.

Output can be found on the output files generated by the external program (here, file *Polynomial.out*) and all the standard iTOUGH2 output files, specifically *Poli.out*. The plotting file (*Poli.tec*) is also created, but instead of the (unknown) observation times, it prints the observation number within each data set as a floating point value in the times column. Obviously, no TOUGH2 output files are created.

The inversion results are not further examined here. Figure 14 shows that the data are matched after three non-linear iTOUGH2 optimization steps. This demonstrates that a non-TOUGH inversion is not tough for iTOUGH2, as expected.

```
iTOUGH2 input file demonstrating parameter estimation (polynomial fit)
using external program and PEST protocol

> PARAMETER
  >> PEST
     >>> NONE
         >>>> NAME : coeff0
         >>>> VALUE
         >>>> GUESS:   -1.0
         <<<<
     >>> NONE
         >>>> NAME : coeff1
         >>>> VALUE
         >>>> GUESS:   -1.0
         <<<<
     >>> NONE
         >>>> NAME : coeff2
         >>>> VALUE
         >>>> GUESS:   -1.0
         <<<<
     <<<
  <<

> OBSERVATION
  >> PEST
     >>> UNIVERSAL:  y=f(x)
          >>>> DATA
               y1    0.94179E+01
               y2    0.71294E+01
               y3    0.69108E+01
               y4    0.65802E+01
               y5    0.41660E+01
               y6    0.57779E+01
               y7    0.38172E+01
               y8    0.24940E+01
               y9    0.50483E+01
               y10   0.32697E+01
               y11   0.64006E+01
               y12   0.60516E+01
               y13   0.60600E+01
               y14   0.96430E+01
               y15   0.10834E+02
               y16   0.12887E+02
               y17   0.14458E+02
               y18   0.16869E+02
               y19   0.18289E+02
               y20   0.21521E+02
               y21   0.25278E+02
          >>>> DEVIATION: 1.0
          <<<<
     <<<
  <<
```

Figure 12. iTOUGH2 input file *Poli* for polynomial fit.

```
> COMPUTATION
  >> STOP
     >>> ITERATIONS: 3
     <<<

  >> OPTION
     >>> PEST
         >>>> TEMPLATE file   : 1
              Polynomial.tpl Polynomial.in

         >>>> INSTRUCTION file: 1
              Polynomial.ins Polynomial.out

         >>>> EXECUTABLE      : &
              'Polynomial.exe < Polynomial.in > Polynomial.out'

         >>>> PRECISION       : SINGLE
         >>>> DECPOINT        : ADD POINT
         <<<<
     <<<
  <<
<
```

Figure 12. iTOUGH2 input file Poli *for* polynomial fit. (cont.)

```
 PEST is not TOUGH
```

Figure 13. Dummy TOUGH2 input file.

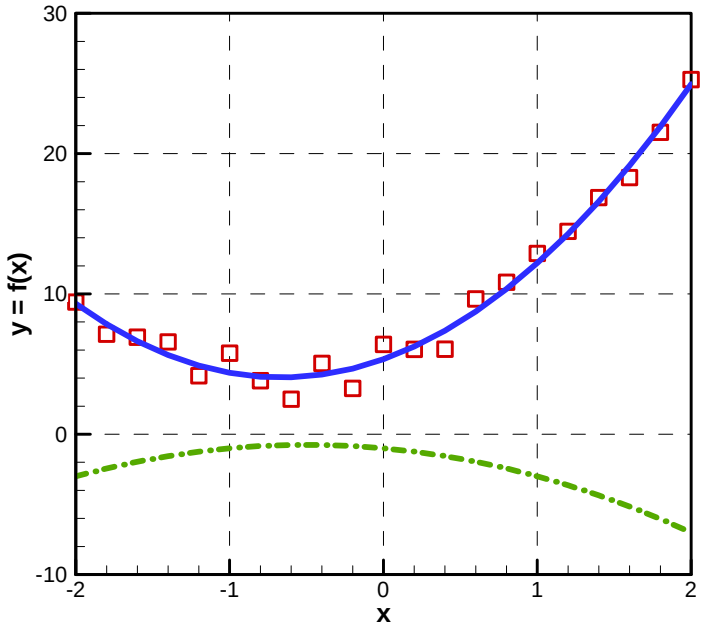


Figure 14. Data (symbols), polynomial with initial guess of coefficients (dash-dotted line), and fit after three iTOUGH2 Levenberg-Marquardt iterations (solid line).

5.2 Parallel Inversion of TOUGHREACT Model

The name TOUGH2 in iTOUGH2 indicates that the forward simulator TOUGH2 is integrated into the inversion code. iTOUGH2 provides inverse modeling capabilities for many but not all of the members of the TOUGH family of non-isothermal multiphase flow simulators. A list of publicly available modules that are fully integrated into iTOUGH2 can be seen at the TOUGH+ web site at http://esd.lbl.gov/TOUGH+/software-itough2.html. The close integration of a module into the inversion framework has considerable advantages over the loose coupling provided by the PEST protocol, as the code is aware of space and time, parameter and observation types, and the optimizer has direct access to variables of the forward simulator, and thus can partly control input to and execution of the TOUGH model. Integration of a modified or newly developed TOUGH2 module into the iTOUGH2 framework is relatively straight-forward. However, the rate at which TOUGH2 modules are integrated into iTOUGH2 cannot keep up with the rate at which new modules are developed or modified. Furthermore, integration of the more complex TOUGH2 codes, particularly TOUGHREACT [*Xu et al.*, 2004] is challenging. The iTOUGH2-PEST interface closes this gap, providing inversion capabilities for TOUGHREACT models (and other advanced simulators of the TOUGH family, see, e.g., Section 5.3).

This example demonstrates parameter estimation for a TOUGHREACT model using the parallel version iTOUGH2-PVM [*Finsterle*, 1998]. The same problem has been solved using Parallel Pest (PPEST) [*Doherty*, 2008]. TOUGHREACT is applied to simulate urea hydrolysis (ureolysis) as a means to remediate ^{90}Sr contamination in the saturated zone [*Spycher et al.*, 2009]. Ureolysis consumes hydrogen ions and produces ammonium and bicarbonate ions. Consequently, the injection of urea into groundwater causes pH and alkalinity to increase, driving calcite precipitation. ^{90}Sr, which strongly partitions into soils, exchanges with ammonium ions produced by ureolysis; the exchanged ^{90}Sr then precipitates with calcite. This reaction network is simulated for a column experiment, in which water with added urea was injected for 15 days, and water composition was obtained at the outlet. Prior to urea injection, molasses was added to the injected solution for a short period of time to stimulate ureolytic activity in the column.

The one-dimensional column model is discretized into 205 gridblocks at regularly spaced intervals of 1 mm. A sequential-iterative (transport/reaction) method is implemented, using a maximum time step of 500 s. The model considers ureolysis as an enzymatic reaction. It accounts for calcite precipitation, ion exchange, and ammonium oxidation. Details about the system behavior and TOUGHREACT model can be found in *Spycher et al.* [2009].

The iTOUGH2 parameter block (Figure 15) shows the five parameters to be estimated: the initial and boundary concentration of the urease enzyme (`zh_ini`), the initial and boundary concentration of the nitrosomonas biomass (`zn_ini`), the logarithm of the precipitation rate constant for calcite and strontianite (`rate-cc`), the exchange coefficient (selectivity) of kalium (`k-sel`), and the cation exchange capacity (`cec`). These parameters enter the TOUGHREACT file *chemical.inp*, which holds all geochemical parameters and properties of the aqueous component species, minerals, gases, and sorbed species. Initial guesses of all these parameters along with lower and upper bounds are provided, as well as whether the value or the logarithm of the parameter is to be estimated. The corresponding information as entered into the PEST control file is also shown in Figure 15.

```
> PARAMETER
  >> PEST

PEST: zh_ini  none relative 3.000000E-10 1.00e-12 0.40e-09  zh  1  0.00 1
     >>> NONE
         >>>> NAME : zh_ini
         >>>> VALUE
         >>>> GUESS: 3.0E-10
         >>>> RANGE: 1.0E-12 4.0E-10
         <<<<

PEST: zn_ini  none relative 1.022300E-14 1.00e-15 1.00e-10  zn  1  0.00 1
     >>> NONE
         >>>> NAME : zn_ini
         >>>> VALUE
         >>>> GUESS: 1.0223E-14
         >>>> RANGE: 1.0E-15 1.0E-10
         <<<<

PEST: rate-cc log  factor   4.000000E-08 1.E-10   1.E-5    rate 1  0.00 1
     >>> NONE
         >>>> NAME : rate-cc
         >>>> LOGARITHM
         >>>> GUESS:  -7.39794
         >>>> RANGE: -10.0  -5.0
         >>>> STEP : 2.0
         <<<<

PEST: k-sel   none relative  0.49        0.01     4.00     sel  1  0.00 1
     >>> NONE
         >>>> NAME : k-sel
         >>>> VALUE
         >>>> GUESS:  0.49
         >>>> RANGE:  0.01 4.00
         >>>> STEP : 0.2
         <<<<

PEST: cec     none relative  10.         2.       200.     sel  1  0.00 1
     >>> NONE
         >>>> NAME : cec
         >>>> VALUE
         >>>> GUESS:  10.0
         >>>> RANGE:   2.0  200.0
         >>>> STEP : 2.0
         <<<<
     <<<
   <<
```

Figure 15. Parameter block for TOUGHREACT inversion, also showing corresponding PEST control file entry.

Time series of measured concentrations of NH_4^+, NO_3^-, dissolved O_2, Urea, Ca, Sr, Na, and K are available and entered into the `> OBSERVATION` block as separate data sets (see Figure 16). Each measurement point has its own weight specified. The data format chosen here is exactly the same as that in the PEST control file.

```
> OBSERVATION

  >> PEST

     >>> UNIVERSAL: NH4+
         >>>> DATA
              NH4-1day   3.9444E-05   2.5352E+04    NH4+
              NH4-2day   2.5111E-04   3.9823E+03    NH4+
              NH4-3day   3.9111E-04   2.5568E+03    NH4+
              NH4-4day   5.4556E-04   3.8330E+03    NH4+
              NH4-5day   4.5722E-04   2.1871E+03    NH4+
              NH4-6day   5.4222E-04   3.8443E+03    NH4+
              NH4-7day   5.3444E-04   1.8711E+03    NH4+
              NH4-8day   5.4167E-04   1.8461E+03    NH4+
              NH4-9day   5.3778E-04   1.8595E+03    NH4+
              NH4-10day  5.3833E-04   1.8576E+03    NH4+
              NH4-11day  5.1722E-04   1.9334E+03    NH4+
              NH4-12day  4.8222E-04   2.0737E+03    NH4+
              NH4-13day  5.5167E-04   1.8127E+03    NH4+
              NH4-14day  5.2222E-04   1.9149E+03    NH4+
              NH4-15day  5.1500E-04   1.9417E+03    NH4+
         <<<<

     >>> UNIVERSAL: NO3-
         >>>> DATA
              NO3-1day   4.4200E-05   2.2624E+04    NO3-
              NO3-2day   5.9700E-05   1.6750E+04    NO3-
              ........   ..........   ..........    ....
              NO3-15day  8.1000E-05   1.2346E+04    NO3-
         <<<<

     >>> UNIVERSAL: XXXX

 similar blocks are provided for XXXX = O2(aq), Urea, Ca, Sr, Na, and K

         >>>> DATA
              XXXX-1day  VALUE       WEIGHT        XXXX
              XXXX-2day  VALUE       WEIGHT        XXXX
              ........   ..........  ..........    ....
              XXXX-15day VALUE       WEIGHT        XXXX
         <<<<
     <<<
  <<
```

Figure 16. Excerpt of observation block for TOUGHREACT inversion.

The > COMPUTATION block (Figure 17) relates the template file to the TOUGHREACT input file *chemical.inp*, and the instruction file to the plot file *tec_conc.dat*, from which the concentration data are to be extracted. The TOUGREACT executable happens to be called *tr2.056_e1x*. Since this inversion is performed on a Linux cluster, all input files need to be copied to the temporary directory, which is accomplished by listing them on separate lines following the keyword FILE:. Finally, the >>> PVM command invokes embarrassingly parallel execution of TOUGREACT for the calculation of the columns of the Jacobian matrix and the evaluation of a potential update step with different Levenberg parameters λ (see *Finsterle* [1998] for details about iTOUGH2-PVM). Since only five parameters are estimated, parallelization is limited to five nodes, which are listed after the command. Five Levenberg-Marquardt iterations will be performed using default options.

```
> COMPUTATION

 >> OPTION
    >>> PEST
        >>>> TEMPLATE: 1
             ureolysis.tpl chemical.inp

        >>>> INSTRUCTION: 1
             ureolysis.ins tec_conc.dat

        >>>> EXECUTABLE FILE: tr2.056_e1x

        FILE: ureolysis.tpl         template file
        FILE: ureolysis.ins         instruction file
        FILE: tk-mtq.v4_1.02y.dat   geochemical data base
        FILE: flow.inp              TOUGHREACT input file for flow
        FILE: solute.inp            TOUGHREACT input file for transport
        FILE: MESH                  TOUGHREACT mesh file
        FILE: INCON                 TOUGHREACT initial conditions file
        FILE: GENER                 TOUGHREACT generation files
        <<<<

    >>> PVM: 5
        HOST1PVM  node0010
        HOST2PVM  node0011
        HOST3PVM  node0012
        HOST4PVM  node0013
        HOST5PVM  node0014
    <<<

 >> STOP
    >>> ITERATIONS: 5
    <<<
 <<
<
```

Figure 17. Computational parameter block for TOUGHREACT inversion.

An excerpt of the template file *ureolysis.tpl* is shown in Figure 18, showing how the precipitation rate for calcite and strontianite are linked simply by using the same parameter name in the template file.

Figure 19 shows the first few lines of the instruction file *ureolysis.ins*, which extracts simulation results from file *tec_conc.dat*, which is usually used for plotting purposes.

```
ptf #
'Ureolysis'
'---------'
'DEFINITION OF THE GEOCHEMICAL SYSTEM'
...
'*'
'MINERALS'           !equilibrium minerals go first
'CO2-3.4' 0  0 0   0
0. 0. 0.
'calcite' 1  3  1   0
1.00e-07       0 1.0 1.0 48.1 0.0 0.0 0.0
# rate-cc  #   0 1.0 1.0 48.1 0.0 0.0 0.0  1.e-6   0
0. 0. 0.
'strontianite' 1  3  1   0
1.00e-07       0 1.0 1.0 48.1 0.0 0.0 0.0
# rate-cc  #   0 1.0 1.0 48.1 0.0 0.0 0.0  1.e-6   0
0. 0. 0.
'*'
...
```

Figure 18. Excerpt of template file for TOUGHREACT inversion.

```
pif @
@ZONE T= "0.273785E-02 yr"@
l1
l1 [Na-1day]94:104    [K+-1day]130:140    [Ca-1day]142:152
&  [Sr-1day]154:164   [NO3-1day]202:212   [O2-1day]226:236
&  [Urea-1day]238:248 [NH4-1day]274:284
@ZONE T= "0.547570E-02 yr"@
l1
l1 [Na-2day]94:104    [K+-2day]130:140    [Ca-2day]142:152
&  [Sr-2day]154:164   [NO3-2day]202:212   [O2-2day]226:236
&  [Urea-2day]238:248 [NH4-2day]274:284
@ZONE T= "0.821355E-02 yr"@
l1
...
```

Figure 19. Excerpt of instruction file for TOUGHREACT inversion.

The inversion results are summarized in Table 2 and compared to the results obtained with Parallel PEST (PPEST). Both codes converged to the same objective function value and the same solution in the parameter space. The differences between the estimated parameters are a result of the different implementation of the Levenberg-Marquardt algorithm, and specifically the different default values of computational parameters (such as the initial values of the Levenberg and Marquardt parameters, step size limitations, etc.) However, these differences are much smaller than the estimation uncertainty, which is also consistently calculated by the two optimization codes. PPEST took almost twice as many TOUGHREACT simulation runs as iTOUGH2 did, mainly because it switched to central finite differences for evaluating derivatives after two iterations.

Table 2. iTOUGH2-PEST-PVM and PPEST Inversion Results of TOUGHREACT Model

Parameter	**Initial**	**Best Estimate**		**Uncertainty**	
		PPEST	**iTOUGH2**	**PPEST**	**iTOUGH2**
Obj. Function	7.3034	5.2356	5.2356	n/a	n/a
Model Runs	n/a	91	51	n/a	n/a
zh_ini	3.000×10^{-10}	1.847×10^{-10}	1.845×10^{-10}	0.239×10^{-10}	0.238×10^{-10}
zn_ini	1.022×10^{-14}	1.026×10^{-14}	1.026×10^{-14}	0.092×10^{-14}	0.093×10^{-14}
$\log_{10}$(rate-cc)	-7.398	-7.302	-7.304	0.051	0.055
k-sel	0.49	0.538	0.535	0.129	0.125
cec	10.000	8.608	8.580	1.148	1.090

This particular inversion took approximately 16 hours to complete on a Linux cluster. Almost all CPU time is used for repeatedly running the TOUGHREACT simulation model; only a negligible fraction is used by the minimization algorithm, residual, and uncertainty analyses. Evaluating the Jacobian matrix and testing Levenberg parameters in parallel on five processors sped up the inversion by a factor of 2.5, which is only a moderate gain because of the relatively small number of parameters to be estimated.

5.3 Evaluating Parallelization of TOUGH2-MP Models

In this example, the CPU-time and other simulation performance metrics are evaluated for flow and transport simulations performed in parallel using an increasing number of processors. This example will demonstrate (1) that iTOUGH2-PEST can be used to run the MPI-based parallel version of the TOUGH2 simulator, i.e., TOUGH2-MP [*Zhang et al.*, 2007, 2008], (2) that the forward model can consist of multiple parallel models run in series, and (3) that the tool can be used to evaluate the scalability of TOUGH2-MP. (Needless to say that this exercise could be done using a simple Unix script file; however, to qualify as an example in this manual, the problem is solved using iTOUGH2-PEST, at the expense of losing elegance, simplicity, and transparency.)

The forward model consists of two sequential TOUGH2_MP simulations. The first simulation uses T2EOS9_MP to establish a steady-state flow field, which is passed on to the second simulation to calculate radionuclide transport in the vicinity of an underground research facility using T2R3D_MP. The model domain is discretized into approximately 90,000 grid blocks with about 270,000 connections between them (Figure 20). (Unfortunately, discussing the simulation results is beyond the scope of this manual. The only objective is to see how fast the simulation is performed as a function of the number of processors.)

This sequence of two TOUGH2_MP simulations is set up using a Unix shell script file, which is created by the PEST template file shown in Figure 21. The two simulations are invoked using the `mpirun` command, which has as an argument the number of processors. Instead of specifying a fixed for the `–np` argument, the PEST parameter `#nproc#` is inserted. The script also provides the appropriate input files and renames the relevant output from the flow and transport simulations as files *OUTPUT_F* and *OUTPUT_T* for subsequent parsing by the two PEST instruction files, one of which is shown in Figure 22. The primary marker ("`WRITE FILE *SAVE* AFTER`") ensures that the final CPU time, number of times steps, number of Newton-Raphson iterations, and number of Aztec iterations are extracted from the output file.

Figure 23 shows the `> PARAMETER` and `> OBSERVATION` blocks of the iTOUGH2-PEST input file. The `> COMPUTATION` block is shown in Figure 24. A single parameter named `nproc` is adjusted: it is the number of processors to be used in a parallel TOUGH2_MP simulation invoked by command `mpirun` (see Figure 21). The number of processors evaluated using iTOUGH2's grid search method ranges from 2 to 48 in increments of 2, selected by command `>>>> RANGE` in combination with command `>>> GRID SEARCH: 23` (see Figure 24 below).

The `> OBSERVATION` block contains four PEST-related data sets: CPU time, number of time steps, number of Newton-Raphson iterations, and number of Aztec iterations. Each set contains two observation points, one for the flow and one for the transport simulation. A value of 0.0 is provided as a dummy measurement, so that the first residual will be the CPU time used for the flow simulation. (Note that the weight of each residual is 1.0 by default.) Consequently, all residuals are equal to the corresponding performance metrics themselves. Moreover, because the L_1 estimator is selected (see Figure 24 below), the absolute values (rather than the squares) of the residuals are taken and added to the data-set-specific contributions to the objective function. As a result, the objective function for the first data set is the total CPU time, i.e., the sum of the CPU times used for the flow and transport simulations. Similarly, the sums of the other performance metrics will be calculated and reported as the result of the grid search. The grid search thus provides the performance statistics as a function of the number of processors used.

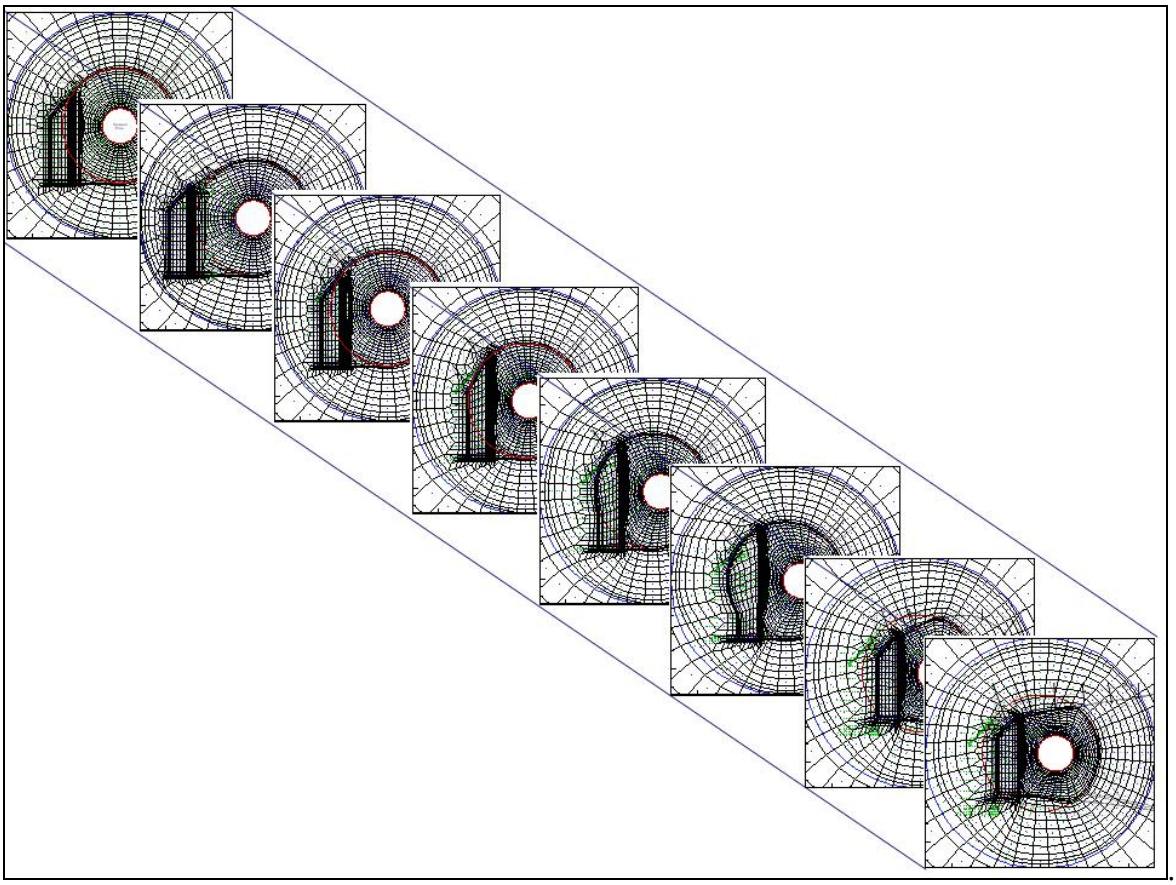


Figure 20. Unstructured grid with approximately 90,000 elements and 270,000 connections, generated using WinGridder [*Pan*, 2007].

```
ptf #
echo iTOUGH2-PEST run of two TOUGH2-MP models     > mpirun.msg
echo ========================================    >> mpirun.msg
date                                             >> mpirun.msg
cp INFILE_F INFILE                               >> mpirun.msg
cp INCON_F  INCON                                >> mpirun.msg
echo Run flow simulation                         >> mpirun.msg
echo ====================                         >> mpirun.msg
mpirun --hostfile hf -np #nproc# t2eos9_mp       >> mpirun.msg
cp OUTPUT   OUTPUT_F                             >> mpirun.msg
cp INFILE_T INFILE                               >> mpirun.msg
cp GENER_T  GENER                                >> mpirun.msg
echo Run transport simulation                    >> mpirun.msg
echo ========================                    >> mpirun.msg
mpirun --hostfile hf -np #nproc# t2r3d_mp        >> mpirun.msg
cp OUTPUT   OUTPUT_T                             >> mpirun.msg
date                                             >> mpirun.msg
```

Figure 21. PEST template file *it2mp.tpl* that creates a Unix shell script file for running TOUGH2_MP flow and transport simulations on an adjustable number of processors.

```
pif @
@ WRITE FILE *SAVE* AFTER@
@ EEE Time@ @=@                    !cpu-timeF!
@ Total number of time steps =@    !time-stepsF!
@ Total number Newton steps =@     !newtonF!
@ Total number of iter in Aztec =@ !aztec-iterF!
```

Figure 22. PEST instruction file *it2mpF.ins* that reads performance metrics from TOUGH2_MP flow simulation output file *OUTPUT_F*.

```
> PARAMETER
  >> PEST
     >>> NONE
         >>>> NAME : nproc
         >>>> VALUE
         >>>> GUESS:  16.0
         >>>> RANGE:  2.0 20.0
         <<<<
     <<<
  <<

> OBSERVATION
  >> PEST
     >>> UNIVERSAL : "CPU Time"
         >>>> DATA
              cpu-timeF     0.0
              cpu-timeT     0.0
         <<<<
     >>> UNIVERSAL : "Time Steps"
         >>>> DATA
              time-stepsF   0.0
              time-stepsT   0.0
         <<<<
     >>> UNIVERSAL : "Newton Iterations"
         >>>> DATA
              newtonF       0.0
              newtonT       0.0
         <<<<
     >>> UNIVERSAL : "Aztec Iterations"
         >>>> DATA
              aztec-iterF   0.0
              aztec-iterT   0.0
         <<<<
     <<<
  <<
```

Figure 23. Parameter and observation block of iTOUGH2-PEST input file.

```
> COMPUTATION
  >> OPTION
     >>> PEST
         >>>> EXECUTABLE   : "sh mpirun.sh"

         >>>> TEMPLATE     : 1
                             it2mp.tpl mpirun.sh

         >>>> INSTRUCTION  : 2
                             it2mpF.ins OUTPUT_F
                             it2mpT.ins OUTPUT_T

                 copy FILE : t2eos9_mp  to temporary directory
                 copy FILE : t2r3d_mp   to temporary directory
                 copy FILE : it2mp.tpl  to temporary directory
                 copy FILE : it2mpF.ins to temporary directory
                 copy FILE : it2mpT.ins to temporary directory
                 copy FILE : hf         to temporary directory
                 copy FILE : INFILE_F   to temporary directory
                 copy FILE : INFILE_T   to temporary directory
                 copy FILE : INCON_F    to temporary directory
                 copy FILE : GENER_T    to temporary directory
                 copy FILE : MESH       to temporary directory
                 copy FILE : PARAL.prm  to temporary directory
         <<<<

     >>> L1-ESTIMATOR
     >>> GRID SEARCH: 18
     <<<
  <<
<
```

Figure 24. Computation block of iTOUGH2-PEST input file.

The `> COMPUTATION` block of Figure 24 starts with the `PEST` command. The executable in this case is the Unix shell script file *mpirun.sh*. Recall that it is generated from the template file *it2mp.tpl* (Figure 21), i.e., it is a simple text file that has read and write, but not execute permissions. Consequently, it is not possible to just start it as a command itself, but it has to be executed through the Bourne shell command `sh`. The space in the executable command calls for quotes.

The template file *it2mp.tpl* is associated with the script file *mpirun.sh* through command `>>>> TEMPLATE`. There are two instruction files in this example, one reading the output from the flow simulation, and one for the transport simulation. The files are properly assigned in command `>>>> INSTRUCTION`. The following lines are not commands interpreted by iTOUGH2-PEST (no command-level markers `>`), but by the *itough2* script file used on Unix machines to start an iTOUGH2 application. This script file generates local directories (so multiple inversions can be run at the same time without creating file sharing conflicts). It parses through the iTOUGH2 input file and looks for the keyword `FILE`, reads the file name that follows the colon, and then copies the file

to the temporary directory. Therefore, to make sure all the files needed by the external program (here the input and control files for the TOUGH2_MP flow and transport simulator), as well as the template and instruction files, are available in the temporary directory, they are listed here (or anywhere else in the iTOUGH2 input file) for the *itough2* script file to read and copy.

As explained before, the L_1 estimator is selected to yield an objective function that directly reflects the CPU time and the other performance measures.

Next, iTOUGH2 is instructed to do a simple grid search, i.e., to evaluate the objective function for 24 parameter values by subdividing the `>>>> RANGE` given in the `> PARAMETER` block into 18 intervals. For this unusual application, it is essential that this subdivision yields values that are whole numbers, as the parameter in question is the number of processors, and a fractional processor does not do a satisfactory job. Here, the command will create a series of whole numbers 2, 3, 4,…, 20. Note, however, that iTOUGH2 only creates real parameter values, not integers. Fortunately, the `-np` argument of the `mpirun` command seems to accept whole numbers with a decimal point as a proper argument. Also note that the decimal point cannot be removed simply by using the `>>>> DECPOINT: NOPOINT` command for reasons that become only evident when fully understanding the workings of this PEST option see Section 3.2.6 of *Doherty* [2008].

The results are visualized in Figure 25. The total CPU time generally decreases with increasing number of processors in a somewhat erratic way as a result of the varying number of time steps, Newton-Raphson iterations, and Aztec iterations needed. Running this problem using 18 processors would be optimal, reducing the CPU time by a factor of over seven compared to a run on two processors.

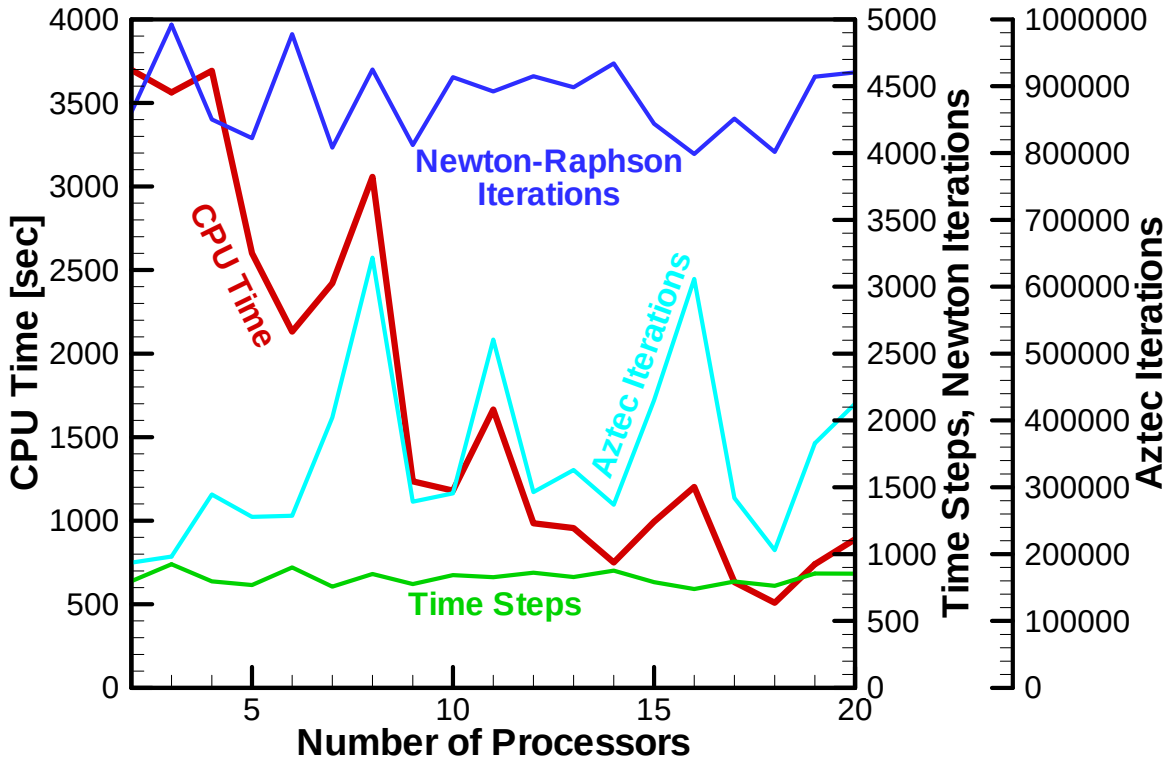


Figure 25. CPU time and iteration statistics as a function of processors.

5.4 Adjusting Pre-Processor and Simulation Parameters

The iTOUGH2-PEST code can be used to simultaneously adjust parameters of an external model and a TOUGH2 model. This is useful if the external model is either a pre- or postprocessor of TOUGH2. In this example, parameters of a mesh generator (which produces the grid representing a discrete fracture network) are updated, and the mesh is recreated and used as input to a TOUGH2 seepage simulation. Select output from both the external mesh generator (here, the number of fractures) and the flow simulator (seepage into an opening excavated from the fractured formation) are evaluated for an uncertainty analysis. Multiple steps are needed to generate a discrete fracture network model (see Table 3).

Table 3. Steps to Generate Discrete Fracture Network Model

Step	Activity	Software
0	Script file invoking mesh generation steps 1–6 External executable called before each TOUGH2 simulation	*sh.DFNMgen* (see Figure 26)
1	Generate 2D network of fracture traces based on statistical parameters on fracture density, fracture length, and fracture orientation provided through an input file that is created by the PEST template file Remove unconnected fractures Discretize fracture traces, assign aperture and permeabilities to fracture elements, create TOUGH2 ELEME and CONNE blocks	*xDFNM*
2	Concatenate ELEME and CONNE block to create base MESH file	*sh.DFNMgen*
3	Move X and Z coordinates of mesh	*xMoveMesh8*
4	Add top boundary element	*xAddBound8*
5	Add bottom boundary element	*xAddBound8*
6	Cut out niche from mesh, adjust permeabilities near niche to reflect excavation disturbed zone	*xCutNiche8*

These mesh generation steps are executed by a Linux shell script file *sh.DFNMgen* (see Figure 26); it is the executable called by iTOUGH2 prior to each TOUGH2 forward simulation. The parameters to be varied by iTOUGH2 are input to the program *xDFNM*, stored on file *input.dat*, which is created by the PEST template file *input.tpl* (see Figure 27). The fracture network consists of two fracture sets generated using six statistical parameters: fracture trace length follows a power-law distribution, with the coefficient α and exponent $-a$ as its parameters; the orientations of the two fracture sets follow normal distributions, each with a given mean and standard deviation. Fracture aperture—and thus permeability—is correlated to the fracture length (for details, see *Liu et al.* [2002] and *Zhang et al.* [2010]), with increased permeabilities in the excavation disturbed zone.

Once the base fracture network has been generated, unconnected fractures are removed, the fracture traces are discretized according to the TOUGH2 spatial discretization scheme, an opening representing an excavated niche is cut form the mesh, and boundary elements are created. The output from these mesh generation steps is a MESH file that is read by TOUGH2 for the subsequent simulation of unsaturated flow through the discrete fracture network and seepage into the niche.

```
#! /bin/sh
# Unix shell script file sh.DFNMgen
#
xDFNM
cat ELEME >  DFNM_mes
cat CONNE >> DFNM_mes
cat >> DFNM_mes << eof

eof
#
xMoveMesh8 << eof
DFNM_mes                              # input mesh file
temp1.mes                             # output mesh file
-5.0                                  # dx
 0.0                                  # dy
-0.25                                 # dz
eof
#
echo
echo Add top boundary
echo ----------------
xAddBound8 << eof
temp1.mes                             # input mesh file
temp2.mes                             # output mesh file
TOP99999                              # boundary element name
BOUND                                 # boundary material type
0.3                                   # boundary element volume
1.0e-5                                # nodal distance to boundary element
-100.0                                # xmin
 100.0                                # xmax
-100.0                                # ymin
 100.0                                # ymax
10.0                                  # zmin
11.0                                  # xmax
eof
#
echo
echo Add bottom boundary
echo -------------------
xAddBound8 << eof
temp2.mes                             # input mesh file
temp3.mes                             # output mesh file
BOT99999                              # boundary element name
DRAIN                                 # boundary material type
1.0E+20                               # boundary element volume
1.0e-5                                # nodal distance to boundary element
-100.0                                # xmin
 100.0                                # xmax
-100.0                                # ymin
 100.0                                # ymax
-1.00                                 # zmin
 1.00                                 # zmax
eof
```

Figure 26. Unix script file that generates mesh for discrete fracture network model.

```
echo
echo Cut out niche
echo -------------
xCutNiche8 << eof
temp3.mes                              # input mesh file
temp4.mes                              # output mesh file
2000.0                                 # niche volume
  1.0e-10                              # nodal distance niche - wall
  1.0                                  # cosine multiplication factor
 -2.0                                  # Xmin
  2.0                                  # Xmax
-10.0                                  # Ymin
 10.0                                  # Ymax
  0.0                                  # Zmin
  2.5                                  # Zmax
  0.0                                  # Xcenter
  0.2083                               # Zcenter
  3.0417                               # Radius
  1.0                                  # thickness of skin zone
100.0                                  # skin zone permeability modifier
  1                                    # gradual skin zone
eof
#
# Remove connections between niche and bottom boundary
# Remove '+++' from GENER file
#
grep -v "NIC98   BOT99999" temp4.mes | grep -v "NIC99" > MESH
cat GENER | sed 's/+++/   /g'                          > dum.gen
mv dum.gen GENER
#
echo
echo Shell script sh.DFNMgen terminated
echo ================================
```

Figure 26 (cont.): Unix script file that generates mesh for discrete fracture network model

```
ptf #
1  10      ! number of layers, number of profiles
10.0 10.5  ! the 2-D domain size x and Z
10.5       ! Thickness of the layers
 1.0   10.       #a         #  #alpha     #  ! lmin, lmax, a, alpha
#angle1    #  #anglesd1  #  #angle2    #  #anglesd2  #  ! mean and sigma
0.1        ! a small angle value used to adjust fracture orientation
1 1 0      ! control parameters
0.10  1.   ! max length of TOUGH2 element
```

Figure 27. Template file *input.tpl*.

Figure 28 visualizes the sequence of mesh generation steps, and shows some realizations obtained by varying the statistical input parameters. The permeability and steady-state saturation fields are also shown.

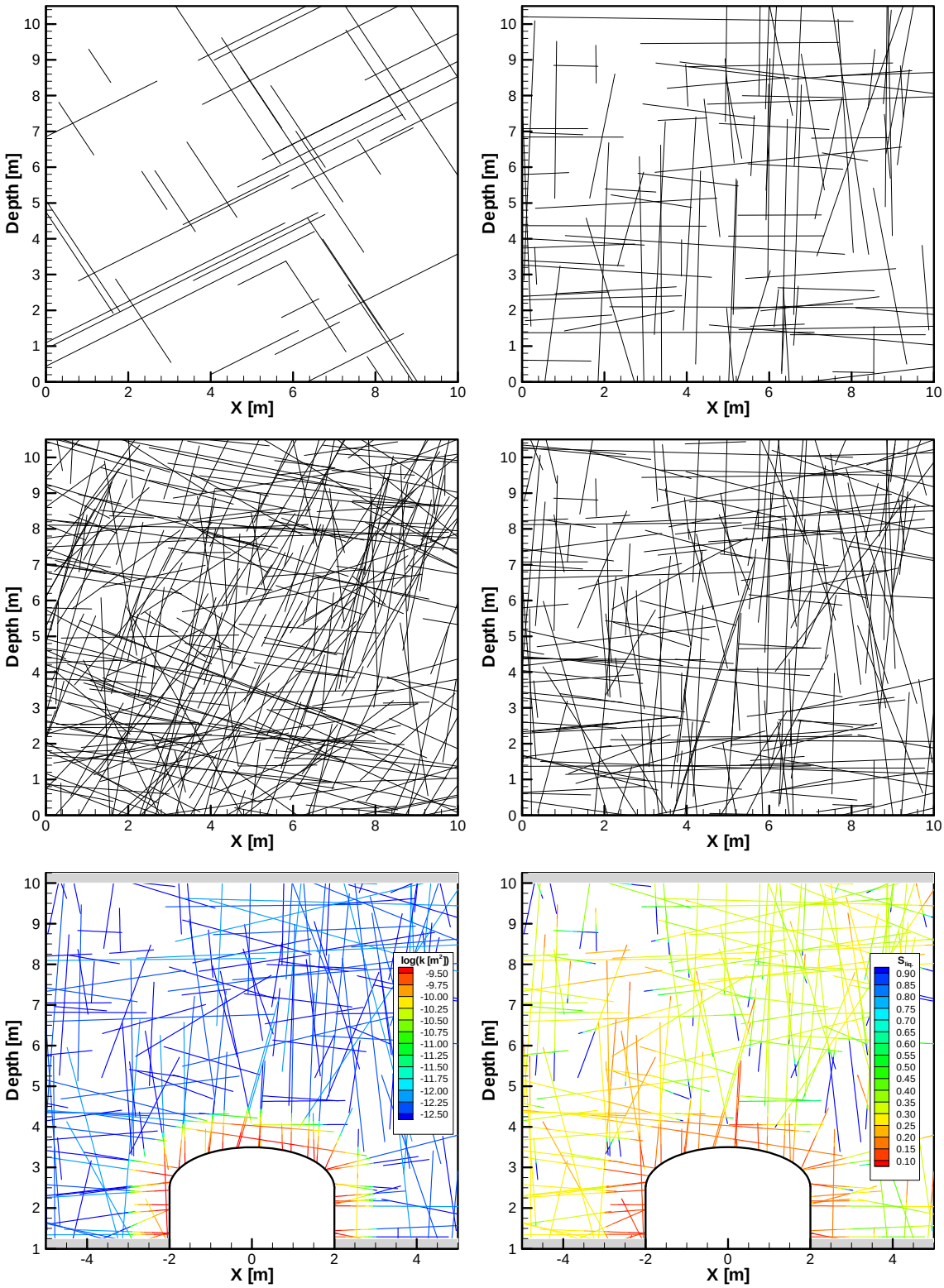


Figure 28. Four realizations of the base discrete fracture network, permeability field, and steady-state saturation distribution.

The TOUGH2 input file for simulating unsaturated flow through the discrete fracture network and seepage into the excavated opening is shown in Figure 29. Note that blocks ELEME and CONNE are absent; the mesh will be read from an external file *MESH*, which is generated by the script file *sh.MESHgen* (Figure 26). The simulation starts at a large negative time. Once steady state is reached, the saturation in the element representing the niche is reset. Seepage continues for one more year, with the total amount of liquid accumulated in the niche element being the result of interest.

Figure 30 shows the > PARAMETER block of the iTOUGH2 input file, defining the five statistical parameters needed by the *xDFNM* mesh generator. The parameters are written to the *xDFNM* input file *input.dat* through the PEST template file *input.tpl* (Figure 27).

```
DFNM: seepage into niche using discrete fracture network model
ROCKS----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
FRACT    0     2650.       1.0   1.0E-12   1.0E-12   1.0E-12              -1000.
SKINZ    0     2650.       1.0   1.0E-12   1.0E-12   1.0E-12              -1000.
NICHE    2     2650.       1.0   1.0E-08   1.0E-08   1.0E-08               1000.

    3           0.99
    9
BOUND    2     2650.       1.0   1.0E-08   1.0E-08   1.0E-08             100000.

    3           0.01
    9
DRAIN    2     2650.       1.0   1.0E-08   1.0E-08   1.0E-08             100000.
REFCO    0     1.0E5      20.0   1.0E+03   1.0E-03   4.4E-10             100000.

RPCAP----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
   11           0.00      0.00
   11            1.5   -5000.0
PARAM----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
   39999    999910000090002110010000 5000
-1.000E+10           0.100E+06                        -9.81
 1.000E-04
                0.01
MOMOP----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
38
GENER----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
TOP99999INF 0                0     COM1 6.3420E-06

START----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
INCON----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8

ENDCY----1----*----2----*----3----*----4----*----5----*----6----*----7----*----8
```

Figure 29. TOUGH2 input file *DFNM* for simulating unsaturated flow through discrete fracture network and seepage into underground opening.

```
> PARAMETER
  >> PEST
     >>> NONE
         >>>> NAME: a
         >>>> VALUE
         >>>> GUESS: 1.03
         >>>> RANGE: 1.01 1.05
         >>>> VARIATION: 0.01
         <<<<

     >>> NONE
         >>>> NAME: alpha
         >>>> VALUE
         >>>> GUESS: 100
         >>>> RANGE: 50 150
         >>>> VARIATION: 10.0
         <<<<

     >>> NONE
         >>>> NAME: angle1
         >>>> VALUE
         >>>> GUESS: 0.0001
         >>>> RANGE: -25.0 25.0
         >>>> VARIATION: 10.0
         <<<<

     >>> NONE
         >>>> NAME: anglesd1
         >>>> VALUE
         >>>> GUESS: 10.0
         >>>> RANGE: 5.0 15.0
         >>>> VARIATION:  2.0
         <<<<

     >>> NONE
         >>>> NAME: angle2
         >>>> VALUE
         >>>> GUESS: 90.0
         >>>> RANGE: 70.0 110.0
         >>>> VARIATION: 10.0
         <<<<

     >>> NONE
         >>>> NAME: anglesd2
         >>>> VALUE
         >>>> GUESS: 10.0
         >>>> RANGE: 5.0 15.0
         >>>> VARIATION:  2.0
         <<<<
     <<<
  <<
```

Figure 30. iTOUGH2 input file *DFNMi*, PARAMETER block.

Figure 31 shows the `> OBSERVATION` block of the iTOUGH2 input file. A single point in time at 1 year is specified, at which the change in total mass of water is extracted. Soon after beginning of the steady-state portion of the simulation, the niche element volume is set to 10^{50} m^3, making it a Dirichlet boundary condition. At restart time 0, the volume of the niche element is set at 1,000 m^3, and its saturation is reset at its initial value, so the change in mass reflects the cumulative amount of water seeping into the opening within 1 year.

There are also two PEST parameters: the total number of fractures of the base network and the number of connected fractures. These data are not related to a particular point in time, and dummy data values of zero and weights of one are provided. This information is read from file *fracture.frq*, which is an output file from the *xDFNM* code. Note that if an input parameter is varied through the PEST protocol, it is necessary to select at least one PEST observation, even if the output of interest comes from the TOUGH2 simulation. This may be a dummy data point read from a dummy file.

```
> OBSERVATION

  >> TIMES: 1 year
     1.0

  >> RESTART TIMES: 1
     -9998000000.0
     NICHE 0 1.0E50

  >> RESTART TIMES: 1    reset water in niche
     0.0
     NICHE 0 1.0E3
     NICHE 1 0.01

  >> PEST
     >>> UNIVERSAL: fractures
         >>>> DATA
              num_fract_tot  0.0  1.0
              num_fract_con  0.0  1.0
         <<<<
     <<<

  >> CHANGE TOTAL MASS
     >>> MATERIAL: NICHE
         >>>> COMPONENT: 1
         >>>> ZERO DATA
         <<<<
     <<<
  <<
```

Figure 31. iTOUGH2 input file *DFNMi*, `OBSERVATION` block.

```
> COMPUTATION

  >> STOP
     >>> Number of SIMULATIONS: 500
     <<<

  >> ERROR propagation analysis
     >>> MONTE CARLO SEED: 5555
     >>> LATIN HYPERCUBE SAMPLING CORRELATION MATRIX: 6
         1E-4    0.0    0.0    0.0   0.0   0.0
          0.0  100.0    0.0    0.0   0.0   0.0
          0.0    0.0  100.0    0.0   0.9   0.0
          0.0    0.0    0.0    4.0   0.0   0.5
          0.0    0.0    0.9    0.0 100.0   0.0
          0.0    0.0    0.0    0.5   0.0   4.0
     <<<

  >> OPTION
     >>> PEST
         >>>> EXECUTABLE     : sh.DFNMgen     run BEFORE TOUGH2!
         >>>> TEMPLATE       : 1
                               input.tpl      input.dat
         >>>> INSTRUCTION    : 1
                               fracture.ins   fracture.frq

         FILE: input.tpl
         FILE: fracture.ins
         FILE: sh.DFNMgen

         <<<<

     >>> STEADY STATE

     >>> PVM: 30  FILE: NODEFILE
     HOST1PVM
     HOST2PVM
     ...
     HOST30PVM
     <<<

  <<
<
```

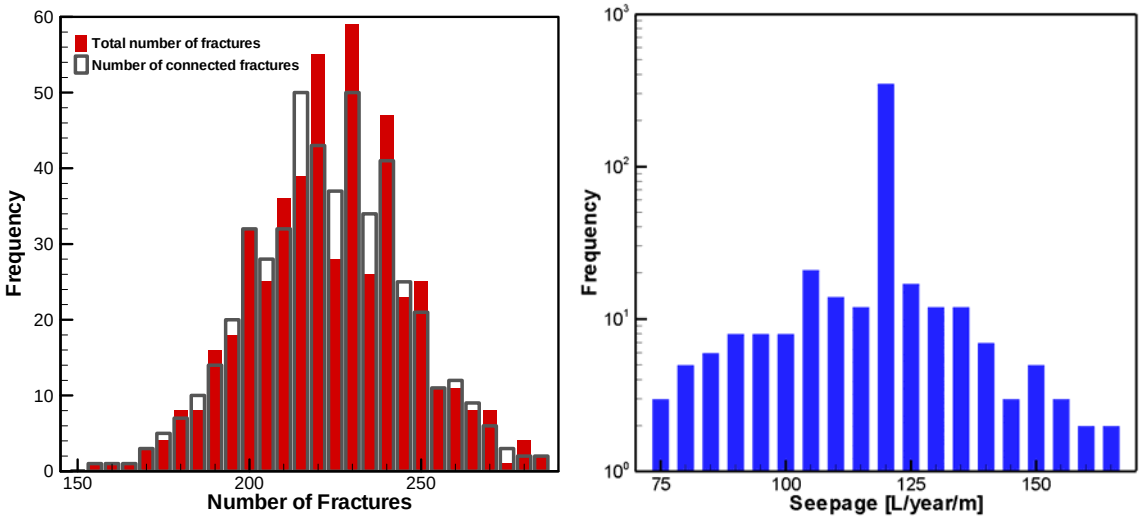
Figure 32. iTOUGH2 input file *DFNMi*, `COMPUTATION` block.

Figure 32 shows the `> COMPUTATION` block of the iTOUGH2 input file. In this application, the execution of 500 Monte Carlo simulations based on the Latin hypercube sampling strategy is used to examine the impact of the characteristics of the discrete fracture network on seepage. A covariance/correlation matrix of the six PEST parameters is provided, with the variances on the diagonal, and correlation coefficients on off-diagonal elements. Here, it is assumed that the two fracture sets are approximately orthogonal to each other; a correlation coefficient of 0.9 between the third and fifth parameters (those representing the mean angles for each fracture set) induces this

statistical correlation. A weaker correlation coefficient of 0.5 is given for the respective standard deviations.

The PEST-related commands and related input files have been previously discussed. The names of external input files—*input.tpl*, *fracture.ins*, and *sh.MESHgen*—are repeated on separate lines following keyword `FILE:` to make sure the *itough2* shell script copies these files to the temporary directory where the iTOUGH2 run is executed. Keyword `BEFORE` is used to indicate that the PEST executable—the Unix script file *sh.MESHgen*—shall be run before the TOUGH2 simulation. (Keyword `AFTER` would be used if the external code were a postprocessor of the TOUGH2 output file.) The 500 Monte Carlo simulations are evaluated in parallel on 30 processors on a Linux cluster. The names of the nodes are stored on file *NODEFILE*, which is generated by the scheduler.

Figure 33 shows the results of the analysis, which evaluates the uncertainty in the conceptual model, i.e., the characteristics of the fracture network, on seepage.



(a) (b)

Figure 33. (a) Histogram of number of fractures generated for different statistical input parameters, and (b) resulting distribution of annual seepage per meter of tunnel.

5.5 Pareto Frontier

The Pareto frontier can be considered to be the set of solutions to a multicriteria optimization problem, where the relative weights of the criteria are varied to examine the tradeoffs between competing objectives. Here, we determine the Pareto frontier by running multiple iTOUGH2 inversions, where the relative weights are adjusted in predefined increments. The grid-search option of iTOUGH2-PEST is used, where the parameter to be varied is the weight assigned to the two observation types, each representing a different objective. For each weight combination, an iTOUGH2 inversion is performed, and the mean residual of each observation type is extracted and used to create the Pareto frontier plot. In this example, iTOUGH2 controls iTOUGH2 optimization runs.

The optimization problem considered is a remediation design problem, where the tradeoff between two objectives is examined. These competing objectives are (1) maximization of contaminant removal within a specified cleanup time of 5 years, and (2) minimization of cleanup costs, simplified here as the total amount of water pumped from six wells during a pump-and-treat operation. The individual minimization problem of determining optimal pumping rates (assuming that the relative costs of pumping and residual contamination are known) is described in *Finsterle* [2005]. This optimization problem is now solved repeatedly for different weights of the two competing objectives. By giving higher weight to the remediation goal, pumping rates are expected to go up; conversely, if emphasis is placed on reducing pumping costs, the pumping rates will generally go down at the expense of increased residual contamination. The tradeoff between these two objectives is evaluated at 40 discrete points with relative weights (w_p and w_c) for the pumping cost and remediation objectives, respectively, under the constraint that $w_p + w_c = 1$. The weights are entered into the iTOUGH2 input file, which is created by the PEST template file *pareto.tpl* (Figure 34). For each weight combination, the optimal distribution of pumping rates in the six wells is determined by an iTOUGH2 optimization that minimizes both the (weighted) total amount of water pumped and the (weighted) residual contaminant mass. The total rate and residual contaminant mass after each optimization is extracted from the residual analysis section of the iTOUGH2 output file using the PEST instruction file *pareto.ins* (Figure 35). Plotting the two objectives against each other provides the Pareto frontier.

Figure 36 shows the iTOUGH2-PEST input file that performs a 40-point grid search. The only parameter adjusted is the weight of the pumping rate criterion, w_p; its value is varied from (almost) zero to (almost) one (the endpoints of the interval are avoided because no residual analysis is performed by iTOUGH2 if all residuals are zero). The second parameter (representing the weight given to the residual contamination criterion) is not a free parameter. It is tied to the first parameter using the equation $w_c = 1 - w_p$, which is implemented using the commands `>>>> TIED TO`, `>>>> ADD`, and `>>>> MULTIPLY`. The 40 iTOUGH2 inversions are invoked through the standard Unix script command *itough2* (or the equivalent WINDOWS batch file), which is provided as the executable.

The resulting Pareto frontier is shown in Figure 37, demonstrating that there is a relatively well-defined optimal solution, where both criteria can be met without too much tradeoff.

```
ptf #

> parameters

  >> generation
     >>> source: INJ_1
         >>>> annotation: Well SW
         >>>> value
         >>>> range: -0.75 -0.01
         <<<<

... other wells

     <<<
  <<

> observation
  >> times: 2 year
     4.999 5.0

  >> total generation
     >>> source: INJ_1 +5
         >>>> sum
         >>>> zero data
         >>>> weight:    #criterion_q#
         <<<<
     <<<
  >> total mass
     >>> model
         >>>> component: 3
         >>>> zero data
         >>>> weight:    #criterion_m#
         <<<<
     <<<
  <<

> computation
  >> stop
     >>> iteration: 15
     <<<
  >> option
     >>> l1-estimator
     <<<
  <<
```

Figure 34. Template file for creating an iTOUGH2 input file with adjustable weights for different observations that represent different objectives.

```
pif @
@RESIDUAL ANALYSIS@
@MASS IN PLACE     [kg]@       w   w   !resid_m!
@GENERATION RATE   [kg/sec]@   w   w   !resid_q!
```

Figure 35. Instruction file to for extracting residual contaminant mass in place and mean pumping rate.

```
> PARAMETER
  >> PEST
     >>> NONE
         >>>> NAME     : criterion_q
         >>>> VALUE
         >>>> GUESS    :  0.5
         >>>> RANGE    :  0.001 0.999
         <<<<
     >>> NONE
         >>>> NAME     : criterion_m
         >>>> VALUE
         >>>> GUESS    :  0.5
         >>>> TIED TO  :    1
         >>>> ADD      :  1.0
         >>>> MULTIPLY : -1.0
         <<<<
     <<<
  <<

> OBSERVATION
  >> PEST
     >>> UNIVERSAL: Pumping Rate
         >>>> DATA
         resid_q 0.0 1.0
         <<<<

     >>> UNIVERSAL: Residual Contamination
         >>>> DATA
         resid_m 0.0 1.0
         <<<<
     <<<
  <<

> COMPUTATION
  >> OPTION
     >>> GRID SEARCH: 39 intervals
     >>> PEST
         >>>> EXECUTABLE : “itough2 remi rem 10”
         >>>> TEMPLATE   : 1
                           pareto.tpl  remi
         >>>> INSTRUCTION: 1
                           pareto.ins  remi.out
         <<<<

         FILE: pareto.tpl
         FILE: pareto.ins

     <<<
  <<
<  .
```

Figure 36. iTOUGH2-PEST input file for running multiple iTOUGH2 inversions to create Pareto frontier.

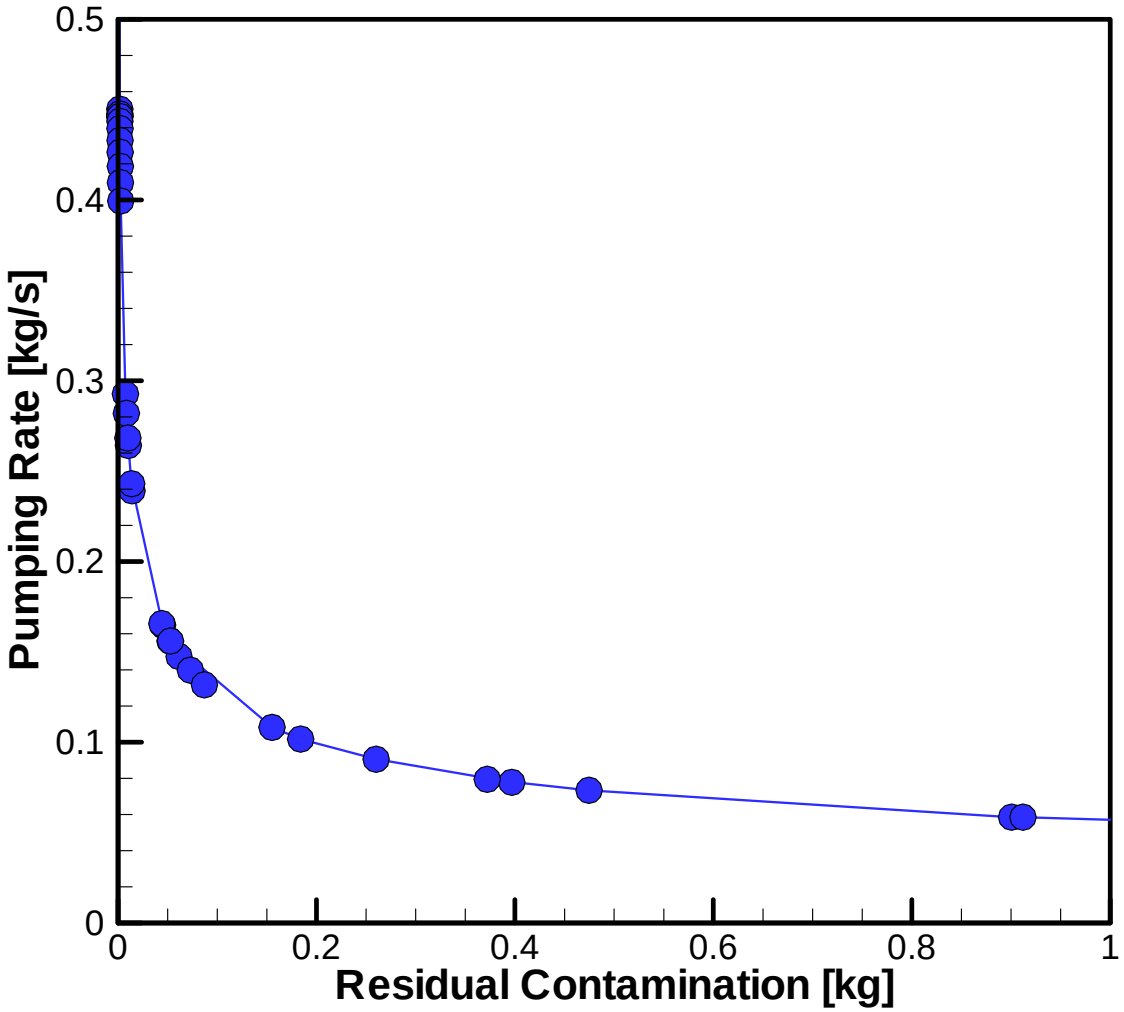


Figure 37. Pareto frontier.

6. CONCLUDING REMARKS

In the indirect approach to inverse modeling, optimization algorithms are wrapped around the numerical model whose parameters are to be estimated based on select output variables calculated by this model. Similarly, sensitivity analyses and uncertainty propagation analyses (specifically sampling-based methods) often treat the underlying model as a black box model. The fact that the optimization algorithms generally can be decoupled from the algorithms that solve the forward problem provides great flexibility in applying them to a large variety of scientific analysis and engineering design problems. On the other hand, a tight link between the forward and inverse problem also has significant advantages, specifically in terms of numerical accuracy, control of the forward simulator by the optimizer, and convenience in specifying input parameters and extracting output variables. The latter approach has been the tenet of iTOUGH2, which tightly integrates the TOUGH2 suite of nonisothermal multiphase flow simulators with routines for automatic model calibration, sensitivity, and uncertainty propagation analyses.

With the support of the PEST protocol as described in this manual, iTOUGH2 now has the flexibility and all the other advantages of a universal, model-independent optimization code. The investments made into the solution of the challenging, strongly nonlinear and often highly parameterized TOUGH2 inverse problems are now available for use in conjunction with any text-based numerical simulator or analytical equation solver.

7. ACKNOWLEDGMENT

I would like to thank John Doherty for making the PEST protocol and related parsing routines publicly available. I am also grateful to Yingqi Zhang and Mike Kowalsky for their careful reviews of this manual. This work was supported, in part, by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Wind and Geothermal Technologies, of the U.S. Department of Energy, and as part of the Subsurface Science Scientific Focus Area funded by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Resources under Award Number DE-AC02-05CH11231.

8. REFERENCES

Banta, E.R., M.C. Hill, E. Poeter, J.E. Doherty, and J. Babendreier, Building model analysis applications with the Joint Universal Parameter IdenTification and Evaluation of Reliability (JUPITER) API, *Computers and Geosciences*, 34, *310–319, 2008.*

Doherty, J., *PEST: Model-Independent Parameter Estimation*, Watermark Numerical Computing, Brisbane, Australia, 2008. http://pesthomepage.org/

Doherty, J., *FORTRAN 90 Modules for Implementation of Parallelised, Model-Independent, Model-Based Processing*, Watermark Numerical Computing, Australia, March 2007.

Finsterle, S., *Parallelization of iTOUGH2 Using PVM*, Report LBNL-42261, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1998.
http://esd.lbl.gov/TOUGHPLUS/manuals/iTOUGH2_PVM_Users_Guide.pdf

Finsterle, S., Demonstration of optimization techniques for groundwater plume remediation using iTOUGH2, *Environmental Modelling and Software, 21*(5), 665–680, 2005.

Finsterle, S., *iTOUGH2 User's Guide,* Report LBNL-40040, Lawrence Berkeley National Laboratory, Berkeley, Calif., 2007a.
http://esd.lbl.gov/TOUGHPLUS/manuals/iTOUGH2_Users_Guide.pdf

Finsterle, S., *iTOUGH2 Command Reference,* Report LBNL-40041 (Updated reprint), Lawrence Berkeley National Laboratory, Berkeley, Calif., 2007b.
http://esd.lbl.gov/TOUGHPLUS/manuals/iTOUGH2_Command_Reference.pdf

Finsterle, S., *iTOUGH2 Sample Problems*, Report LBNL-40042 (Updated reprint), Lawrence Berkeley National Laboratory, Berkeley, Calif., 2007c.
http://esd.lbl.gov/TOUGHPLUS/manuals/iTOUGH2_Sample_Problems.pdf

Liu, H. H., G. S. Bodvarsson, and S. Finsterle, A note on unsaturated flow in two-dimensional fracture networks, *Water Resour. Res., 38*(9), 1176, doi:10.1027/2001WR000977, 2002

Pan, L., *User Information Document for: WinGridder Version 3.0*, Document ID 10024-UID-3.0-00, 2007.
http://esd.lbl.gov/TOUGHPLUS/manuals/WinGridder-V3_Users_Guide.pdf

Poeter, E.P., and M.C. Hill, *Documentation of UCODE, a Computer Code for Universal Inverse Modeling*, U.S. Geological Survey Water-Resources Investigations Report 98-4080, 1998. http://water.usgs.gov/software/ucode.html

Pruess, K., C. Oldenburg, and G. Moridis, *TOUGH2 User's Guide, Version 2.0*, Report LBNL-43134, Lawrence Berkeley Laboratory, Berkeley, Calif., 1999.
http://esd.lbl.gov/TOUGHPLUS/manuals/TOUGH2_V2_Users_Guide.pdf

Spycher, N., G. Zhang, S. Sengor, M. Issarangkun, T. Barkouki, T.Ginn, Y. Wu, R. Smith, S. Hubbard, Y. Fujita, R. Sani, and B. Peyton, Application of TOUGHREACT V2.0 to environmental systems, *Proceedings*, TOUGH Symposium 2009, Lawrence Berkeley National Laboratory, Berkeley, Calif., September 14–16, 2009.

Xu, T., E.L. Sonnenthal, N. Spycher, and K. Pruess, *TOUGHREACT User's Guide: A Simulation Program for Non-Isothermal Multiphase Reactive Geochemical Transport in Variably Saturated Geologic Media*, Report LBNL-55460, Lawrence Berkeley National Laboratory, Berkeley, Calif., 2004.

Zhang, K., H. Yamamoto, and K. Pruess, *TMVOC-MP: A Parallel Numerical Simulator for Three-Phase Non-isothermal Flows of Multicomponent Hydrocarbon Mixtures in Porous/Fractured Media*, Report LBNL-63827, Lawrence Berkeley National Laboratory, Berkeley, Calif., September 2007.
http://esd.lbl.gov/TOUGHPLUS/manuals/TMVOC-MP_Users_Guide.pdf

Zhang, K., Y.-S. Wu, and K. Pruess, *User's Guide for TOUGH2-MP — A Massively Parallel Version of the TOUGH2 Code*, Report LBNL-315E, Lawrence Berkeley National Laboratory, Berkeley, Calif., 2008.
http://esd.lbl.gov/TOUGHPLUS/manuals/TOUGH2-MP_Users_Guide.pdf

Zhang, Y., C.M. Oldenburg, and S. Finsterle, Percolation-theory and fuzzy rule-based probability estimation of fault leakage at geologic carbon sequestration sites, *Env. Earth Sci.*, *59*, 1447–1459, doi:10.1007/s12665-009-0131-4, 2010.