

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA CIVILE, CHIMICA, AMBIENTALE E DEI MATERIALI

*CORSO DI LAUREA IN INGEGNERIA PER L'AMBIENTE E IL TERRITORIO
CURRICULUM EARTH RESOURCES ENGINEERING*

TESI DI LAUREA

in

Petroleum Geosystem

***HIGH-ENTHALPY GEOTHERMAL RESERVOIR MODEL CALIBRATION
USING PEST***

CANDIDATO
Silvia Armani

RELATORE:
Chiar.mo Prof. Villiam Bortolotti

CORRELATORE
PhD Dr. Ester Maria Vasini

Anno Accademico 2015/2016

Sessione III

ABSTRACT

The main purpose of this thesis work is focused on the use of PEST (*Parameter Estimation*) to calibrate numerical models of High Enthalpy Geothermal Reservoirs (HEGR). PEST is a parameter estimation and analysis of the uncertainties of complex numerical models tool, that can be instructed to work with a standalone simulator. So, the T2Well-EWASG was used as coupled wellbore-reservoir simulator for multiphase-multicomponent HEGR. The idea of this thesis work is that the possibility to implement some automation degrees in the wellbore-reservoir model calibration task would improve substantially the Reservoir Engineers work. To become familiar with PEST, it has been necessary a preliminary training to learn how to manage its input files, its keywords, and the utility programs having the function of verifying the correctness and consistency of the created files. Then, one of the examples of PEST manual (which Fortran source code is supplied) was reproduced and analyzed, and subsequently modified. In particular, starting from this example, a simple linear model with two free parameters, some changes have been performed: "fixing" a parameter to inhibit its change during the calibration; reading a more complex model output file respect to the original example; inserting dummy data that should not be processed and instructing PEST to consider only the data of interest; changing the model adding parameters to be calibrated, and including them in the analysis changing the PEST inputs files. Finally, these skills were applied to use PEST with T2Well-EWASG to calibrate a numerical model, relative to a real HEGR, previously calibrated via a trial and error approach in a PhD thesis work. Among the real data used there were also short production-tests done in a geothermal field located in the Dominica Commonwealth.

The preliminary results show that the PEST-T2Well-EWASG calibration system works fine, and that it is a useful tool that can improve the work of reservoir engineering.

Contents

<i>List of Figures</i>	1
<i>List of Tables</i>	2
1. INTRODUCTION	5
1.1 <i>System and model definitions and classification</i>	5
1.2 <i>Calibration of the model</i>	9
1.3 <i>High enthalpy geothermal reservoirs</i>	11
1.3.1 <i>Geothermal energy</i>	11
1.3.2 <i>Locating high-enthalpy geothermal fields</i>	14
1.3.3 <i>The pros and cons, and future of geothermal energy</i>	15
1.3.4 <i>Generalities on the simulation of geothermal reservoirs</i>	15
1.3.5 <i>Problems related to the use of numerical models for the simulation of geothermal reservoirs</i>	17
1.4 <i>T2Well in brief</i>	18
1.4.1 <i>Mass and energy balance and Drift Flux Model</i>	19
1.4.2 <i>EWASG module</i>	21
1.4.3 <i>Input and output files</i>	21
2. PEST TOOLS	27
2.1 <i>Model Input files and PEST template files</i>	28
2.1.1 <i>The parameter delimiter</i>	29
2.1.2 <i>Parameter names</i>	30
2.1.3 <i>Setting the Parameter Space Width</i>	30
2.1.4 <i>Preparing a Template File</i>	31
2.2 <i>Instruction files</i>	31
2.2.1 <i>How PEST Reads a Model Output File</i>	32
2.2.2 <i>The Marker Delimiter</i>	32
2.2.3 <i>Observation Names</i>	33
2.2.4 <i>The Instruction file keywords</i>	33
2.2.5 <i>Creation of an Instruction File</i>	38
2.3 <i>The PEST control file</i>	38
2.3.1 <i>Parameter Groups Section</i>	40
2.3.2 <i>Parameter Data Section</i>	41

3. <i>ADDITIONAL USED TOOLS</i>	45
3.1 <i>G95 compiler and Fortran language</i>	45
3.2 <i>CodeBlocks</i>	49
4. <i>APPLICATION OF PEST</i>	51
4.1 <i>PEST example of a bilinear model</i>	51
4.1.1 <i>Calibration of fixed parameter</i>	61
4.1.2 <i>Modification of the example model</i>	62
4.2 <i>New extended example model</i>	64
4.2.1. <i>How to read a more complex output file</i>	66
4.2 <i>T2Well numerical model calibration: multilayer high enthalpy geothermal system</i>	68
4.3 <i>T2Well-PEST</i>	71
CONCLUSIONS	83
<i>References</i>	85

List of Figures

Figure 1. General scheme of a model calibration.....	10
Figure 2. Summary scheme that relates all the notions linked to models explained in the paragraph 1.1 and 1.2.	11
Figure 3. The geothermal gradient ..	12
Figure 4. World map showing variation in surface heat flow.	13
Figure 5. Different possible types of grids.	19
Figure 6. TOUGH2 input file example.....	23
Figure 7. Example of PEST input file.	29
Figure 8. Example of PEST template file.....	29
Figure 9. Example of <i>Control file</i> , extracted from PEST User Manual Part I, 2016.	40
Figure 10. Statements order that is required in a unit of a Fortran program.	46
Figure 11. The CodeBlocks user interface.	49
Figure 12. Data of file <i>soilvol.dat</i> fitted in two straight lines: the “residual shrinkage” segment and the “normal shrinkage” segment.....	52
Figure 13. Two line model parameters scheme.....	53
Figure 14. The source code of the example model.....	54
Figure 15. <i>in.dat</i> input file.....	54
Figure 16. <i>out.dat</i> output file.....	55
Figure 17. Template file <i>in.tpl</i>	56
Figure 18. <i>in.pmt</i> file.	56
Figure 19. File <i>in.par</i> in which the values of SCALES are all equal to 1.0, OFFSETs equal to 0.0 and PRECIS and DPOINT are ‘single point.’	57
Figure 20. Instruction file <i>out.ins</i>	57
Figure 21. File <i>out.obf</i>	58
Figure 22. File <i>measure.obf</i>	59
Figure 23. Control file <i>twofit.pst</i>	59
Figure 24. Modified section of <i>twofit.pst</i> file.....	60
Figure 25. Comparison of experiment results calculated in different ways.	61
Figure 26. Modified control file <i>twofit.pst</i> . In the red circle there is the PARTRANS modified of parameter s_1	62
Figure 27. <i>twofit.par</i> changed after the modification of the control file.	62
Figure 28. Changes applied to the model code <i>twoline.for</i> . To notice the comments in red.	63
Figure 29. Modified <i>twoline.for</i> with two more parameters h and k . Highlighted in red the changed steps.....	65
Figure 30. New results from file <i>out.dat</i>	66
Figure 31. File <i>twofit.par</i> . New values of parameters estimated with PEST.	66
Figure 32. Modified code for <i>twoline.exe</i> in order to have a more complex output file.....	67
Figure 33. New format of output file <i>out.dat</i> after having modified the model executable <i>twoline.exe</i>	67

Figure 34. New version of file <i>out.ins</i> .	68
Figure 35. Conceptual model of the WW-01 well-reservoir system: it is possible to see the well WW-01 and the formation	69
Figure 36. WW-01 wellbore-reservoir model 2D vertical section	70
Figure 37. Initial pressure and temperature conditions assumed for the wellbore-reservoir model	70
Figure 38. Comparison between measured and simulated flowing pressures.	71
Figure 39. Input file for T2Well-PEST application.	73
Figure 40. PEST template file called <i>WW-01_0.9.tpl</i> .	73
Figure 41. File <i>.pmt</i> , where all the parameters names are listed, generated by using the utility tool TEMPCHEK.	74
Figure 42. File <i>.par</i> , where all the parameters names and values are listed, generated by using the utility tool TEMPCHEK.	74
Figure 43. <i>FStatus_2.ins</i> instruction file.	75
Figure 44. File <i>FStatus_2.obf</i> generated from INSCHEK tool.	76
Figure 45. File <i>experimental_data.obf</i> .	76
Figure 46. Control file generated with PESTGEN, called <i>tough2_WW01.pst</i> .	78
Figure 47. Final output file <i>FStatus_2</i> .	79
Figure 48. <i>tough2_WW01.par</i> containing the estimated parameters set.	80
Figure 49. Graph showing the matching between the experimental data, the manually simulated dataset and the values obtained from PEST calibration.	80

List of Tables

Table 1. Experimental data table	52
Table 2. Reservoir formation horizontal permeability as obtained by manual calibration of the model	71
Table 3. Correct values for the variation interval of the parameters	72

1.INTRODUCTION

The calibration of a model (i.e. the research of the best values to be assigned to its parameters) is a complex task that in general requires to spend a lot of work. This is particularly true in the field of the reservoir engineering where the possibilities to have reliable measured data is rare because of both the high measurement costs and the low reproducibility of measures. Moreover, a reservoir model is characterized by a huge number of different parameters and often the data refers to different measurement techniques using different measurement units. Well calibrated models will produce robust and reliable simulations. Therefore, the possibility to have available a reliable tool to automatize the process of calibration surely improve the task of the reservoir engineer.

In particular, we are interested to work with wellbore-reservoir coupled high enthalpy geothermal models. Therefore, we have used PEST (*Model-Independent Parameter Estimation*, a program software for the parameter estimation and analysis of the uncertainties of environmental models and in general of complex numerical models) coupled with T2Well-EWASG (a wellbore-reservoir simulator, which uses the equation of state EWASG (Equation-of-State for Water, Salt and Gas), dedicated to multiphase-multicomponent geothermal high enthalpy reservoir) to enhance the calibration task. It is noteworthy to highlight that, PEST-T2Well-EWASG could be also used to improve the interpretation of well-tests performed on geothermal reservoirs.

Before facing with the practical issues of coupling the PEST and T2Well and using their tools and keywords, it is useful to briefly clarify some fundamental concepts, necessary for understanding the concept of *simulation* in general. In particular, the meanings of the terms system, process, model, numerical model, simulation, geothermal reservoirs, etc., will be defined.

1.1 System and model definitions and classification

The *system* is defined as an interacting set of parts which form an individual “body” that allows the process (a transmission of energy, mass or information) to occur. In our case the system will be a geothermal reservoir. Generally, system has two categories of properties that characterize its

behavior: *parameters*, properties of the system invariant respect to the time, and *variables* that change through the time because of the interaction between the system and the external world. An example of a parameter of an aquifer could be the compressibility of water, while a variable could be the distribution of the water pressures inside the aquifer.

A system can be schematized as an entity, in which a process occurs, and that has relations with the external environment through an input and an output. The output (or response) of the system will depend on the actions that the external world exerts on the system (said inputs or solicitations or perturbations) and on what happens within the system itself, that is its internal state. The system output $U(t)$ can be mathematically represented as a function f that depends on another function that describes the internal state $S(t)$, on a function that describes the external solicitations $I(t)$ and on a set the set of model parameters p_1, p_2, \dots :

$$U(t) = f[S(t), I(t), p_1, p_2, \dots] \quad (1.1)$$

In the case of geothermal aquifers, one input can be the start of production of vapor from the production well with a defined flow rate; the response may be the pressure variation of the fluids, while the internal state can be described by the distribution of the saturations, and the parameter can be the porosity, the permeability etc., ..., .

In real cases, it is very difficult or often impossible to properly know all the characteristics of a system, or sometime it is impossible to work directly on the system itself, thus the system is studied using a simplified version called *model*. The model includes exclusively the more important aspects of the system that affect the analyzed problem. The model could be of different types: *physical model*, that can be a scale model in which the elements of the system are represented in scale; *analog model*, in which the system properties are represented through different physical quantities or *symbolic models* where the system is represented by symbols that can be handled. An example of symbolic model is the mathematical model in which equations and functions are used to represent the system and that can be divided in two categories: *analytical* and *numerical* models. Analytical models will provide, if any, an exact solution for the system, while numerical models develop an approximate solution that is reasonably close to the expected results. Given that very often it is impossible to find a solution of the analytical model, the numerical one is the only tool to represent adequately the behavior of the system. Moreover, the systems can be classified according to different criteria. In particular:

- According to the exchanges of matter and energy: a system is said *open* (from the thermodynamic point of view) when it exchanges both matter and energy with the external environment.

If the exchange of energy and matter is not allowed, it is said *isolated*, and if it is not allowed only the exchange of matter, is said *closed*. The geothermal reservoirs, in general, are open systems.

- Depending on the time variation of parameters: A system in which the same stress, repeated at different times, produces the same output, is said *invariant*. It is, however, inevitable that the parameters vary with time, as an effect of the aging process; the systems in which the parameters vary in time are called *variants*. It is possible to assume, however, that, despite a system is variant, the parameters that characterize it, are constants on sufficiently small time intervals. For example, the porosity, on long time intervals can vary as a result of the compaction of the rock, however it can be assumed that remains constant in a sufficiently short period of time, for example over a period of a few months.

Thus it is the rate of change of the parameters compared with the speed of evolution of the system which determines the invariance of the system.

- Depending on the time variation of variables: Depending on whether the variables that describe the system are constant or not in time, the systems are defined: *static* or *dynamic*. For a system to be defined as dynamic is sufficient that only one of the variables mutates over time.

- According to the values assumed by the variables: The type of variables of the system, continuous or discrete, subdivides the system in *continuous* or *discrete respectively*. In the first case the variables have a biunivocal correspondence with the real number set. In the second case the variables have a biunivocal correspondence with the set of natural numbers, or with a finite subset of them.

- Depending on the properties of relations: If the relationships between dependent variables and independent variables allow the determination of the value of a dependent variable in a unique way, the system is said *deterministic*. Otherwise, if the relationships among the dependent variables and the independent ones require the adoption of statistical relations, it is called *stochastic* or *probabilistic* system.

Lastly, observing the output of a system as a result of the input solicitations, the concept of steady-state and transient can be defined. The steady state is the state in which the system has

finished its evolution and produces a constant output or tending asymptotically to a value, said *steady output*. The transient state is instead the state in which the system is in evolution between two states of regime and the output tends to stabilize towards the steady output of the second state.

1.2 Calibration of the model

After the selection of the appropriate model to represent the system under investigation, there is the *interpretation* phase, in which the system output data obtained from the preliminary tests are interpreted using the model. This phase is used in important activities like the model calibration and analysis of sensitivity and error propagation [Bortolotti, 2013].

More specifically, the *calibration* operation involves the estimation of the values of constants and parameters regarding the model structure. In order to do this, it is necessary to compare the observed variables with that of the output produced by the run of the model and to try to reduce the discrepancies between the two set of variables. The observed variables are obtained from experiments, thus in general, the estimation step looks for the values of parameters that have the higher probability of being accurate taking into account the error tolerance: *trial-and-error* method (general scheme shown in **Figure 1**).

This procedure could be implemented by hand, with severe limitation, or by means of specific software. Another way to calibrate the model is to use parameters and constants values derived from a different model estimation in another location similar to the currently studied area. This last method should be utilized by experienced users only.

The sensitivity analysis is a procedure to identify the intensity of the model output especially as a consequence of the variation of its parameters. In other words, the research of the parameters that more influence the behavior of the model (and therefore of the system). This analysis is important to figure out which laboratory measurements or in situ must be improved. Parameters that do not affect significantly the model do not require to be measured or estimated with high precision. While the error propagation analysis allows to study the impact of the uncertainty of parameter values on the predictions made by models, through the analysis of the propagation of errors or Monte Carlo simulation, which provides correlations between parameters.

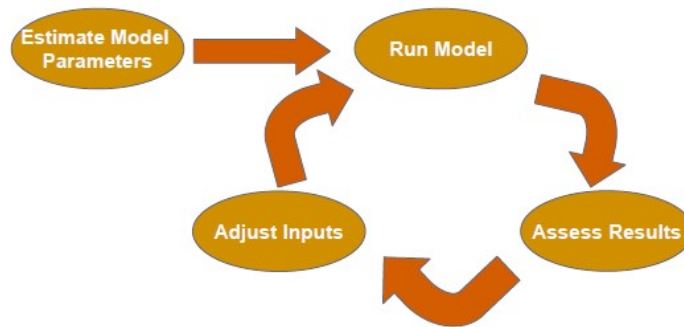


Figure 1. General scheme of a model calibration [Taylor et al, 2012].

When a satisfactory estimation of the parameters of the considered model have been obtained, the model should perform adequately the scope for which it was created, therefore its behavior must be checked. This process of the verification of a calibrated model is called *validation* [Institute of Transportation Engineers, 1992].

The *conventional simulation* technique determines the output of a calibrated model for a given set of initial conditions and time history to obtain information regarding the system behavior in its future evolution.

The *inverse simulation* is the reverse of the previous type of simulation, in fact in this case, the time history of a selected system *output* variable is laid down and the algorithm of the inverse simulation bring the investigator to calculate the time history of the corresponding *input* variable. *Inverse modeling*, parameter estimation, history matching and model calibration are terms essentially describing the same technique. They aim to assess the best model and its parameters for predicting the behavior of a dynamic system [Finsterle, 2007].

In **Figure 2** is shown a summary scheme that relates all these notions explained above.

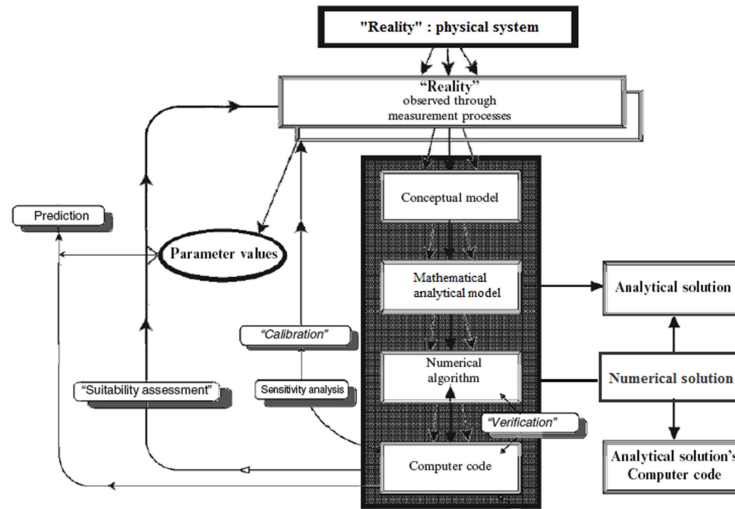


Figure 2. Summary scheme that relates all the notions linked to models explained in the paragraph 1.1 and 1.2 [Baveye et al., 2007].

1.3 High enthalpy geothermal reservoirs

1.3.1 Geothermal energy

Energy from the Earth's interior that flows through its surface is on average very low. Anyway, it is sufficiently abundant to make it locally worth exploiting. In fact, only in the top 3 km of the Earth's crust stores an estimated 4.3×10^7 EJ of thermal energy due to the rocks temperature and their thermal capacity. The principal geothermal potential is from heat flowing through the crust. Where geothermally heated, water rises to the surface in hot springs. Sometimes in this case, it has been used *directly* for heating, recreational and horticultural purposes since Roman times, while this type of energy was used *indirectly* for the first time when geothermally generated electricity was produced in 1904 at Larderello, Italy.

The interior of the Earth becomes hotter with increasing depth for two reasons:

- heat is generated by the *decay* of some natural radioactive isotopes (uranium-238 and uranium-235, thorium-232 and potassium-40);
- heat flows from the hot interior by means of convection and conduction.

The geothermal heat flows through the Earth determines a *geothermal gradient* (the rate at which temperature increases with depth). The geothermal gradient variation beneath the continental

surface is shown in **Figure 3**, where it is possible to notice that temperature increases much more with depth in the lithosphere than in the deep mantle.

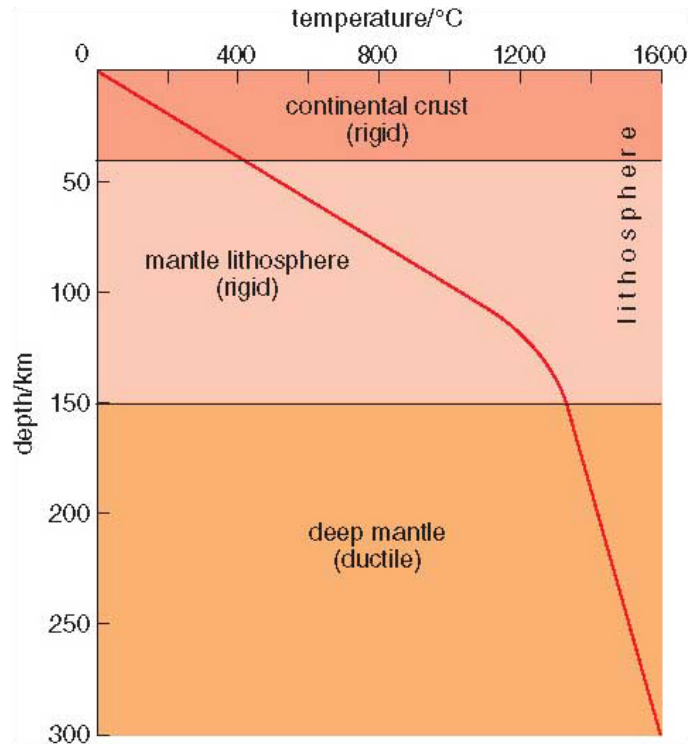


Figure 3. The geothermal gradient, the temperature increases more rapidly with depth in the lithosphere than it does in the deep mantle (<http://www.open.edu/openlearn/science-maths-technology/science/environmental-science/energy-resources-geothermal-energy/content-section-1>).

Even if they are not melted rocks, rocks forming the deep mantle are hot enough to behave in a ductile way. Parts of the hotter mantle rise slowly because they are slightly less dense than cooler mantle, this is a physical transport of heat towards the surface. This process is called *convection* and is an efficient means of heat transport.

Except at plate boundaries and hotspots convection, this method does not transfer heat in the lithosphere. Instead heat is transferred by *conduction*, which is a far less efficient process than convection, so the lithosphere acts as an insulating blanket (trapping the heat rising from below). Harvesting geothermal energy depends on the relationship between heat flow through the lithosphere and the local geothermal gradient.

The most favorable areas for geothermal exploitation seems to be those in volcanically active regions, at destructive and constructive plate margins. It is less obvious that is also worth exploiting some areas that are remote from volcanically active regions.

The local potential of geothermal energy is described by the *enthalpy* H of the area (total energy content of the geothermal system that lies below it). The enthalpy of a system cannot be measured directly: only a change or difference in energy has physical meaning. Enthalpy is a thermodynamic potential, so in order to measure the enthalpy of a system, one must refer to a defined reference point. Therefore, what is measured is the change in enthalpy, ΔH . At constant pressure, the ΔH equals the energy transferred from the environment through heating (the heat flow through the crust, measurable at the surface).

The equation of enthalpy is:

$$H = U + pV \quad (1.2)$$

Where, U is the internal energy of the system, p is the pressure of the system, V is the volume of the system.

The **Figure 4** shows that large areas of the Earth's surface have basically the same heat flow. However, the temperature at depth in the crust depends on how conductive the different rocks are, and this in turn depends on their physical properties. This subsurface temperature is crucial in evaluating whether or not an area constitutes a geothermal resource.

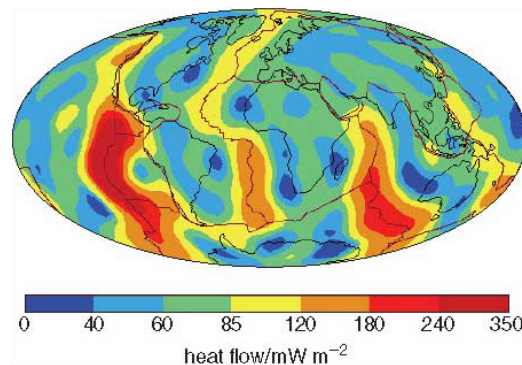


Figure 4. World map showing variation in surface heat flow in mW m^{-2} in relation to continental and oceanic crust, and major plate boundaries. The colors relate to the scale in mW m^{-2} . There are areas in continents where heat flow does rise to very high values, but they are too small to show on this map (<http://www.open.edu/openlearn/science-maths-technology/science/environmental-science/energy-resources-geothermal-energy/content-section-1>).

In a general classification, there are three types of area with geothermal potential; those with high, medium and low enthalpy.

- High-enthalpy areas are those where subterranean water and steam are at temperatures greater than $180\text{ }^{\circ}\text{C}$;

- Medium-enthalpy areas those where temperatures range between the boiling temperature of water (100 °C) and 180 °C;
- Low-enthalpy areas where temperatures are lower than 100 °C.

Using geothermal energy effectively speeds up heat flow and locally increases the heat lost by the Earth. But geothermal heat is continually supplied, so hot water and steam are depleted faster rather than the energy resource itself. However, if hot fluids removed are replaced by cold water pumped to recharge the geothermal field, the cold water heats up again. Carefully managed recharge enables the enthalpy of an area to be exploited continuously.

Geothermal fields usually need three characteristics to be exploited successfully:

1. An energy source to heat the fluids;
2. An aquifer, either natural or artificially created, to act as a reservoir for fluids that can transport heat energy to the surface;
3. A cap rock with low permeability to seal in these geothermal fluids (in a similar way to the trapping of petroleum). [Sheldon, 2005; Smith, 2005].

High- to medium-enthalpy geothermal resources are conventionally divided into water- or vapor-dominated fields. Vapor-dominated systems where the steam is at high temperature and dry tend to be the most economically viable geothermal resources, because the pV term in the enthalpy equation is high. Vapor-dominated fields also suffer less from problems of corrosion from the mineral-laden waters that are typical of deep aquifers. When such hot water passes through pipelines, not only it is highly corrosive, but solids dissolved in it precipitate in the pipes as temperature decreases, and could clog them, [Sheldon, 2005].

1.3.2 Locating high-enthalpy geothermal fields

The research of potentially useful geothermal fields is focused initially on locating rocks that have been chemically altered by natural geothermal fluids, looking for obvious surface features of geothermal activity (geysers and hot springs). Then, the fluid flow measurement through the field permits the estimation of the probable economic potential of the reservoir.

After that a promising resource has been located, the exploration wells are drilled. However, given the high pressures and temperatures typical of a geothermal resource, special precautions must be taken. For example, once the drill stem penetrates a superheated water saturated zone,

the liquid will flash to steam as the pressure drops and this is very dangerous. The well head itself is capped with valve gear to regulate the flow of fluids, and the whole assembly is then conveyed to an electricity generating plant [Sheldon, 2005].

1.3.3 The pros and cons, and future of geothermal energy

If correctly exploited, the geothermal energy is renewable but the geothermal fluids emit gases such as CO₂, H₂S, SO₂, H₂, CH₄ and N₂ when used for electricity generation. However, geothermal power plants are usually sited in areas of natural geothermal activity, where such emissions occur anyway. Other potential pollutants are various ions dissolved in the geothermal fluids, but these are almost always returned to the reservoir when the spent fluids are re-injected. Regarding safety, accidents are usually rare. Protracted geothermal exploitation can induce ground subsidence, although this is minimized by re-injecting spent fluids. Small intensity earthquakes can sometimes result from large-scale exploitation. Anyway, most high-enthalpy geothermal areas are naturally predisposed to seismicity.

The main disadvantage of geothermal power generation is that suitable high-enthalpy areas are geographically very restricted, many being in areas of low population density (or under the sea). On the contrary, the low-enthalpy potential of normal heat flow is universal and potentially useful for heating and even air conditioning, given the necessary investment.

In the short term, even twenty- to forty-fold growth in geothermal capacity will not result in it being a significant contributor to global energy needs by 2020. Geothermal power plants are generally much smaller than fossil-fuel and nuclear generators; tens of MW, rather than a few GW for the biggest “conventional” plants. To replace a single GW fossil fuel power station requires from 20 to 33 geothermal plants. The seeming benefit of local and 'environmentally friendly' geothermal power generation, because of economic factors related to their construction, delays its adoption at national to international scales [Sheldon, 2005].

1.3.4 Generalities on the simulation of geothermal reservoirs

The simulation of geothermal reservoirs is closely related to the one of the hydrocarbon reservoirs, which began in the '50s. The simulation by means of numerical models of the subsurface has rapidly evolved in recent years, thanks to the use of ever more advanced digital techniques and the availability of computers with computing capabilities continuously growing. It

has also increased the demand for a quantitative estimation of movement and heat transfer processes in the subsurface.

A good methodology in the creation of models allows one to increase the reliability of the simulation results, in fact, the determination of the objectives of the operation at the start of the modeling and the possible results, allows a more effective use of simulation [Bedient, 1999].

Moreover, although the simulation of a geothermal reservoir is often required by the competent authorities to grant permissions to exploitation, it is often necessary to integrate a numerical simulation with other tools in order to get a better analysis of the site on which it must operate.

A typical scheme of the project of numerical models is the following:

1. Definition of the simulation objective: for example, in the case of geothermal reservoirs can be the prediction of the values of pressure and temperature as a result of putting into production of new wells;
2. Development of a conceptual/geological model of the system (geometry, zoning, spatial distribution of petrophysical properties, initial saturation in fluids and initial pressures, etc...);
3. Identification of the thermodynamic parameters of the fluids in the reservoir and their spatial variations; definition of the curves of relative permeability and capillary pressure;
4. Choice of the equations that govern the processes and of a simulation software. Both must be verified: the equations must describe the physical, thermodynamic, chemical and biological processes that has been established to simulate, while the code, that implements these equations, must be verified by comparing the numerical results with the analytical solution to a known problem;
5. Definition of the grid ("mesh"): it defines the discretization in blocks of the system's spatial domain. It must be appropriate to reproduce the volume occupied by the field, of the boundary conditions and of particular conformations of the subsurface such as faults or fractures;
6. Model initialization: to each block are assigned the values of petrophysical parameters of the rocks, and initial thermodynamic and fluid dynamic values of fluids present in the reservoir. The combination of phase 5 to 6 constitute the so-called pre-processing, namely the creation of the data needed to process the numerical model, which will be stored in a simulator input file;
7. Model calibration or history matching;
8. Sensitivity analysis;
9. Predicting the behavior of the model in terms of future programs of the reservoir cultivation, or use in forecast mode. The analysis of results is also called post-processing;

10. Corrections and changes to the model according to the results of the simulations.

1.3.5 Problems related to the use of numerical models for the simulation of geothermal reservoirs

A lack of the use of numerical models in the simulation of geothermal systems is that they require large amounts of input data, which are often too expensive and difficult to obtain, so that often either simplifying assumptions are made or values obtained from the literature are adopted, that cannot be closely representative of the real properties of the reservoir.

The values of parameters measured in situ are intended as the average or total values of the area involved (for example by means of pumping tests, transmissivity and storage coefficient of the aquifer are obtained), which cannot reflect a particular local situation of the field. On the other hand, values obtained in the laboratory (such as permeability or porosity) on cores, are only representative of the area in which the sampling of the material has been made, which can have very different characteristics from the surrounding areas.

The attempt to model a system, of which there are few data available, is the same a useful operation as it allows one to identify areas where a greater number of information are needed, increasing in this way the understanding of the reservoir [Bedient, 1999].

The need to have results in reduced times requires the use of very powerful computers and therefore expensive; despite the adoption of the most powerful computers available on the market, the detail of the grids cannot massively increase, as well as the precision of calculation, all this to the detriment of the accuracy of the final results of the simulation.

The use of numerical models in the oil field has highlighted the need and the priority to focus the efforts on finding the most likely geological model of the reservoir and its internal structure and share the tasks, within the simulation team, in function of the individual skills [Chierici, 2004].

The calibration of the model, obtained by means of the solution of the so-called *inverse problem*, allows to obtain a *validated* model; such model is not necessarily unique: they may exist in fact several models (different distributions of parameters) that reproduce the past history of the reservoir, but only one can be used as the real one.

Although the model validation phase is mandatory, it does not guarantee that the validated model can forecast the future behavior of the reservoir [Chierici, 2004]. It is also no possible to prove that the results of a simulation are correct [Bedient, 1999].

The sensitivity analysis is performed by varying, within a reasonable range, the values of one parameter at a time and comparing the results of the series of simulations. This allows one to determine the influence of the uncertainty of parameters on the final response of the simulation. This procedure allows one to determine the most likely values of parameters; the studies of sensitivity are the most realistic way of use of numerical models for the prediction of reservoirs behavior [Chierici, 2004].

1.4 T2Well in brief

T2Well is a coupled wellbore-reservoir numerical simulator for non-isothermal, multiphase, multicomponent flows. This software was created as an extension of the numerical reservoir simulator TOUGH2 to calculate the flow simultaneously and efficiently in both the wellbore and the reservoir, by introducing a special wellbore subdomain into the numerical grid. For the wellbore subdomain grid-blocks, the 1D momentum equation of the mixture (which may be two-phase) is solved as described by the Drift-Flux Model (DFM, explained later in paragraph 1.4.1), [Pan and Oldenburg, 2012]. The equations are numerically solved with the Integrated Finite Difference Method. The IFDM is a numerical method for approximating the solution to differential equation using finite difference equations to approximate derivatives [Bortolotti, 2015]. This method is characterized by a geometrical grid constraint: the connecting segment between the centers (or nodes) of two close blocks must be perpendicular to the interface between the two blocks. Structured grids always satisfy this constraint, as the blocks are either cubes or parallelepipeds or radial sections, whereas in case of unstructured grids, the Voronoi tessellation must be used [Berry et al., 2014]. In **Figure 5** are reported the different possible types of grids as reported by Berry et al., 2014.

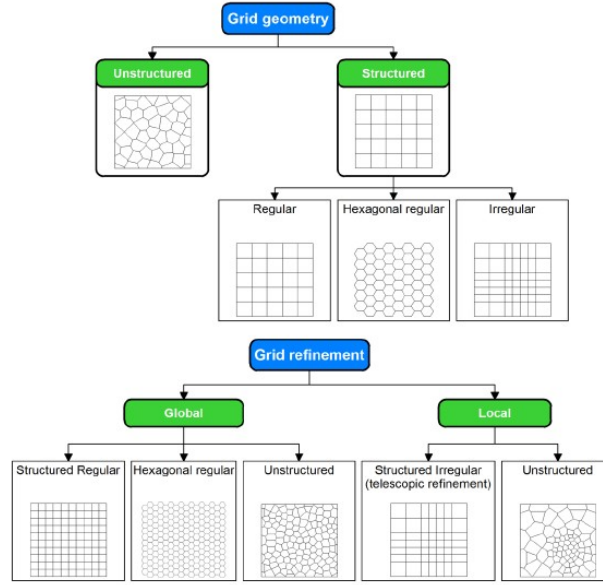


Figure 5. Different possible types of grids [Berry et al., 2014].

T2Well coupled with an adequate Equation of State (EOS), that manages the fluids thermodynamic of the system, can be used to simulate the commonly exploited high enthalpy geothermal systems. To deal with high enthalpy geothermal reservoir characterized by water, salts and one non-condensable gas, T2Well was coupled with the EOS EWASG (Equation-of-State for Water, Salt and Gas, see paragraph 1.4.2).

In the following subparagraphs, there will be both a generic explanation of the equations used by T2Well and of the structure of its input and output files.

1.4.1 Mass and energy balance and Drift Flux Model

The equations for the mass and energy conservation used by T2Well have the same structure as in TOUGH2:

$$\frac{d}{dt} \int_{V_n} M^k dV_n = \int_{\Gamma_n} \mathbf{F}^k \cdot \mathbf{n} d\Gamma_n + \int_{V_n} q^k dV_n \quad (1.3)$$

Where V_n is the volume of an arbitrary element of the system and Γ_n is the closed surface that bounds the element volume. On the left side of the equation 1.3, which is the accumulation term, M^k represents the quantity of mass or energy per volume, where the label κ can assume $1, \dots, NK$ values when it represents the mass components (that could be water, air, solutes or H_2) and $NK+1$

values in energy balance case. On the right side, the vector \mathbf{n} is the normal to the surface element $d\Gamma_n$ and F^k represents the mass or heat flux term, the second integral with q^k denotes sinks and sources terms [Pruess et al., 1999].

The main differences from the equations used for the porous media by TOUGH2 are in the energy flux, energy accumulation and in the computation of phase velocity in the subdomains representing the wellbore. T2Well uses the DFM briefly described in the following, to calculate the velocity of both liquid and gaseous phases.

The DFM was first developed by Zuber and Findlay (1965) and it represents a valid alternative for the two-phase flow study in a pipe. More in detail it is used for the phase velocities determination without solving the equation of the momentum for each phase. This model is based on some empirical constitutive relationships, in which all the variables must be considered either as area-averaged or constant over a cross-section. The main relationship is the following equation:

$$v_G = C_0 j + v_d \quad (1.4)$$

which establishes that the gas velocity v_G , can be related to the volumetric flux of the mixture j , and the drift velocity of the gas, v_d , via the parameter C_0 , named profile parameter, which takes in account for the effect of local gas saturation and velocity profiles over the pipe cross-section [Pan et al, 2011].

In the case of subdomain grid-blocks representing the wellbore, the 1D momentum equation of the mixture, which may be two-phase, (liquid and gas for example) is solved according to the DFM. Here the mixture velocity is calculated by solving numerically the momentum equation, whereas the velocities of individual phases are calculated from the mixture velocity. More in detail, the pressure gradient, gravity component, and time derivative of momentum are treated fully implicitly, while the spatial gradient of momentum is treated explicitly. The friction term is calculated with a mixed implicit-explicit scheme [Pan and Oldenburg, 2012].

1.4.2 EWASG module

Considering that geothermal fluids usually consist of complex mixtures of water, salts and gases, and their thermodynamic and transport properties affect reservoir conditions and performance, as mentioned before, the EWASG module was developed for the TOUGH2 multi-purpose numerical reservoir simulator to handle three-component fluid mixtures of water, sodium chloride and a slightly soluble non-condensable gas (NCG) [Battistelli et al., 1993; 1997].

Since 1999 EWASG has been available to the public within the TOUGH2 V.2.0 package [Pruess et al., 1999]. Sodium chloride (NaCl), is used to simulate the effects of dissolved salts, whereas the effects of one NCG were included following a general formulation in which the NCG can be chosen by the user among a set of implemented NCGs (in the TOUGH2 V.2.0 package, the NCGs available are CO₂, CH₄, H₂, N₂, air and O₂ added more recently). So, the multi-phase system is assumed to be composed of three mass components: water, sodium chloride, and one NCG, (carbon dioxide for conventional geothermal applications).

All the relevant thermophysical properties are evaluated by means of a subroutine structure, so that the correlations currently used can be easily modified as soon as more reliable experimental data and correlations become available. It is assumed that the three phases (gaseous, liquid, and solid) are in local chemical and thermal equilibrium, and that no chemical reactions take place other than interphase mass transfer [Calore and Battistelli, 2003].

As mentioned before, the thermo-physical property correlations used in EWASG, are accurate for different conditions of geothermal reservoirs, as an example, the fluid pressure can be up to 80 MPa, temperatures can vary in a range between 100 and 350°C, CO₂ partial pressure can arrive up to 10 MPa and salt mass fraction up to halite saturation.

1.4.3 Input and output files

Being T2Well based on the TOUGH2, their input files are very similar, however there are small differences that will be explained in this paragraph. Let's first describe shortly the TOUGH2 input file.

To characterize a flow system, TOUGH2 requires many data including hydrogeological parameters and constitutive relations of the permeable medium, as relative and absolute permeability, porosity, capillary pressure; fluids thermophysical properties, initial and boundary

conditions of the flow system, sink and sources. It requires also the specification of space-discretized geometry, program options, computational parameters and time-stepping information. All these data must be provided to TOUGH2 in one or more ASCII data files within a fixed format with up to 80 characters per record. All the information is organized in blocks and reported in the standard metric (SI) unit system.

Here below, the main keywords used in a typical TOUGH2 input file are described briefly (it is possible to see the complete list of the input file and keywords in the TOUGH2 v.2.0 manual [Pruess et al.,1999]):

- ROCKS describes the rock types providing also the hydrogeological parameters as porosity, permeability, heat conductivity and specific heat...;
- ELEME and CONNE provide the mesh geometric information like nodes coordinates, areas of interface; in ELEME the rock type for each grid block is also specified;
- MULTI specifies the fluid components number and grid block balance equations;
- SELEC provides the thermophysical properties data;
- PARAM defines all the parameters for the computation such as program options, time steps and simulated time;
- GENER is used to define sources and sinks;
- INCOM and INDOM permit to define the initial conditions.

An arbitrary record, with arbitrary data, that do not start with a TOUGH2 keyword will be ignored, permitting to directly insert comments in the input file. TOUGH2 will collect all these comments and write them as records in the output file. **Figure 6** shows a piece of a TOUGH input file with all keywords explained before.

```

ROCKS-----1-----*-----2-----*-----3-----*-----4-----*-----5-----*-----6-----*-----7-----*-----8
CAPR1 3 2600. 0.01 1.0E-19 1.0E-19 1.0E-18 2.30 920.
      3 .40 .05 1. 1.
      7 0.44380 8.01E-2 5.792e-07 5.E7 1.
SHAL2 3 2600. 0.05 24.0E-15 24.0E-15 24.0E-15 3.50 920.
      3 .30 .01 1. 1.
      7 0.44380 8.01E-2 5.792e-07 5.E5 1.
DEEP3 3 2600. 0.03 5.0E-15 5.0E-15 5.0E-15 3.50 920.
      3 .40 .05 1. 1.
      7 0.44380 8.01E-2 5.792e-07 5.E6 1.
ATM04 3 2600. 0.99 1.0E-19 1.0E-19 1.0E-18 2.30 99920.
      3 .40 .05 1. 1.
      7 0.44380 8.01E-2 5.792e-07 1.E5 1.
SELEC-----2-----3-----4-----5-----6-----7-----8-----9-----10-----11-----12-----13-----14-----15-----16
1 0 1 0 3 2
3.
MULTI-----1-----*-----2-----*-----3-----*-----4-----*-----5-----*-----6-----*-----7-----*-----8
3 4 3 6
START
PARAM-----1-----*-----2-----*-----3-----*-----4-----*-----5-----*-----6-----*-----7-----*-----8
8 49999 1001000301 0020 00300 3
.0 8.64E+11 -1. 9.8065
1.E-9 1.0
1.E-5 1. 1.e-8 1.e+4
235.0 .0030 0.000001 3.0E5
|
ELEM
A11 1 ATM04 .1000E+52 .1000E+05 0.5000E+020.5000E+020.5000E+03
A21 1 CAPR1 .1000E+070.0000E+00 0.5000E+020.5000E+020.4500E+03
A31 1 1.1000E+070.0000E+00 0.5000E+020.5000E+020.3500E+03

```

Figure 6. TOUGH2 input file example.

All the specific parameters of T2Well are stored under the keyword ROCKS, and they must be introduced only for the wellbore ROCKS types, as follows:

```

ROCKS-----1-----*-----2-----*-----3-----*-----4-----*-----5-----*-----6-----*-----7-----*-----8
wellb NAD 2600.0 1.0 1.0E-13 1.0E-13 1.0E-13 -2.1 1000.0
0.0 0.0 2.1 0.0 0.0
1 0.2 0.1 0.9 0.7
1 0.0 0.0 1.0
NTEMP RWB UHT
ZF(1) TF(1)
ZF(2) TF(2)
...
ZF(NTEMP) TF(NTEMP)

```

The first four records are the traditional ones, referring to the properties of the rocks (density, porosity, permeability, thermal conductivity, in the first record specific heat, in the second record pore compressibility and expansion, heat conductivity in desaturated conditions, tortuosity factor and Klinkenberg parameter, in the third record relative permeability function parameters and in the fourth record capillary pressure function). The new records follow and are read only if NAD (first record, second field) is larger than 3. Thus, after the record of the capillary pressure, it is possible to introduce the following parameters:

NTEMP: number of couples (cell depth; formation temperature) with which the code will determine the corresponding formation temperature at the wellbore cell depth. If NTEMP is equal

to 0, the temperature of the formation is taken equal to the initial temperature of wellbore, otherwise the code will read the couples depth-temperature in the following NTEMP records;

RWB (m): the completion radius;

UHT: the over-all heat transfer coefficient;

It is possible, by proper rock type introduction, to take into account different parts for the same wellbore, characterized by different parameters, such as the wellbore and the completion radius, the over-all heat transfer coefficient. Furthermore, characterizing the wellbore, for example, with two different rocks type, it is possible to apply an analytical computation of heat exchange only for a wellbore portion, for which, for modelling purpose, the surrounded formation must not be explicitly modelled. In fact, the analytical computation of heat exchange between wellbore and reservoir is possible only if the model is composed by the wellbore (no grid blocks of surrounding formation) and it is activated if the thermal conductivity (CWET) of the wellbore rock type is negative. With the keyword NAD it is possible to select the type of wellbore reservoir heat exchange analytical relation [Vasini, 2016].

Regarding the main output file, here will be reported only a brief summary, for more details see TOUGH2/ECO2N manual [Pruess, 2005].

The formats of the output file are basically the same as that of TOUGH2/ECO2N except that a profile of velocities (of the mixture, gas, and liquid phase) in the wellbore is added next the regular profile output (at user specified output steps). In addition, some informational outputs regarding wellbore cells, connections, and their geometry features are also included in the front of the main output file. In addition to the TOUGH2 output, T2Well gives two additional information: FStatus and FFlow that are briefly described below together with the other files with fixed name. These description are taken from the "*TWell/ECO2N Version 1.0: Multiphase and Non-Isothermal Model for Coupled Wellbore-Reservoir Flow of Carbon Dioxide and Variable Salinity Water*" [Pan et al., 2011].

-FStatus: five status variables of each wellbore cell at every time step and depth: time, distance to wellhead, gas saturation, mass fraction of CO₂ in liquid, pressure, temperature, gas density;

- Flow: five variables of each wellbore connection at every time step and depth: time, distance to wellhead, liquid phase mass flow rate, gas phase mass flow rate, liquid phase velocity, gas phase velocity, mixture velocity;

- FOFT: optional (transient output of state variables for user-specified cells in main input file. First two variables are the index and the simulation time, respectively. They are followed by the cell index and five variables at the cell in turn of each cell listed in FOFT section of the main input file. The five variables are pressure, gas saturation, mass fraction of CO₂ in liquid phase, mass fraction of salt in liquid phase, and the temperature, respectively.);
- COFT: optional (transient output of flow rate and velocity for user-specified connections in main input file. Data structure here are very similar to FOFT, except for that here the five variables are gas phase mass flow rate, liquid phase mass flow rate, gas phase velocity, liquid phase velocity, and total CO₂ mass flow rate, respectively, for each connection listed in COFT section in the main input file.);
- DOFT: a time series of total liquid and gas volume (see TOUGH2/ECO2N manual for details).

2. PEST TOOLS

The acronym *PEST* stands for *Parameter Estimation* and was originally created for model calibration in which the parameter values are calculated with inverse modelling by matching the output of the model to the values of measurements of the system state. PEST is different from the others parameter estimation software because it works in a model-independent way, in fact PEST interacts with the model through its own input and output files.

In a few words, PEST runs many times a executable program representing the model (program-model in short) every time with different values of some parameters of a program-model in order to minimize the difference from the output of the models and a set of experimental data. Therefore PEST must be able to interact with the input and output files used by the program-model.

PEST run a program-model through a call to the operating system (OS), process named *system call*, that has the same effect as typing the model name in the command line window. For this reason the program-model must be accessible to a user through the command line. In practice, the folder in which the model is stored the program-model should be inserted in the PATH environment variable of the OS so that the OS knows where to find it.

If a model prompts the user for keyboard input, then the process of calibration apparently seems not to be automatable, but this situation can be easily accommodated. In fact the keyboard responses to the model's prompts can be placed into a text file and the model will import the answers from this file (see below for details). In this way, PEST can run the model without needing any user involvement [Doherty, 2016].

Therefore using PEST it is possible to automatize the calibration of a generic model as long as its executable is accessible and usable via a terminal way (command line approach).

More in detail, to execute a calibration run, PEST requires three types of input file:

- *template files*, in which all the parameters are defined for each input file of the studied model;
- *instruction files*, one for each model output file on which model-generated observations exist;

- *control file*, supplies PEST with all template and instruction files names, corresponding model input and output files names, the problem size, control variables, initial parameter values, measurement values and weights, etc. . . .

Once having built these files, it is possible to use the utility programs TEMPCHEK, INSCHEK and PESTCHEK to check both their correctness and consistency.

2.1 Model Input files and PEST template files

PEST provides a set of parameter values which it wants the model to use for a particular run. These data are generally taken from the model input file.

Some models read their data from the terminal prompt, the user must supply these data in response to model prompts, but this can also be done through a file. It is possible to redirect the responses to the model, that the user has been previously written in a file, through the terminal redirection < symbol (redirection is a form communication between process of a many OSs). Therefore if the model in question is run using “model” command, and the responses are typed in advance to a file named for example *file.inp*, then PEST can run the model without supplying the terminal input using the command:

```
model < file.inp
```

If the *file.inp* contains some parameters that PEST must optimize, it is possible to create a template for it as if it were any other model input file.

The template file instructs PEST to read the input files of the simulator and is necessary only for those input files which contain parameters that require an estimation. Generally, an input file of a model can be of any length, however PEST requires that it is less than 2000 characters in width. The same applies to the template files and it is suggested that template files have to be provided with “.*tpl*” extension in order to distinguish them from other file types. Basically a template file is simply a model input file replica except that the space occupied by each parameter in the latter file is replaced by a sequence of characters which identify the space that the parameter occupies. The first line of a template file must contain the letters *ptf* (PEST template file) followed by a parameter delimiter explained later. It is possible to see the examples of a PEST input file and its relative PEST template file taken from PEST Manual in **Figure 7** and **Figure 8** respectively.

```

MODEL INPUT FILE
3, 19                               no. of layers, no. of spacings
1.0, 1.0                             resistivity, thickness: layer 1
40.0, 20.0                            resistivity, thickness: layer 2
5.0                                    resistivity: layer 3
1.0                                    electrode spacings
1.47
2.15
3.16
4.64
6.81
10.0
14.9
21.5
31.6
46.4
68.1
100
149
215
316
464
681
1000

```

Figure 7. Example of PEST input file.

```

ptf #
MODEL INPUT FILE
3, 19                               no. of layers, no. of spacings
#res1    #,#t1    #                 resistivity, thickness: layer 1
#res2    #,#t2    #                 resistivity, thickness: layer 2
#res3    #                                     resistivity: layer 3
1.0                                    electrode spacings
1.47
2.15
3.16
4.64
6.81
10.0
14.9
21.5
31.6
46.4
68.1
100
149
215
316
464
681
1000

```

Figure 8. Example of PEST template file.

Both input and template file are necessary to create the PEST control file explained later.

2.1.1 The parameter delimiter

The *parameter delimiter* is the equivalent of the field format of Fortran, indicates the boundaries in which PEST will read the input file of the model. The parameter delimiter is a single character that follows the keyword “*ptf*” after a blank space in the PEST template file (# in **Figure 8**). In a template file, a *parameter space* is identified as the set of characters limited by and including two parameter delimiters. When PEST 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 characters [a-z], [A-Z] and [0-9] are invalid to identify the parameter delimiter. The defined parameter delimiter must appear nowhere within the template file except in its parameter delimiter capacity, in fact wherever PEST finds that character in the template file it assumes that it is delimiting a parameter space.

2.1.2 Parameter names

All parameters are referenced by a name, these references are necessary in template files (in which the parameters locations on model input files are identified), in control file (where parameter initial values and lower and upper bounds are provided), and in input files which contains parameters that PEST has to process. Parameter names cannot exceed twelve characters in length and are case-insensitive, moreover any characters is allowed except for the space character and the parameter delimiter chosen previously.

Given that each parameter space is defined by two parameter delimiters, the parameter name to which the space belongs must be written between the two delimiters.

2.1.3 Setting the Parameter Space Width

Numbers can be represented with greater precision in wider spaces than in narrower spaces, anyway PEST can adjust to limited precision in parameters representation on model input files, as long as sufficient precision is employed in order to distinguish a parameter value from the value of that same parameter incremented for derivatives calculation. It could be useful to provide the values of the parameters with as much precision as the model is able to read them with, in this way they can be provided to the model with the same precision with which they have been calculated by PEST.

Generally the numbers are read by the models either from the terminal or from an input file in two different ways: namely from specified fields, or as a sequence of numbers (of any length); the latter method is often referred to as *free field* input or as *list-directed* input. Notice how no

whitespace or comma is needed between numbers which are read using a field specifier. Very often, a model input file is used as starting point for the construction of a template file. In such input file, numbers are read using free field input and are often be written without trailing zeros, for this reason during the construction of the template file it has to be recognized where there are free field input numbers and consequently expand the parameter space (to the right) in respect to the original number, leaving whitespace or a comma between successive spaces, or between a parameter space and a neighboring character or number.

In a similar way, numbers read through field-specifying format statements may not occupy the full field width in a model input file, in this case, during the template file compilation, it should be necessary, again, to expand the parameter space beyond the extent of the number (here to the left of the number only) until the space coincides with the field defined in the format specifier.

2.1.4 Preparing a Template File

The preparation of a template file is very simple procedure, in many models it is done just by using a text editor, coping the input file and replacing the parameter numerical values with their respective parameter space and name identifiers. As mentioned before, one time that the template file has been created, its correctness can be checked using the TEMPCHECK utility program. This program can also write a model input file which will be based on the template file and a user-supplied list of parameter values. Then, if the model is run with the provided TEMPCHEK-prepared input file, the model will not have difficulties in reading the PEST prepared input files.

2.2 Instruction files

For each output file of the model that contains observations, an instruction file has to be provided to PEST in order to instruct the program to follow its directions which PEST must follow to read that file. This instruction file has to have the extension “.ins”. The first line of a PEST instruction file must begin with the three letters “*pif*” (PEST instruction file).

Remember that if an output file is more than 2000 characters in width, PEST will not read it. Pay attention that given that the precision in the representation of model-generated observations is

essential, unlike the precision of parameter values, it is possible to vary with any control variables the precision with which a model's output data are written in order to have the maximum available precision. Also this file together with input and template files is used to generate the control file.

2.2.1 How PEST Reads a Model Output File

PEST must be taught how to read a model output file (that must be a text file) and identify the numbers it must extract from that file. A list of instructions on how to find data on an output file must be provided to PEST, instead of using a template for an output file.

Essentially, PEST acts in the same way as a person does. Both run eyes down the file looking for something that is known, a *marker* properly selected to link to it the observations. In case of simple models, more in detail single-purpose models where small development time has been invested in highly descriptive output files, no markers are necessary, the default initial marker will be the top of the file. It is possible to have either primary or secondary markers. PEST employs primary marker to scan the model output file line by line, looking for a reference point for subsequent observation identification or further scanning, while it uses a secondary marker as a reference point given that a single line is examined from left to right.

2.2.2 The Marker Delimiter

The role of the marker delimiter in an instruction file is not unlike that of the parameter delimiter in a template file, it must be placed just After the acronym "*pif*" and a single space and before the first character of a text string comprising a marker and immediately after the last character of the marker string and it has to define the extent of a marker. The portion of text between a pair of marker delimiters is not interpreted by PEST as a set of instructions. The choice of the marker delimiter is limited, in fact it cannot be one of the following characters: A - Z, a - z, 0 - 9, !, [,], (,), :, &, the space or tab characters and it should not occur within the text of any markers as this will cause confusion to PEST.

2.2.3 Observation Names

Each observation has to be provided with a unique name that must be twenty characters or less in length. These characters can be any ASCII characters except for [,], (,), or the chosen marker delimiter character. These same observation names must also be cited in the PEST control file where measurement values and weights are provided. These rules do not apply to the dummy observation name, “*dum*”; it can occur many time in an instruction file if necessary and signifies to PEST that although the observation is to be located as if it were a normal observation, the number corresponding to the dummy observation on the model output file is not actually matched with any laboratory or field measurement. Thus, a “*dum*” observation must not appear in a PEST control file where measurement values and observation weights are provided. The dummy observation is simply a device for model output file navigation.

2.2.4 The Instruction file keywords

When an instruction file is created, it has to be provided with a precise and clear syntax which must be followed exactly. The instruction items on a single line must be separated by at least one space. Instructions in a single line of a model output file are written on a single line of the PEST instruction file. So the new instruction line start implies that PEST must read at least another one new model line of the output file, the number of lines that it has to read depends on the first instruction on the new instruction line. If the first instruction on the new line is the character “&”, that like the others instruction items has to be separated from its following instruction item by at least one space, it means that the new instruction line is simply a continuation of the old one. PEST reads the model output file in *top-to-bottom* (forward) direction and it is important to remember that an instruction cannot direct PEST to read a previous line on the output file. Here following are reported the main keywords for the instruction file:

Primary Marker

The primary marker has already been discussed briefly. On encountering a primary marker in an instruction file PEST reads the model output file, line by line, looking for the string between the marker delimiter characters. When it finds the string it places its “cursor” at the last character of the string. If any other instruction as the primary marker on the same instruction line directs PEST to further processing this line, that processing must pertain to parts of the model output file line following the string identified as the primary marker. If in a primary (or secondary) marker there are blank characters, exactly the same number is expected to match the string on the output file of the model. It should be noted that despite the utility of markers, the search for a primary marker is a very time-consuming process as each line of the model output must be individually read and scanned for the marker.

Line Advance

The syntax for the line advance item is “ ln ” where n is the number of lines to advance. The line advance item must be the first item of an instruction line; it and the primary marker are the only two instruction items which can occupy this initial spot. Unlike the case of the primary markers, in implementing a line advance, PEST does not have to examine the model output file lines while it advances. It simply moves forward n lines, placing its processing cursor just before the beginning of this new line, this point will become the new reference point for more processing of the model output file. Generally a line advance command is followed by other instructions. If a line advance item precedes the first instruction line of a PEST instruction file, the reference point for line advance is taken as a “dummy” line just above the first line of the model output file.

Secondary Marker

A secondary marker is a marker which does not occupy the first position of a PEST instruction line. Hence it instructs PEST to move its cursor along the current model output file line until it finds the secondary marker string, and to place its cursor on the last character of that string ready for subsequent processing of that line. If a particular secondary marker is preceded only by other markers, and the text string corresponding to that secondary marker is not found on a model output file line on which the previous markers’ strings have been located, PEST will assume that it has not yet found the correct model output line and resume its search for a line which holds the text from all the markers. When it will find a line with both primary and secondary markers, it

will commence the next instruction line execution. If any instruction items different from the markers precede an unmatched secondary marker, PEST will consider that mismatch an error condition and will stop the execution displaying an appropriate error message. Remember that also the secondary markers may be used sequentially.

Whitespace

The whitespace instruction is similar to the secondary marker in that it allows the user to navigate through a model output file line prior to reading a non-fixed observation. It directs PEST to move its cursor forwards from its current position until it encounters the next blank character. PEST then moves the cursor forward again until it finds a nonblank character, finally placing the cursor on the blank character preceding this nonblank character ready for the next instruction. The whitespace instruction is a simple “w”, separated from its neighboring instructions by at least one blank space.

Tab

The tab instruction places the PEST cursor at a user-specified character position (i.e. column number) on the model output file line which PEST is currently processing. The instruction syntax is “tn” where n is the column number. The column number is obtained by counting character positions (including blank characters) from the left side of any line, starting at 1. Like the whitespace instruction, the tab instruction can be useful in navigating through a model output file line prior to locating and reading a non-fixed observation.

Fixed Observations

First of all, it is necessary to specify that an observation reference can never be the first item on an instruction line. As for the others instructions, if there are different observation on a particular line of the model output file, these must be read from left to right and cannot be read backward along the line. To identify observations it is possible to say to PEST that a particular observation can be found between and also including columns n_1 and n_2 on the output file line of the model where the cursor is resting at that moment. This is the best way to read an observation value because PEST simply has to read a number from the space identified without doing any research. This type of observations is called “*fixed observations*”. The instruction for PEST for reading a

fixed observation consists of two parts. The first part consists of the observation name between square brackets, while the second one indicates the number of the first and last columns from which to read the observation. These instructions must not be separated by a space because PEST will interpret the space as the end of the instruction item.

When the model output is written in a tabular form using fixed field width specifiers, reading the numbers as fixed observations is very useful. However it has to be remembered that the space defined by the column numbers must be wide enough to accommodate the maximum length that the number will occupy over the model runs for which PEST will read the output file. This space must not be so wide to include parts of others numbers, otherwise an error will occur and PEST will stop its execution with an appropriate error message.

If an instruction line contains only fixed observations whitespace or tabs are not necessary and there will not any need for a secondary marker. This because these items are generally used for navigating through a model output file line before reading a non-fixed observation (explained later); in fact this navigation is not required to locate a fixed observation as its location on a model output file line is defined without ambiguity by the column numbers included within the fixed observation instruction.

Semi-Fixed Observations

Semi-fixed observations are very similar to fixed observations in that two numbers are provided in the pertinent instruction item, this numbers purpose being to locate the observation's position by column number on the model output file. Nevertheless, differently from fixed observations, these numbers do not locate precisely the observations; in fact when PEST finds a semi-fixed observation instruction, it continues to the first of the two columns of interest and then it look for the second the second identified column or a non-blank character in that output file line from left to right starting from the previous column number. If PEST finds a non-blank character before reaching the second column, an error condition will arise. By the way, if it finds a non-blank character at the first of the two column numbers, it then locates the nearest whitespace on either side of the character; in this way, it identifies one or a number of non-blank characters situated between whitespace. It tries to read these characters as a number, being the value of the observation named in the instruction of semi-fixed observation. The width of this number can be greater than the difference between the column numbers cited in the instruction. As for fixed observation, the instruction for reading a semi-fixed observation consists of two parts,

specifically observation name followed by two column numbers that must be in ascending order, the latter being separated by a colon. To remember that for semi-fixed observations, the observation name is enclosed in round brackets rather than square brackets and also here there not must be space between the two parts of the instruction.

If, when reading the model output file, PEST encounters only whitespace between (and including) the two nominated column numbers, or if it encounters non-numeric characters or two number fragments separated by whitespace, an error condition will occur and PEST will terminate execution with an appropriate error message.

It should be clear that for PEST is clearly more complicated to read a semi-fixed observation than a fixed one because it must establish for itself the extent of the number that it has to read.

However it should be noted that it takes more effort for PEST to read a semi-fixed observation than it does for it to read a fixed observation as PEST must establish for itself the extent of the number that it must read.

Non-Fixed Observations

Differently from fixed and semi-fixed observations, *non-fixed* observation instruction does not include any column numbers because the number which PEST must read is found using either secondary markers, whitespace or/and tabs which precede the non-fixed observation on the instruction line.

If, on a particular output file line, you do not know exactly where a model will write the observation number, but you do know the structure of that line, then you can use this knowledge to navigate your way to the number. In the instruction file of PEST, a non-fixed observation is represented simply by the observation name surrounded by exclamation marks and no spaces that separate these elements as for the others types of observations.

When PEST finds a non-fixed observation instruction it first searches forward from its current cursor position until it finds a non-blank character; this character is assumed by PEST as the beginning of the non-fixed observation number. Then PEST looks forward again until it finds either a blank character, the end of the line, or the first character of a secondary marker which follows the non-fixed observation instruction; thus PEST assumes that the non-fixed observation number finishes at the previous character position. If it is unable to read a number because of the presence of non-numeric characters or if it encounters the end of a line while looking for the

beginning of a non-fixed observation, PEST will stop the execution with a run-time error message.

2.2.5 Creation of an Instruction File

In this file there are the command to instruct PEST to read the measured observation (pressure and temperature profiles...) and simulated observation (PEST output variables). An instruction file, as the others PEST input files, can be built using a text editor. In building an instruction set to read a model output file it is necessary caution and attention, especially if markers, whitespace, tabs and dummy observations are utilized. PEST will always follow the given instructions to the letter, but it may not read exactly the desired number if an instruction is wrong. If PEST does not find an observation number where it expects one during the reading of the file, a run-time error will occur. To allow the user to find the problem, PEST will display of where it encountered the error, and of the instruction it was implementing when the error occurred.

PEST utility program PESTCHEK during the check of all the input data of PEST before a PEST run, reads all the instruction files cited in a PEST control file ensuring that no syntax errors are present in any of these files. Whereas INSCHEK program, checks a single PEST instruction file for syntax errors. If an instruction file is error-free, INSCHEK can then use that instruction file to read a model output file, printing out a list of observation values read from that file. In this way you can be sure that your instruction set “works” before it is actually used by PEST.

2.3 The PEST control file

In the control file there are the information and the characteristics for the parameter estimation. For shortness not every variable appearing in the PEST control file will be discussed in the following subparagraphs. Some keywords, and some entire sections of the PEST control file, are discussed entirely in the PEST manual Part 1 (Chapter 4) where the aspects of PEST’s functionality to which they pertain are presented in detail [Doherty, 2016].

Once template and instruction files have been prepared for a particular case, a PEST control file must be prepared to keep everything together. Unlike template and instruction files, there are some conventions associated with the PEST control file name. In fact, the file must have the

extension of “.pst”. Its filename base is referred to as the PEST “case name” herein; PEST uses this same filename base for the files which it generates in the course of its run. For example, let *case* represent this case name. The PEST control file is therefore named *case.pst*. As it runs, PEST generates a set of files which all have this same filename base: *case.rec* (the run record file), *case.par* (best parameter values), *case.rst* (contains restart information), *case.jco* (the Jacobian matrix file) and others.

The PEST control file can be built in a number of ways. It can easily be prepared using a text editor following the directions provided in the manual. Alternatively, it is possible to use the PESTGEN utility to generate a PEST control file for the current case which uses default input variables; this file can then be modified using a text editor. In all of these cases the PEST control file can be checked for correctness and consistency using the PESTCHEK utility.

The first line of a PEST control file must contain only the string “pcf” (PEST control file). This text is case-insensitive (as is all other text featured in a PEST control file).

The PEST control file consists of integer, real and character variables separated by spaces. The value of each variable must be separated from its neighbor by at least one space. Real numbers can be supplied with the minimum precision necessary to represent their values; the decimal point does not need to be included if it is redundant. Note, however, that all real numbers are stored internally by PEST as double precision numbers. The exponentiation is expressed using the *e* symbol.

The PEST control file is subdivided into sections, each of which has a section header whose name begins with the “*” character; a space must separate this character from the text which follows it.

A simplified example of a PEST control file is presented in **Figure 9**. Here many optional variables have been omitted.

```

pcf
* control data
restart estimation
5 19 2 2 3
2 3 single point
10.0 -3.0 0.3 0.03 10
3.0 3.0 0.001
0.1
50 0.005 4 4 0.005 4
1 1 1
* parameter groups
ro relative 0.01 0.0001 switch 2.0 parabolic
h relative 0.01 0.0001 switch 2.0 parabolic
* parameter data
ro1 fixed factor 0.5 .1 10 ro 1.0 0.0
ro2 log factor 5.0 .1 10 ro 1.0 0.0
ro3 tied factor 0.5 .1 10 ro 1.0 0.0
h1 none factor 2.0 .05 100 h 1.0 0.0
h2 log factor 5.0 .05 100 h 1.0 0.0
ro3 ro2
* observation groups
obsgrp1 cov.mat
obsgrp2
prgrp1

* observation data
ar1 1.21038 1.0 obsgrp1
ar2 1.51208 1.0 obsgrp1
ar3 2.07204 1.0 obsgrp1
ar4 2.94056 1.0 obsgrp1
ar5 4.15787 1.0 obsgrp1
ar6 5.7762 1.0 obsgrp1
ar7 7.7894 1.0 obsgrp1
ar8 9.99743 1.0 obsgrp1
ar9 11.8307 1.0 obsgrp2
ar10 12.3194 1.0 obsgrp2
ar11 10.6003 1.0 obsgrp2
ar12 7.00419 1.0 obsgrp2
ar13 3.44391 1.0 obsgrp2
ar14 1.58279 1.0 obsgrp2
ar15 1.1038 1.0 obsgrp2
ar16 1.03086 1.0 obsgrp2
ar17 1.01318 1.0 obsgrp2
ar18 1.00593 1.0 obsgrp2
ar19 1.00272 1.0 obsgrp2
* model command line
model.bat
* model input/output
ves1.tpl a_model.in1
ves2.tpl a_model.in2
ves1.ins a_model.ot1
ves2.ins a_model.ot2
ves3.ins a_model.ot3
* prior information
pi1 1.0 * h1 = 1.0 3.0 prgrp1
pi2 1.0 * log(ro2) + 1.0 * log(h2) = 2.6026 2.0 prgrp1

```

Figure 9. Example of *Control file*, extracted from PEST User Manual Part I, 2016.

2.3.1 Parameter Groups Section

Every model parameter must belong to a parameter group; the group to which each parameter belongs is supplied through the parameter-specific PARGP keyword supplied in the “parameter data” section of the PEST control file. Each parameter group must possess a unique name of twelve characters or less in length.

A tied or fixed parameter can be a member of a group; however, as derivatives are not calculated with respect to such parameters, the group to which these parameters belong is of no significance. Hence fixed or tied parameters can be assigned to the dummy group “none”. If a parameter is

assigned to any group other than “none” in the “parameter data” section of the PEST control file, the properties for that group must be defined in the “parameter groups” section of the PEST control file.

In the following part of the paragraph, there will be a description of only one keyword for the parameter group that is PARGPNME because of its utility in the development of the elaborate. The others several keywords, named INCTYP and DERINC, DERINCLB, FORCEN, DERINCMUL,DERMTHD and SPLITHRESH, SPLITRELDIFF and SPLITACTION are omitted here, but the complete description is present in the already cited PEST User Manual Part I, 6th edition published in 2016 by Doherty.

The PARGPNME indicates the parameter group name. This must be a maximum of twelve characters in length, but it is preferable to limit to six characters that length.

If a group is featured in the “parameter groups” section of a PEST control file it is not essential that any parameters belong to that group. However if, in the “parameter data” section of the PEST control file, a parameter is declared as belonging to a group that is not featured in the “parameter groups” section of the PEST control file, an error condition will arise.

The parameter group name “none” is illegal. This name is only reserved for fixed and tied parameters. However assignment of such parameters to the “none” group is actually not recommended as you may want to untie or unfix them in the future. For this reason it is better to assign them to a parameter group, even if the group-specific variables which govern derivatives calculation have no meaning for these parameters [Doherty, 2016].

2.3.2 Parameter Data Section

For every parameter cited in a PEST template file, up to ten fields of instruction must be provided in the PEST control file. On the contrary, every parameter in the PEST control file must be cited at least once in a PEST template file.

The “parameter data” section of the PEST control file is divided into two parts; in the first part a line must appear for each parameter. In the second part, a little extra data is supplied for tied parameters (namely the name of the parameter to which each such tied parameter is linked). If there are no tied parameters, the second part of the “parameter data” section of the PEST control file is omitted.

Each item of parameter data is now shortly discussed, for more details see Paragraph 4.9 of PEST manual v.1.

- The *PARNME* is the parameter name. Each parameter name has to be unique and of maximum 12 characters in length; it is also case insensitive.

- The *PARTRANS* is a keyword which must assume one of four values, that can be “none”, “log”, “fixed” or “tied”.

If a parameter is fixed, taking no part in the inversion process, *PARTRANS* must be supplied as “fixed”. If a parameter is linked to another parameter, *PARTRANS* value will be “tied”. In the latter case the parameter plays only a limited role in the inversion process. However the parameter to which the tied parameter is linked (this “parent” parameter must be neither fixed nor tied itself) takes an active part in the inversion process. In fact the tied parameter is simply pulled by the parent parameter, the value of the tied parameter will maintain at all times the same ratio to the parent parameter as the ratio of their initial values.

If a parameter is neither fixed nor tied, the parameter transformation variable *PARTRANS* must be supplied as “none”.

- The *PARCHGLIM* is used to designate whether an adjustable parameter is relative-limited, factor-limited or absolute-limited. This parameter variable can be provided with a value of “relative” or “factor” to designate that a parameter is subject to a relative limit or a factor limit respectively. Alternatively, the string “absolute(*N*)” can replace “relative” or “factor” as the value of the *PARCHGLIM* keyword, where *N* is a number between 1 and 10. If a parameter is tied or fixed, its change limit is ignored.

- The *PARVAL1* is a parameter’s initial value represented by a real variable. For a fixed parameter, this value remains invariant during the inversion process. For a tied parameter, the ratio of *PARVAL1* to the parent parameter’s *PARVAL1* sets the ratio between these two parameters that is maintained throughout the inversion process.

Ideally the initial value of a parameter should be the pre-calibration estimate of the parameter’s value based on expert knowledge alone. If a parameter has an initial value of zero, the parameter can be neither a tied nor a parent parameter as the tied-parent parameter ratio cannot be calculated.

- The *PARLBND* and *PARUBND* are two real variables representing respectively the lower and the upper bound of a parameter. For adjustable parameters the initial parameter value

(*PARVAL1*) must lie between these two bounds. However for fixed and tied parameters the values provided for *PARLBND* and *PARUBND* are ignored.

- The *PARGP* is the group name to which a parameter belongs. As derivatives are not calculated with respect to fixed and tied parameters, PEST provides a dummy group name of “none” to which such tied and fixed parameters can be assigned. Any group other than “none” which is cited in the “parameter data” section of the PEST control file must be properly defined in the “parameter groups” section of this file. It has to be remembered that for reasons discussed in Paragraph 2.3.1, the parameter group “none” is not recommended practice for fixed and tied parameters.

- The *SCALE* and *OFFSET* keywords allow the user to redefine the domain of the parameter. In fact just before that the value of the parameter is written to a model input file, it is multiplied by the real variable *SCALE* and after that the other real variable *OFFSET* is added. Since they intervene on the parameter value at the last minute before it is written to the model input file, they do not take part in the inversion process.

- The *DERCOM* variable should be set to 1 unless PEST’s external derivatives functionalities are used.

3. ADDITIONAL USED TOOLS

In this chapter, there will be an explanation for each program useful and utilized for the following coupling between T2Well and PEST software (Chapter 4).

More specifically, it is possible to find some basic information about G95 compiler, the Fortran language and the program CodeBlocks.

3.1 G95 compiler and Fortran language

The G95 Fortran 95 is a free multi-operating systems compiler for Fortran language. An input file is compiled according to its extension and this is determined by the G95 compiler. For Fortran files, it is possible to have the following extensions: .f, .F, .for, .FOR, .f90, .F90, .f95, .F95, .f03 and .F03. These extensions determine whether the Fortran sources are to be considered as fixed or free format. .f, .F, .for, and .FOR ending files are assumed to be fixed form, while .f90, .F90, .f95, .F95, .f03 and .F03 extension are assumed to be treated as free source format.

The basic options for compiling Fortran sources with G95 compiler are:

-c : does not run the linker, compile only, for example, compiling in the command prompt:

```
g95 -c hello.f90: Compiles hello.f90 to an object file named hello.o;
```

-v : shows the actual programs invoked by G95 and their arguments, it is useful for tracking path problems;

-o : specifies the output file name, that could be either the executable or an object file. In the Windows systems, an .exe extension is automatically generated, if no output file name is specified, a.exe is the default name in Windows systems. Writing in the command prompt:

```
g95 -o hello h1.f90 h2.f90 h3.f90: the compiler creates multiple source files and links them together to an executable file named hello.exe.
```

Now, as regard the Fortran language, used to write the source example model, some basic definitions are given together with the explanation of the basic and the most frequently utilized keywords.

A *Fortran program* is formed by one or more program units. A *program unit* is generally a sequence of statements that define the data environment and the necessary steps for the calculations; it has to terminate by an END statement (statements concepts will be explained

later). A program unit can be either a main program, an external subprogram, a module, or a block data program unit. An executable program includes always one main program, and, optionally, other kinds of program units that can be separately compiled.

As regard the *program statements*, they are divided into two general classes: executable and non-executable. An *executable statement* specifies an action to be performed, while a *non-executable statement* describes program attributes, (arrangement and characteristics of data, editing and data-conversion information,...). A Fortran statement cannot start with a digit.

The **Figure 10** reported below, shows the statements order that is required in a unit of a Fortran program. The vertical lines in the scheme separate statement types that can be alternated. The statements types that are divided by horizontal lines cannot be interspersed.

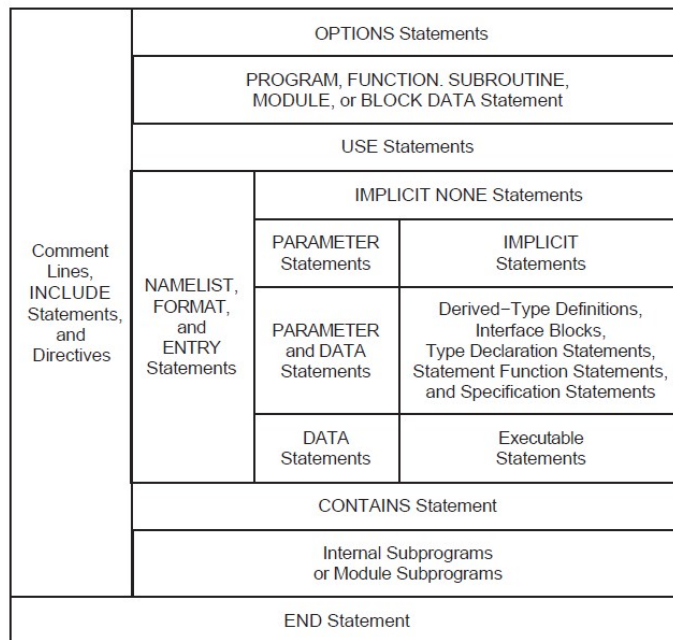


Figure 10. Statements order that is required in a unit of a Fortran program [Intel® 2003-2004].

The PROGRAM statement is optional. The END statement is the only required part of a program. In a program, source code could be in *free, fixed, or tab form*. Fixed or tab forms must not be in the same source program with free form.

- In *free source form*, statements are not limited to specific positions on a source line. A source line of a free form can contain from 0 to 132 characters and blank characters are significant, exclamation point character (!) indicates a comment, moreover the ampersand character (&) indicates a continuation line, (up to 39 continuation lines are permitted in this programs...).

- In *fixed and tab* source forms, a statement can appear within a line with restrictions. By default, a statement can occupy up to 72 characters, any text following position 72 is ignored and no warning message is printed. Also in this form, the exclamation point character (!), together with the letter C (or c) or an asterisk (*) indicate a comment. For the comment character in these forms, it is possible to choose between several characters (except 0 and blank) and up to 19 continuation lines are permitted using Fortran 95/90.

This statement marks the end of a program unit. In a program unit the END statement cannot be continued, and no other statement in this unit can have an initial line that seems to be the END statement program unit.

- The PARAMETER attribute defines a named constant and can be specified in a type declaration statement or a PARAMETER statement. *Real data types* can be specified as follows:

REAL

REAL([KIND=]*n*)

REAL**n*

DOUBLE PRECISION

Where *n* is kind 4, 8, or 16. If a kind parameter is specified, the real constant has the kind specified, while if a kind parameter is not specified, the kind is *default real*: REAL(4). DOUBLE PRECISION is REAL(8) and no kind parameter is permitted for data declared with this precision type.

Integer data types can be written as follow:

INTEGER

INTEGER([KIND=]*n*)

INTEGER**n*

Where *n* is kind 1, 2, 4, or 8. As in the real data type, if there is a specified kind parameter, the integer has the kind specified, while if a kind parameter is not specified, integer constants can be interpreted as reported: if the integer constant is within the default integer kind range, the kind is *default integer*: INTEGER(4); whereas if the integer constant is outside the default integer kind range, the kind of the integer constant is the smallest integer kind which holds the constant.

An *integer constant* is a whole number with no decimal point, which can have a leading sign and can be interpreted as a decimal number. Integers are expressed in decimal values (base 10) by default.

- The *DO construct* controls the repeated execution (loop) of a statements/constructs block. The number of iterations of a loop can be specified in the initial DO statement in the construct, or the number of iterations can be left indefinite by a simple DO ("DO forever") construct or DO WHILE statement. The EXIT and CYCLE statements modify the loop execution. The former statement terminates execution of a loop, while the latter terminates the execution of the loop current iteration. If an error or end-of-file occurs, the DO construct terminates. The range of a DO construct includes all the statements and constructs that follow the DO statement, up to and including the terminal statement. If DO construct contains another construct, the inner construct must be entirely contained in the DO construct.
- The *CONTINUE statement* is mainly used to terminate a DO construct when the construct would otherwise end improperly with either a GO TO, arithmetic IF, or other prohibited control statement. This simple statement does nothing by itself and do not have effect on execution sequence of program results.
- The *IF construct* conditionally executes one block of statements or constructs, while the *IF statement* conditionally executes one statement. The decision to transfer control or to execute the statement or block is based on a logical expression evaluation within the IF statement/construct. As a rule, if a construct name is specified at the beginning of an IF THEN statement, this same name must appear in the corresponding END IF statement. The logical expressions are evaluated in the order in which they appear, until an ELSE or END IF statement is encountered or a true value is found. Once one of the above conditions are encountered, the block immediately following it is executed and the construct execution terminates.
- The *READ statement* is a data transfer input statement. Data can be input either from external sequential, direct-access records, or from internal records. The input data taken from the decided type of record is transferred by the sequential READ statements.
- The *OPEN statement* connects a Fortran logical unit to a file or device and declares attributes for read and write operations, more easily the OPEN statement connects an external file to a unit, creating a new file and connecting it to a unit.
- The *CLOSE Statement* terminates the connection between a logical unit and a file or device, it can also disconnect a file from a unit. The CLOSE statement specifiers can appear in any order, the status specified in this statement supersedes the status specified in the OPEN statement. If this statement is specified for a unit that is not open, it has no effect.

- The *WRITE* statement is a data transfer output statement. Data can be output to external sequential or direct-access records, or to internal records. The statements can be formatted by using format specifiers or namelist specifiers, or they can be unformatted [Intel® 2003-2004].

3.2 CodeBlocks

To improve the writing of a Fortran file source, it was used CodeBlocks, a *free C, C++ and Fortran IDE* (Integrated Developed Environment) designed to be very extensible and fully configurable. In fact it was built around a plugin framework in order to be extended with plugins to add functionalities.

For example the function compiling and debugging is already provided by plugins.

In my work, CodeBlocks has been used to write the source and to build the executable of the program-model used.

The CodeBlocks user interface is visible below in the **Figure 11**.

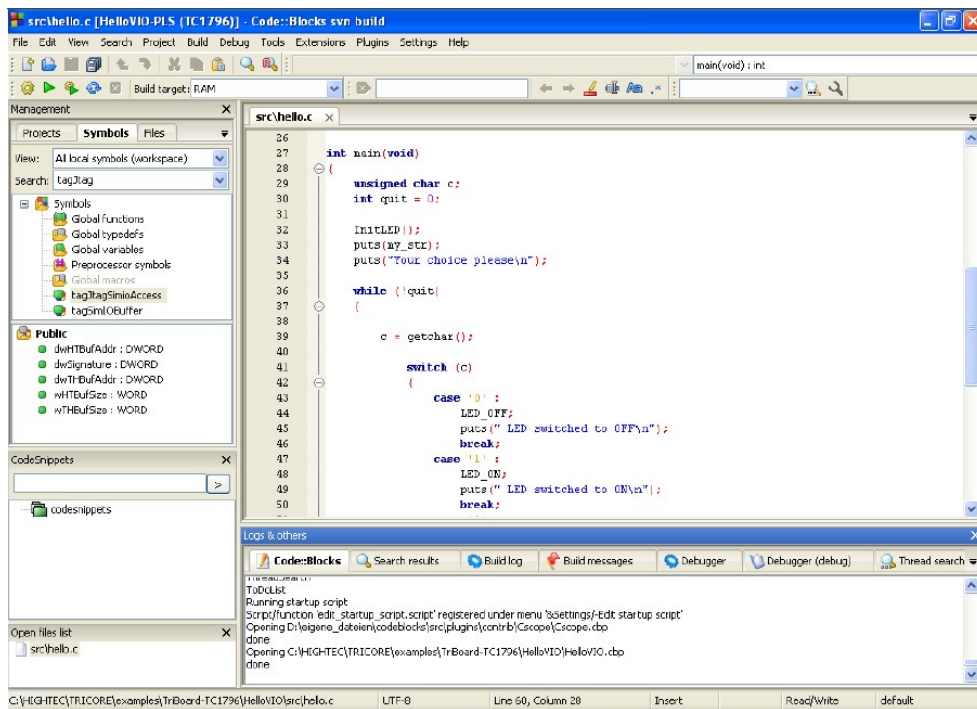


Figure 11. The CodeBlocks user interface.

Now, referring to the CodeBlocks manual v. 1.1 [Björklund et al.], it is possible to describe the components of the interface:

- **Management:** this window contains the interface *Projects* which will be referred to as the project view. To specify, a project can be defined as the set of command of IDE to create starting from the source files the executable file. Here all the project opened in CodeBlocks at a certain time are shown. *Symbols tab* in the Management window shows all variables, symbols, etc...;
- **Editor:** In **Figure 11** as an example, a source named *hello.c* is opened with syntax highlighting in the editor.
- **Open files list:** a list of all files opened in the editor is shown, in this example: *hello.c*.
- **CodeSnippets:** here it is possible to manage text modules, links to urls and links to files. It is possible to display this window via menu *View → CodeSnippets*;
- **Logs & others:** here search results, log messages of a compilers, bugs, etc... are output.

CodeBlocks offers a very flexible and comprehensive project management, for more details about projects, template, scripts, etc... refer to the CodeBlocks v. 1.1 mentioned before.

4. APPLICATION OF PEST

In this chapter, will be showed some examples of the different acquired skills regarding PEST using the example present in PEST User Manual Part I as step stone. In particular, the original PEST example found in the manual has been modified in order to explore four different exercises and to apply the acquired knowledge on more complex models, input and output files. The objective of these modification is to approach the typical structure which will be found in the T2Well files.

It will follow the description of the procedure of applying the acquired skills regarding PEST in order to calibrate a numerical model on a real geothermal reservoir, previously already calibrated manually with the trial-and-error method in the work of PhD thesis. The model simulates a coupled wellbore-reservoir flow in geothermal systems using T2Well-EWASG as numerical simulator. The aim is to make the simulation obtained using a specific model more objective thanks to its automatic calibration using PEST. There will be first the description of the model and of the results obtained by trial-and-error calibration, then the calibration procedure with PEST software will be explained step by step (input files preparation, outputs, results analysis).

4.1 PEST example of a bilinear model

In this paragraph, it is reported the reproduction of an example extracted from the Chapter 18 of PEST User manual part I, 6th Edition, [Doherty, 2016]. This example demonstrates step by step the application of PEST to a practical problem in order to understand the procedure and the functions of the software. Once PEST package has been downloaded on the computer, the files that will be cited in this paragraphs can be found in a subfolder of the main PEST folder called *pestex*.

The laboratory data of the example consist in the results of an experiment in which the specific volume of a soil clod is measured at different water contents as the clod is desiccated through oven heating. The experimental data are shown in **Table 1**. These data are visible also in the text file *soilvol.dat*. It is possible to fit two straight lines to these data as displayed in **Figure 12**. In soil physics, the lower slope straight segment fitted through the points of low water content is

referred as the “residual shrinkage” segment, while the other segment with the slope near unity is considered the “normal shrinkage” segment.

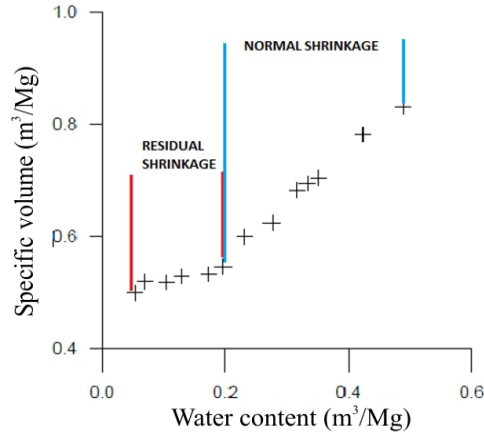


Figure 12. Data of file *soilvol.dat* fitted in two straight lines: the “residual shrinkage” segment and the “normal shrinkage” segment.

Table 1. Experimental data table.

Water content (m³/Mg)	Specific volume (m³/Mg)
0.052	0.501
0.068	0.521
0.103	0.52
0.128	0.531
0.172	0.534
0.195	0.548
0.23	0.601
0.275	0.626
0.315	0.684
0.332	0.696
0.35	0.706
0.423	0.783
0.488	0.832

Now, it is necessary to construct a model, that fits the experimental observations with the analytical ones. Looking at **Figure 13** it is possible to see two intersecting line segments. The slope segment is named s_1 for the first segment and s_2 for the second one. Moreover, the intercept of the first segment on the y-axis is y_1 and the x-coordinate of the point of intersection of the two line segments is x_c . Thus, it is possible to write the system of equations for these two lines in the following way:

$$\begin{aligned}
 y &= s_1 \cdot x + y_1 & x &\leq x_c \\
 y &= s_2 \cdot x + (s_1 - s_2) \cdot x_c + y_1 & x &> x_c
 \end{aligned}$$

In which x is the water content and y the soil clod specific volume.

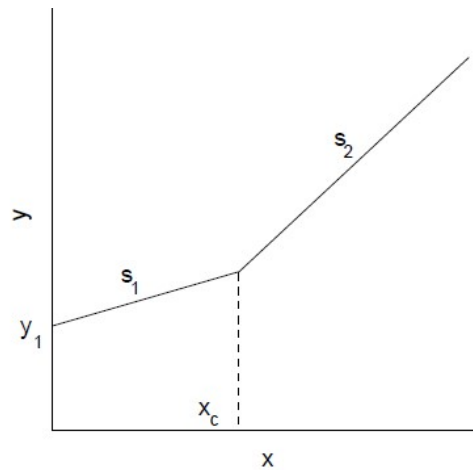


Figure 13. Two line model parameters scheme.

Based on this concept, I have written a simple Fortran program and build it with CodeBlocks to have an executable version of it (*twoline.exe*). The source program has been called *twoline.for* and is visible entirely in **Figure 14** below.

```

0 ..... 10 ..... 20 ..... 30 ..... 40 ..... 50 ..... 60 ..... 70 ..... 80
1  program twoline
2
3  integer*4 i,nx
4  real*4 s1,s2,y1,xc
5  real*4 x(50),y(50)
6
7  write(6,10)
8 10 format(/,' Program TWOLINE. Watermark Computing')
9  write(6,20)
10 20 format(/,' Reading model input file IN.DAT....')
11
12  open(unit=20,file='in.dat')
13
14  c read the line parameters
15
16  read(20,*) s1,s2
17  read(20,*) y1
18  read(20,*) xc
19
20  c read the abscissae at which there are measurement values
21
22  read(20,*) nx
23  do 100 i=1,nx
24  read(20,*) x(i)
25 100 continue
26  close(unit=20)
27  write(6,110)
28 110 format(' File IN.DAT read ok.')
```

```

29
30 c evaluate y for each x
31
32 do 200 i=1,nx
33 if(x(i).le.xc) then
34 y(i)=s1*x(i)+y1
35 else
36 y(i)=s2*x(i)+(s1-s2)*xc+y1
37 end if
38 200 continue
39
40 c write the y values to the output file
41
42
43 write(6,220)
44 220 format(/,' Writing model output file OUT.DAT...')
45 open(unit=20,file='out.dat')
46 do 300 i=1,nx
47 write(20,*) x(i),y(i)
48 300 continue
49 close(unit=20)
50 write(6,320)
51 320 format(' File OUT.DAT written ok.')
```

Figure 14. The source code of the example model.

twoline.for starts reading an input file called *in.dat* which supplies it with values for s_1 , s_2 , y_1 , x_c and the water contents (x values in equations) at which soil clod specific volumes are required and at the end writes a single output file (*out.dat*) listing both water contents and the specific volumes calculated for these water contents.

The input file *in.dat* is shown in **Figure 15** and the output file *out.dat* is visible in **Figure 16** below.

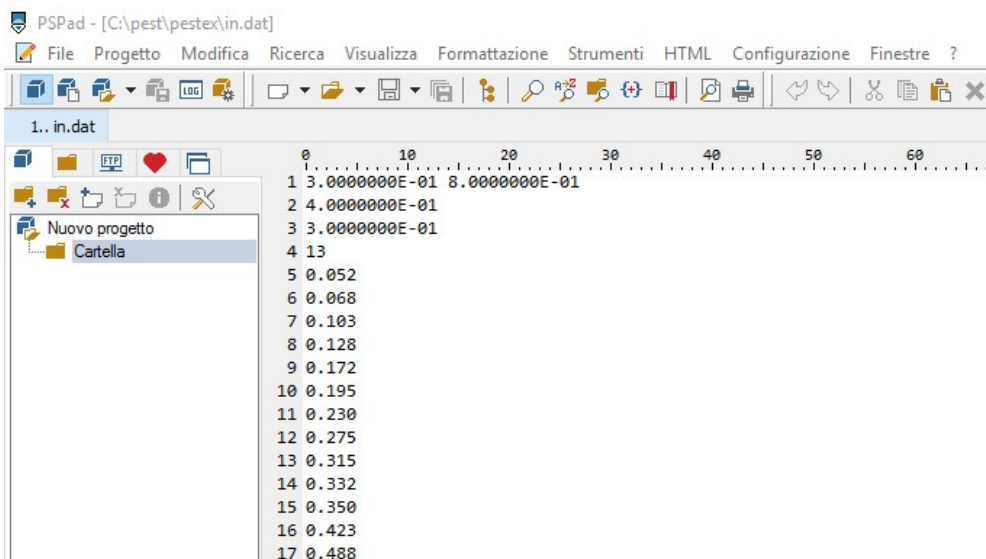


Figure 15. *in.dat* input file.

	0	10	20	30	40	50	60
1	0.520000E-01	0.415600					
2	0.680000E-01	0.420400					
3	0.103000	0.430900					
4	0.128000	0.438400					
5	0.172000	0.451600					
6	0.195000	0.458500					
7	0.230000	0.469000					
8	0.275000	0.482500					
9	0.315000	0.502000					
10	0.332000	0.515600					
11	0.350000	0.530000					
12	0.423000	0.588400					
13	0.488000	0.640400					

Figure 16. *out.dat* output file.

The model *twoline* starting from its input file calculates the specific volumes of the soil clod sample at water contents corresponding to our experimental dataset. Using PEST it possible to adjust the model parameter in order to reduce the discrepancies between laboratory and model-generated specific volumes as small as possible. In this case, the parameters are the four line parameters, namely s_1 , s_2 , y_1 and x_c . Once the model is complete, the next task is to prepare the *twoline*-PEST files.

The template file can be realized simply by copying the file *in.dat* in a file with extension *.tpl* (called for example *in.tpl*) and modifying the latter by replacing the numeric value with the appropriate parameter names. Moreover, the header line “ptf” has been added to the top of the file. The template file *in.tpl* is shown in **Figure 17**.

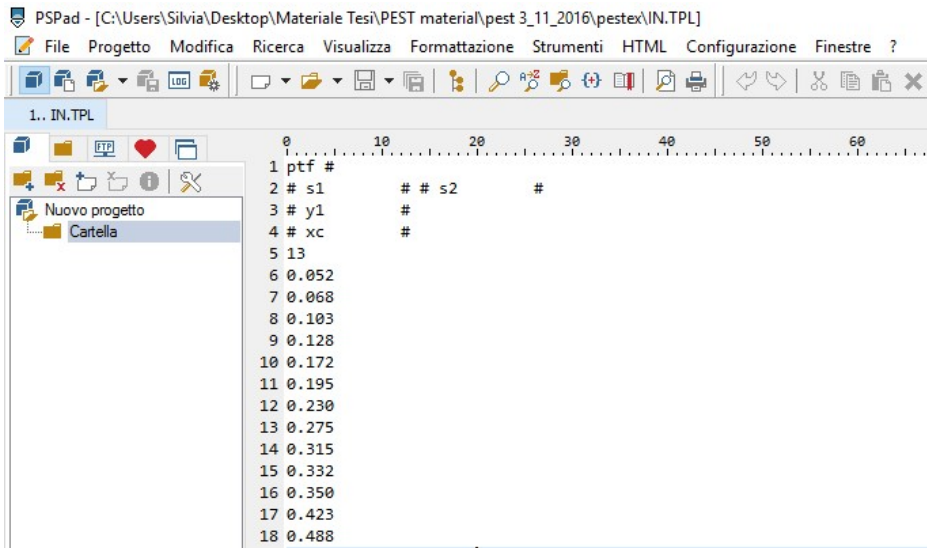


Figure 17. Template file *in.tpl*.

twoline reads all the parameters using a free field format, so each parameter space width is not critical; however, if two parameters are found on the same line, they must be separated by a space. In this case the width of the parameters space is 13 characters in order to have the maximum precision available for representing single precision numbers. Once *in.tpl* is ready, it should be checked running the program *TEMPCHEK* writing in the command prompt:

```
tempchek in.tpl
```

TEMPCHEK writes a file in which all the parameters cited in file *in.tpl* are listed. This file will be called *in.pmt* visible in **Figure 18**.

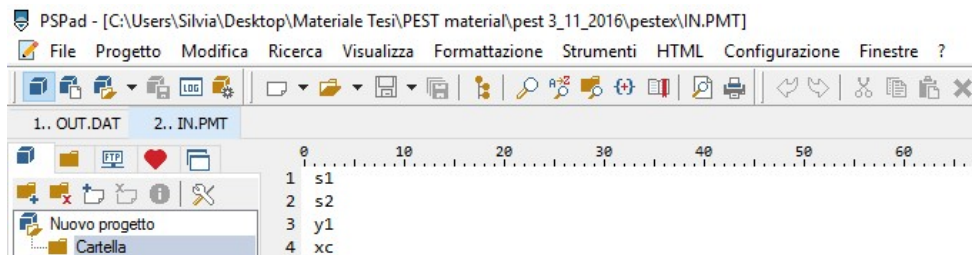


Figure 18. *in.pmt* file.

Now, by copying the file *in.pmt* to *in.par* and adding PEST parameter values, SCALES and OFFSETs to the listed parameter names, as well as values for the character variables PRECIS and DPOINT, it is possible to create a PEST parameter value file. The **Figure 19** shows this file.

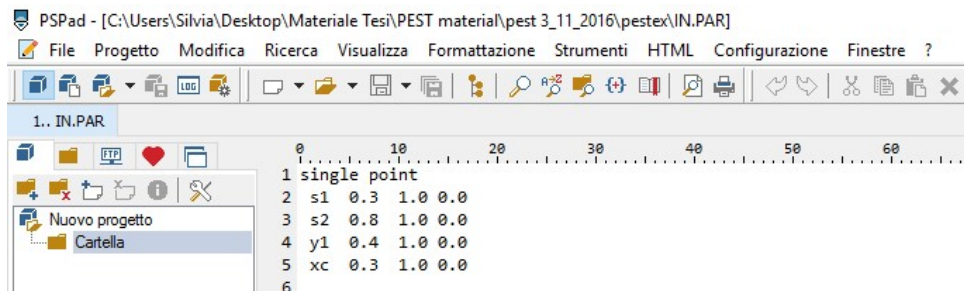


Figure 19. File *in.par* in which the values of SCALES are all equal to 1.0, OFFSETs equal to 0.0 and PRECIS and DPOINT are ‘single point.’

Given that *in.par* will be used shortly with the PESTGEN program to generate a PEST control file, each parameter value supplied has to be the same as the initial parameter value to be used in the process of parameter estimation. At this point, TEMPCHEK should be run again using the command:

```
tempchek in.tpl in.dat in.par
```

When run using this command, TEMPCHEK generates file *in.dat*, the *twoline* input file, using the parameter values provided in file *in.par*. *twoline* should then be run again, making sure that it is read correctly.

Next the instruction file should be prepared. This can be easily accomplished using a text editor by writing the instructions to a file named *out.ins* shown below in **Figure 20**.

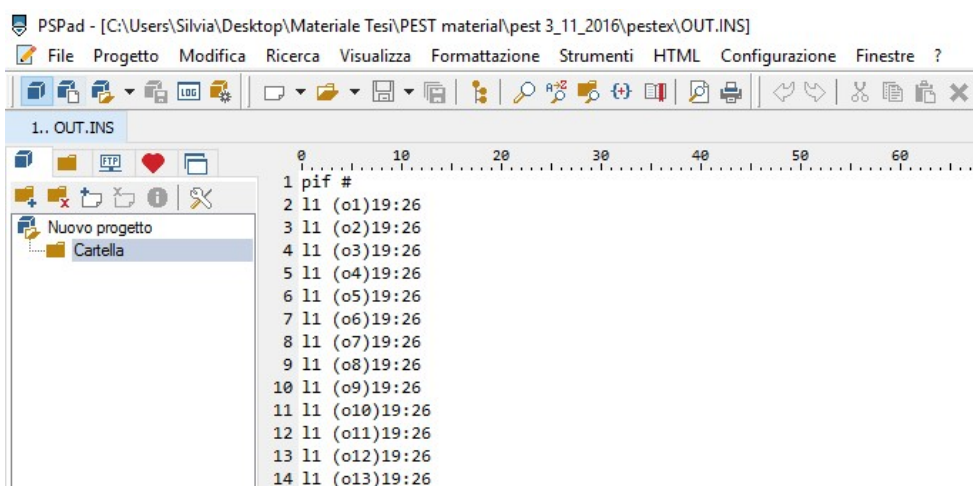


Figure 20. Instruction file *out.ins*.

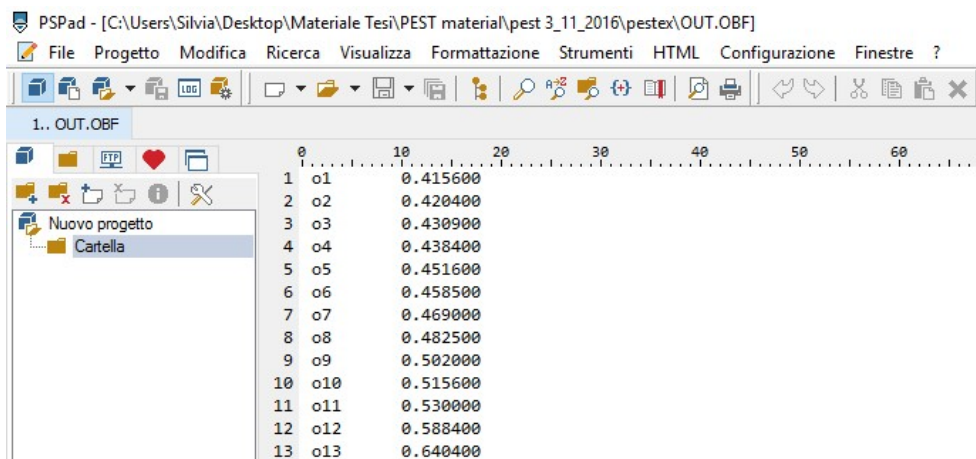
Using this instruction set, all model-generated observations are read as semi-fixed observations; while they could have also been read as fixed observations but it is not known how wide a number can be in the second column of file *out.dat*. The program INSCHEK should be used to check that file *out.ins* contains a legal instruction set and this is done using the command:

```
inschek out.ins
```

If there are no errors, INSCHEK should be run again, this time directing it to read a *twoline* output file using the instruction set and writing in the command prompt:

```
inschek out.ins out.dat
```

A file named *out.obf* will be produced by INSCHEK. In this files the values that it reads from the file *out.dat* for the observations cited in file *out.ins* are listed. It is possible to view the file *out.obf* in **Figure 21**.



	0	10	20	30	40	50	60
1	o1	0.415600					
2	o2	0.420400					
3	o3	0.430900					
4	o4	0.438400					
5	o5	0.451600					
6	o6	0.458500					
7	o7	0.469000					
8	o8	0.482500					
9	o9	0.502000					
10	o10	0.515600					
11	o11	0.530000					
12	o12	0.588400					
13	o13	0.640400					

Figure 21. File *out.obf*.

The PEST- *twoline* interface is now complete as PEST can now correctly act on the *twoline* input file and read its output file. Now a PEST control file must be generated. This file must provide to PEST an appropriate set of control variables and the values measured in laboratory of the specific volume. First of all, the file *out.obf* has to be copied to file *measure.obf* then the values of model-generated observations have to be replaced with the corresponding values from the **Table 1** and the form of the file *measure.obf* is as in **Figure 22** here below.

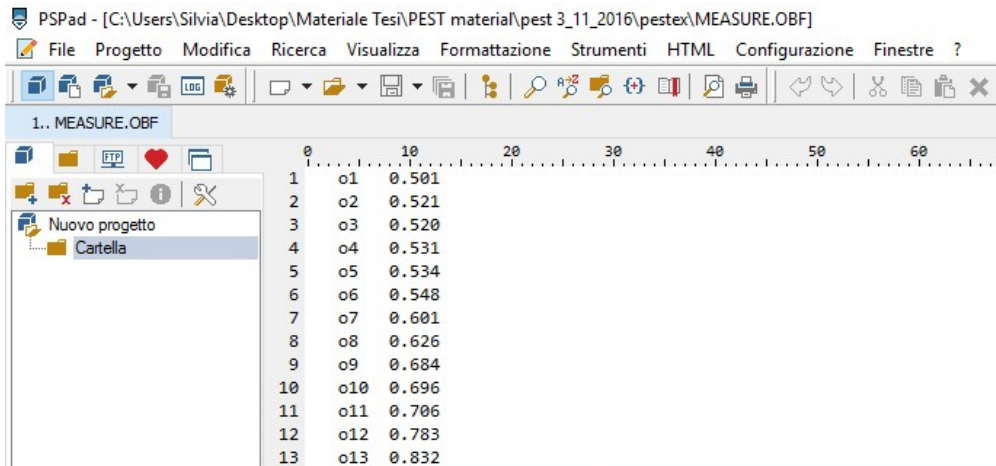


Figure 22. File *measure.obf*.

Then run PESTGEN using the command:

```
pestgen twofit in.par measure.obf
```

With this command PESTGEN generates a PEST control file named *twofit.pst*; see Figure 23.

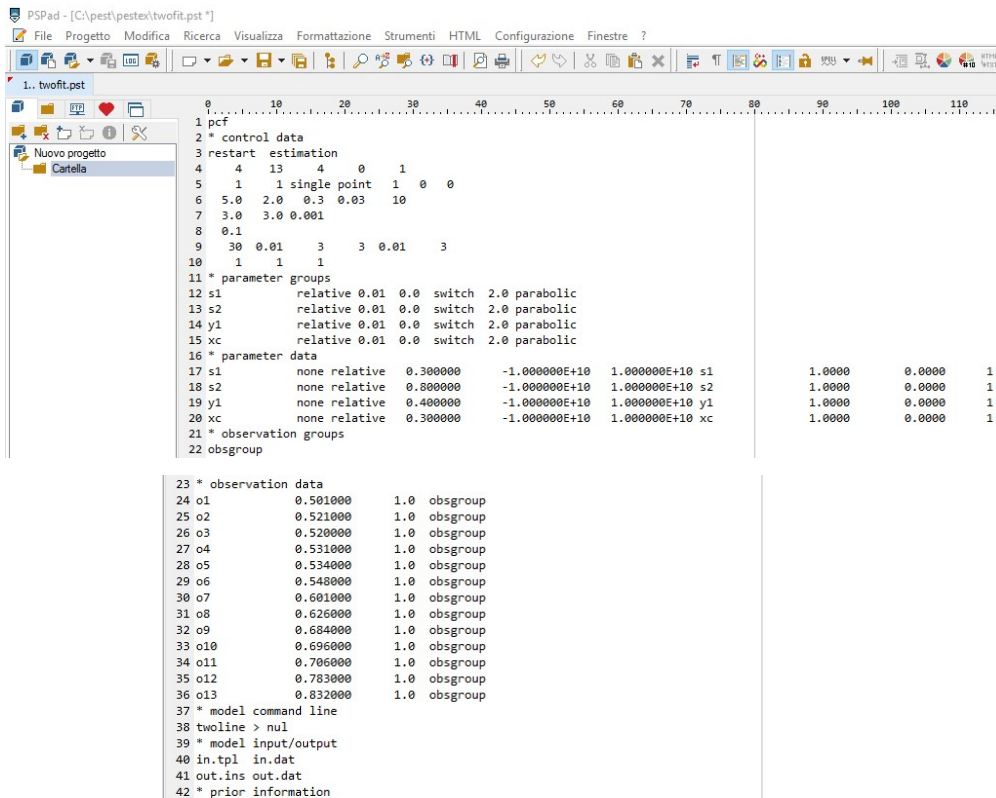


Figure 23. Control file *twofit.pst*.

File *twofit.pst* should now be edited because some of the default values used by PESTGEN in writing this file are not appropriate to the problem. In particular, it is necessary rename the default name of the model (from *model* to *twoline*); the filenames listed in the *model input/output* section of *twofit.pst* need to be modified as well. The **Figure 24** lists that part of *twofit.pst* to which the correction have been made.

```
37 * model command line
38 twoline > nul
39 * model input/output
40 in.tpl in.dat
41 out.ins out.dat
42 * prior information
43 []
```

Figure 24. Modified section of *twofit.pst* file.

Once these changes have been made, the preparation for the PEST run is complete. As a final check for completeness, correctness and consistency of the entire PEST input dataset, program PESTCHEK should be run using the command:

```
pestchek twofit
```

If all is correct, finally PEST can be run using the command:

```
pest twofit
```

In the *pestex* subfolder PEST writes a run record file, *twofit.rec*, and file, *twofit.par*, containing the estimated parameter set. In **Figure 25** the lines of best fit superimposed on the laboratory data are shown.

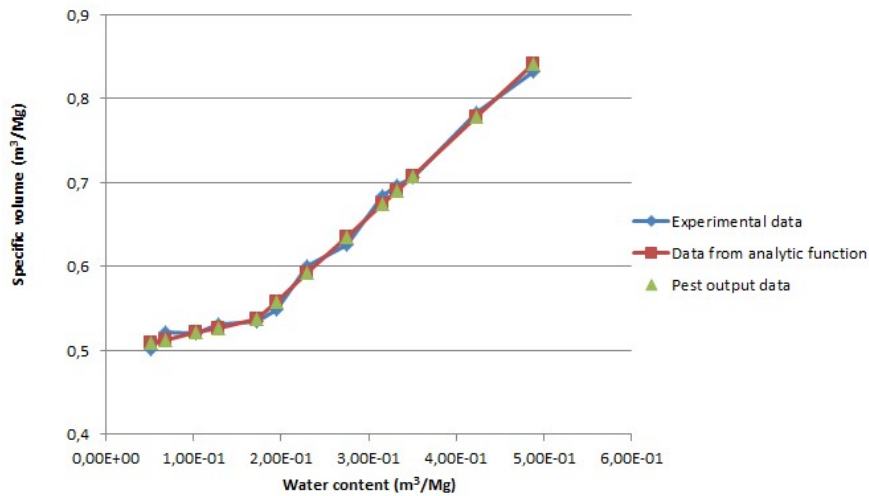


Figure 25. Comparison of experiment results calculated in different ways.

4.1.1 Calibration of fixed parameter

In general, it is useful to have the possibility to instruct PEST to work with model in which only a subset of the model parameters must be calibrated.

Therefore, the example of paragraph 4.1 has been modified in order to fix a parameter. In this case files *in.dat*, *in.tpl*, *in.pmt*, *in.par*, *out.dat*, *out.ins*, *out.obf* and *measure.obf* have been unchanged from the starting example. The only file that has been modified is the *twofit.pst*. In particular, it has been changed only the PARTRANS of the parameter s_1 from “none” to “fixed” as shown in **Figure 26**. With this change, PEST will run to estimate all the others parameters as s_2 , γ_1 and x_c except s_1 . After having saved the new version of file *twofit.pst* and checked it correctness with PESTCHEK as in the example, PEST has been run.

```

1 pcf
2 * control data
3 restart estimation
4 4 13 4 0 1
5 1 1 single point 1 0 0
6 5.0 2.0 0.3 0.03 10
7 3.0 3.0 0.001
8 0.1
9 30 0.01 3 3 0.01 3
10 1 1 1
11 * parameter groups
12 s1 relative 0.01 0.0 switch 2.0 parabolic
13 s2 relative 0.01 0.0 switch 2.0 parabolic
14 y1 relative 0.01 0.0 switch 2.0 parabolic
15 xc relative 0.01 0.0 switch 2.0 parabolic
16 * parameter data
17 s1 fixed relative 0.300000 -1.000000E+10 1.000000E+10 s1 1.0000 0.0000 1
18 s2 none relative 0.800000 -1.000000E+10 1.000000E+10 s2 1.0000 0.0000 1
19 y1 none relative 0.400000 -1.000000E+10 1.000000E+10 y1 1.0000 0.0000 1
20 xc none relative 0.300000 -1.000000E+10 1.000000E+10 xc 1.0000 0.0000 1
21 * observation groups
22 obsgroup
23 * observation data
24 o1 0.501000 1.0 obsgroup
25 o2 0.521000 1.0 obsgroup
26 o3 0.520000 1.0 obsgroup
27 o4 0.531000 1.0 obsgroup
28 o5 0.534000 1.0 obsgroup
29 o6 0.548000 1.0 obsgroup
30 o7 0.601000 1.0 obsgroup
31 o8 0.626000 1.0 obsgroup
32 o9 0.684000 1.0 obsgroup
33 o10 0.696000 1.0 obsgroup
34 o11 0.706000 1.0 obsgroup
35 o12 0.783000 1.0 obsgroup
36 o13 0.832000 1.0 obsgroup

```

Figure 26. Modified control file *twofit.pst*. In the red circle there is the PARTRANS modified of parameter s_1 .

At the end of the estimation, it is possible to find in the work folder, called “s1_fisso”, the file *twofit.par* containing the new estimated parameter set reported in Figure 27.

```

1 single point
2 s1 0.3000000000000000 1.000000 0.000000
3 s2 0.9627923500000000 1.000000 0.000000
4 y1 0.4900190400000000 1.000000 0.000000
5 xc 0.1801308800000000 1.000000 0.000000
6

```

Figure 27. *twofit.par* changed after the modification of the control file.

As it is possible to notice from Figure 27, the applied change worked properly, in fact the parameter s_1 remained fixed to its initial value and the other three parameters changed according to the calibration run.

4.1.2 Modification of the example model

In order continue the acquisition of new skills on the use of PEST, the file *twoline.for* has been modified, with the program CodeBlocks, adding two more parameters.

In particular, the example of paragraph 4.1 has been modified in order to use PEST with a more complex output file respect to the original one using the skills acquired before in Chapter 2.

In particular, the command *write* and *format* of the code have been changed as if PEST read the numbers in fixed field of 15 characters instead of variable field. Moreover the indentation of the code has been shifted to the right of 6 spaces just as the Fortran language requires. In **Figure 28**, it is possible to see only the extract of the code where the change to the field has been applied. In particular, referring to line 61 and line 64 to see the modified code lines.

```

41
42 c evaluate y for each x
43
44     do 200 i=1,nx
45         if(x(i).le.xc) then
46             y(i)=s1*x(i)+y1
47         else
48             y(i)=s2*x(i)+(s1-s2)*xc+y1
49         end if
50     200 continue
51
52 c write the y values to the output file
53
54     6
55     write(6,220)
56 220 format(/,' Writing model output file OUT.DAT...')
57     open(unit=20,file='out.dat')
58     do 300 i=1,nx
59         C S.A. 16/11/2016 Imposto output a campo fisso da 15 caratteri ciascuno
60         C write(20,*) x(i),y(i)
61         write(20,600) x(i),y(i)
62     300 continue
63 C 600 format(' ',E12.3,' ',E12.3)
64 600 format(E15.6,E15.6)
65     close(unit=20)
66     write(6,320)
67 320 format(' File OUT.DAT written ok.')
68
69 end
70

```

Figure 28. Changes applied to the model code *twoline.for*. To notice the comments in red.

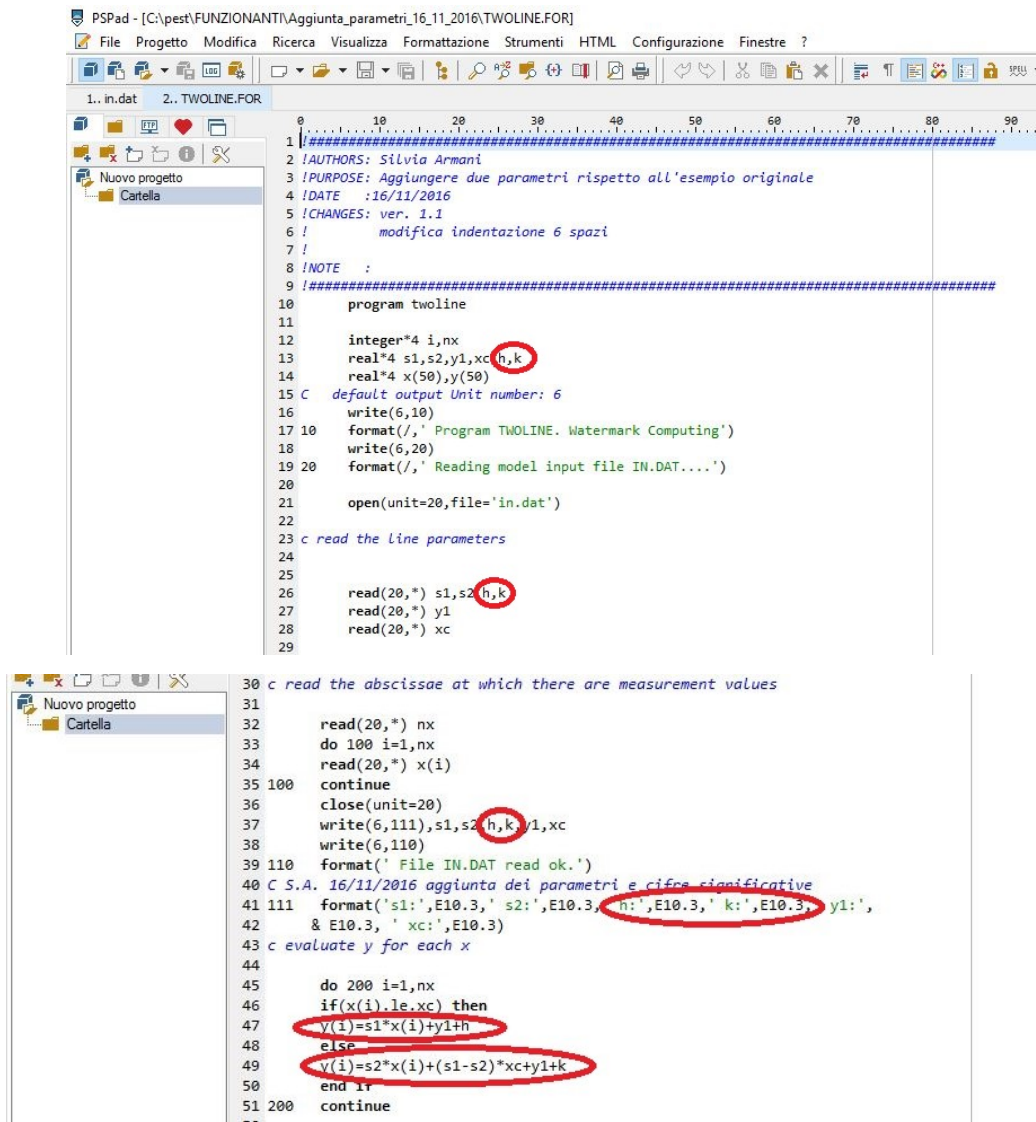
After that, the *twoline.for* code has been saved in executable form by using the G95 compiler and its option “- c” by writing in the command prompt the following:

```
g95 -c twoline.for
```

Finally, the model with its input file *in.dat* has been run to check if the applied changes were correct. Having verified that, PEST has been run as in the original example in Paragraph 4.1 and thus the results are the same.

4.2. New extended example model

In this part of the chapter, two more parameters have been added to the model file in order to instruct PEST to read a more complex model with several parameters to estimate. Again, for doing this, the file *twoline.for* has been modified with the program CodeBlocks as it is visible in **Figure 29** adding the parameter *h* and *k*.



```
1 |#####|
2 |!AUTHORS: Silvia Armani
3 |!PURPOSE: Aggiungere due parametri rispetto all'esempio originale
4 |!DATE :16/11/2016
5 |!CHANGES: ver. 1.1
6 |      modifica indentazione 6 spazi
7 |!
8 |!NOTE :
9 |#####|
10 |      program twoline
11 |
12 |      integer*4 i,nx
13 |      real*4 s1,s2,y1,xc,h,k
14 |      real*4 x(50),y(50)
15 |c default output Unit number: 6
16 |      write(6,10)
17 |10 format(/,' Program TWOLINE. Watermark Computing')
18 |      write(6,20)
19 |20 format(/,' Reading model input file IN.DAT...')
20 |
21 |      open(unit=20,file='in.dat')
22 |
23 |c read the Line parameters
24 |
25 |
26 |      read(20,*) s1,s2,h,k
27 |      read(20,*) y1
28 |      read(20,*) xc
29 |
30 |c read the abscissae at which there are measurement values
31 |
32 |      read(20,*) nx
33 |      do 100 i=1,nx
34 |      read(20,*) x(i)
35 |100 continue
36 |      close(unit=20)
37 |      write(6,111),s1,s2,h,k,y1,xc
38 |      write(6,110)
39 |110 format(' File IN.DAT read ok.')
```

```
40 |C S.A. 16/11/2016 aggiunta dei parametri e cifre significative
41 |111 format('s1:',E10.3,' s2:',E10.3,' h:',E10.3,' k:',E10.3,' y1:',
42 |      & E10.3,' xc:',E10.3)
43 |c evaluate y for each x
44 |
45 |      do 200 i=1,nx
46 |      if(x(i).le.xc) then
47 |      y(i)=s1*x(i)+y1+h
48 |      else
49 |      y(i)=s2*x(i)+(s1-s2)*xc+y1+k
50 |      end if
51 |200 continue
52 |
```



```

52
53 c write the y values to the output file
54
55
56     write(6,220)
57 220 format(//,' Writing model output file OUT.DAT...')
58     open(unit=20,file='out.dat')
59     do 300 i=1,nx
60 C      S.A. 16/11/2016 Imposto output a campo fisso da 15 caratteri ciascuno
61 C      write(20,*) x(i),y(i)
62     write(20,600) x(i),y(i)
63 300 continue
64 C 600 format(' ',E12.3,' ',E12.3)
65 600 format(E15.6,E15.6)
66     close(unit=20)
67     write(6,320)
68 320 format(' File OUT.DAT written ok. ')
69
70 end
71

```

Figure 29. Modified *twoline.for* with two more parameters *h* and *k*. Highlighted in red the changed steps.

More in detail, note that the parameters *h* and *k* have been added as real variables as the others parameters of the model. Also the number of significant digits is the same than the original four parameters. Then the two new parameters have been added in the equations solving the model system trying not to complicate too much the resolution (**Figure 29**). Moreover the indentation of the code has been shifted to the right of 6 spaces just as the Fortran language requires. After that, the new version of *twoline.for* code has been saved in executable form by using the G95 compiler and its option “-c” by writing in the command prompt the following:

```
g95 -c twoline.for
```

Finally, the model with its input file *in.dat* has been run to check if the applied changes were correct. Having verified that, PEST has been run as in the original example in Paragraph 3.4 but modifying all its input files. The initial assigned value for the parameters *h* and *k* was 1.0. All the procedure for preparing PEST has been the same involving also all the utility programs to check the consistency. At the end of PEST estimation, the values of the output file and of the parameters estimated were of course different from the originals results of the example in Paragraph 4.1. The new values are reported below in **Figure 29** for the outputs taken from file *out.dat* and in **Figure 30** for the parameters values extracted from *twofit.par* file.

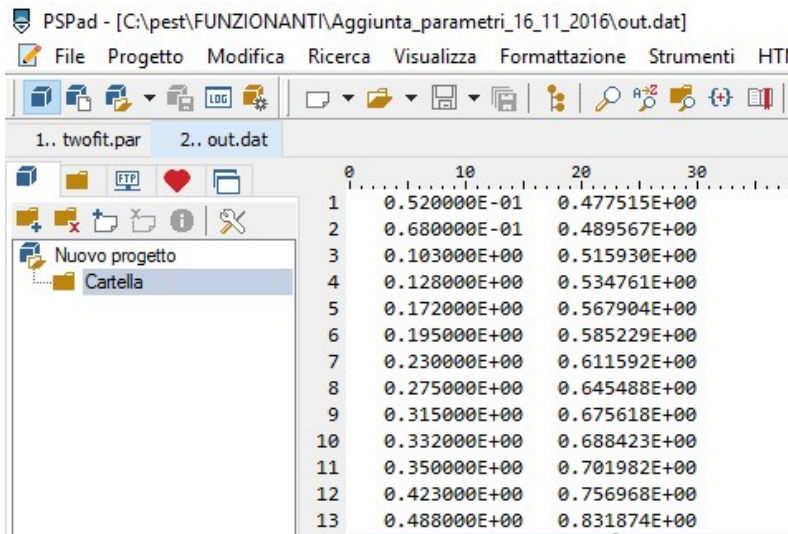


Figure 30. New results from file *out.dat*.

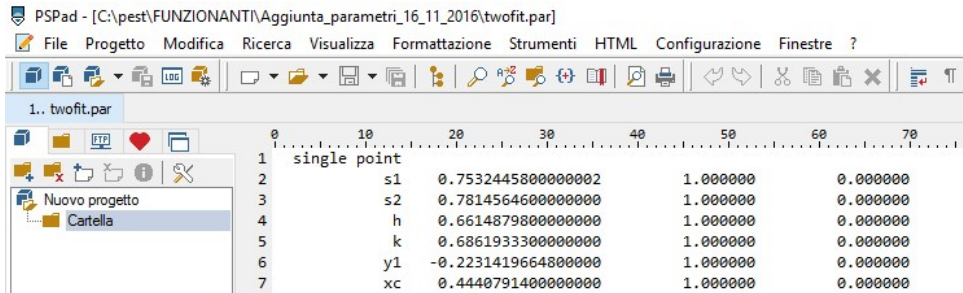


Figure 31. File *twofit.par*. New values of parameters estimated with PEST.

4.2.1. How to read a more complex output file

In this subparagraph, the example of paragraph 3.4 has been modified in order to make PEST read a more complex output file respect to the original one using the skills acquired before in Chapter 2. In this case the model *twoline* has been modified using the text editor PSPad in order to make more complex the structure of the output file of the model. In the **Figure 32** it is reported only the changed extract of the new *twoline.exe* (the original one is visible in **Figure 14**).

```

52 c write the y values to the output file
53
54
55     write(6,220)
56 220 format(/,' Writing model output file OUT.DAT...')
57     open(unit=20,file='out.dat')
58     write(20,220)
59 C   S.A. 17/11/2016 Scritto prima del comando "do" perchè altrimenti scrive il titoletto dopo ogni calcolo di y
60     do 300 i=1,nx
61 C   S.A. 16/11/2016 Imposto output a campo fisso da 15 caratteri ciascuno
62 C   write(20,*) x(i),y(i)
63 C   S.A. 17/11/2016 Scrive nel foglio output quello scritto in riga 56
64     write(20,600) x(i),y(i)
65 300 continue
66 C 600 format(' ',E12.3,' ',E12.3)
67 600 format(10X,E15.6,E15.6)
68 C   S.A. 17/11/2016 10x è il comando per inserire 10 spazi vuoti nel file di output prima della scrittura di x e y
69     close(unit=20)
70     write(6,320)
71 320 format(' File OUT.DAT written ok. ')
72
73 end
74

```

Figure 32. Modified code for *twoline.exe* in order to have a more complex output file.

After that, the model has been run to have the new format of output file and to check that every change does not influence the correct functioning of the model. The new output file is reported in **Figure 33**. It is possible to see the difference in relation to the original one in **Figure 16** (Paragraph 4.1).

The screenshot shows the PSPad editor with the file `out.dat` open. The output content is as follows:

```

1
2 Writing model output file OUT.DAT...
3      0.520000E-01  0.508942E+00
4      0.680000E-01  0.512748E+00
5      0.103000E+00  0.521074E+00
6      0.128000E+00  0.527021E+00
7      0.172000E+00  0.537488E+00
8      0.195000E+00  0.558251E+00
9      0.230000E+00  0.591972E+00
10     0.275000E+00  0.635327E+00
11     0.315000E+00  0.673865E+00
12     0.332000E+00  0.690243E+00
13     0.350000E+00  0.707585E+00
14     0.423000E+00  0.777917E+00
15     0.488000E+00  0.840541E+00

```

Figure 33. New format of output file *out.dat* after having modified the model executable *twoline.exe*.

Now, preparing PEST input files, *in.dat*, *in.tpl*, *in.pmt*, *in.par*, *out.dat*, *out.obf* and *measure.obf* have been left unchanged from the starting example. The file that has been modified is the *out.ins*. It has been changed following the instructions in Paragraphs 2.2, 2.2.1, 2.2.2 and as shown in **Figure 34** below.

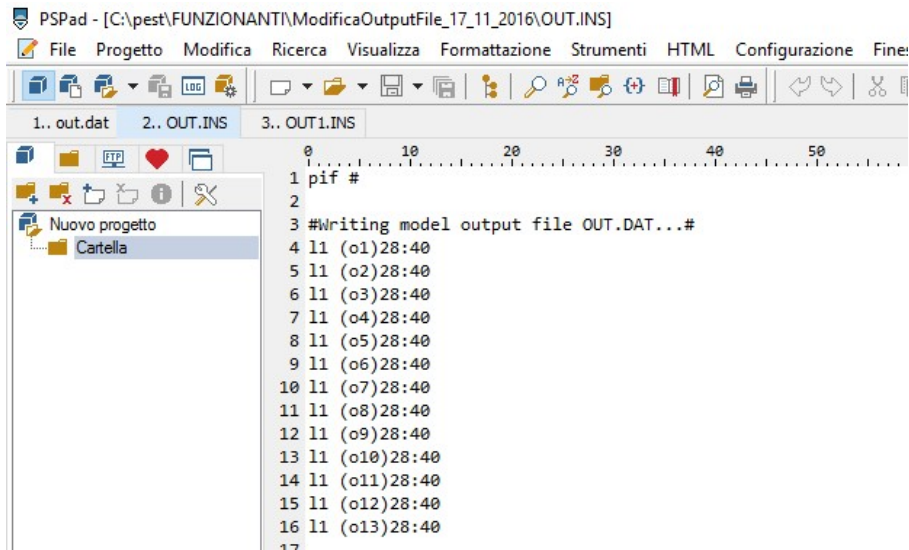


Figure 34. New version of file *out.ins*.

With this operation PEST will run to estimate all the parameters $s1$, $s2$, y_1 and x_c but reading from the output files only the observation from characters 28 to 40 and skipping the title “Writing model output file OUT.DAT” with the help of marker character “#”. After having checked the correctness of all the modified files with PESTCHEK as in the example, PEST has been run. The results are the same of the original example because no data has been varied.

4.2 T2Well numerical model calibration: multilayer high enthalpy geothermal system

Finally, a high enthalpy geothermal model is calibrated with PEST. This model has already been calibrated via trial and error in a PhD Thesis work and it deals with a short production test of a well sited in Commonwealth of Dominica. More in detail now will be reported an extract of a PhD Thesis [Vasini et al., 2015; Vasini, 2016] to describe the characteristics of the location. The well is a vertical slim hole 1200 m deep and producing from a liquid-dominated reservoir. The maximum temperature and pressure, measured under shut-in conditions, are 238°C and 102 bar, respectively.

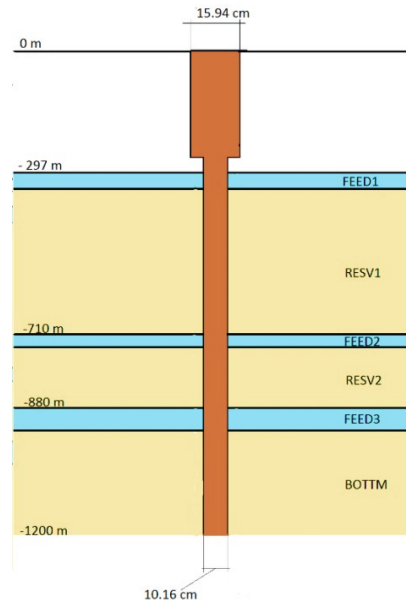


Figure 35. Conceptual model of the WW-01 well-reservoir system: it is possible to see the well WW-01 and the formation [Vasini, 2016].

In **Figure 35** is shown the conceptual model of the numerical model that was used. It is a wellbore-reservoir coupled model constituted of: a cap-rock from 0 to -230 m (elevation referred to the ground); a first feed zone located between -297 m and -344 m (FEED1); a reservoir layer between -344 m and -710 m (RESV1); a second feed zone between -710 m and -734 m (FEED2); a second reservoir layer between -734 m and -880 m (RESV2); a third feed zone between -880 m and -940 m (FEED3).

The model is completed by a low permeable rock domain (BOTTM) below the third feed zone. The well is characterized by a change in diameter at -263 m from an internal diameter of 15.94 cm to an internal diameter of 10.16 cm. The numerical model has been represented by a 2D radial grid with 1658 elements and with the wellbore along the axis of symmetry and with a radial extension of 1500 m.

Figure 36 shows a vertical cross section of the model where the main feed zones can be identified (with lighter color: yellow, green and cyan). In the model the cap-rocks has not been included: the heat exchange between wellbore and the formation between 0.0 m and -297 m has been simulated adopting the analytical approach.

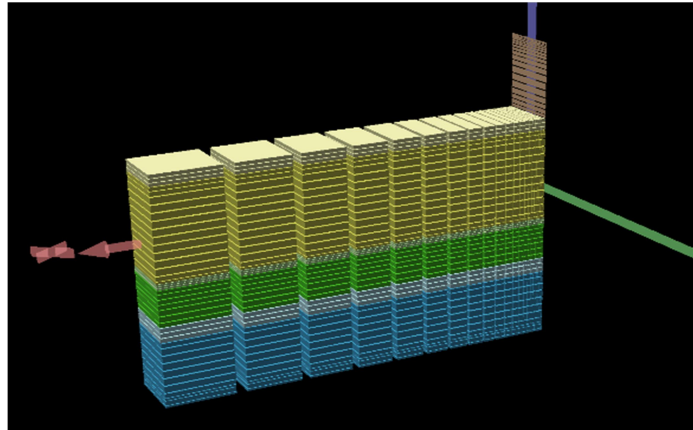


Figure 36. WW-01 wellbore-reservoir model 2D vertical section. The principal feed zones are represented with lighter colors (yellow, green and cyan). This representation has been made by using TOUGH2Viewer [Bonduà et al. 2012].

The shut-in temperature and pressure logs measured in the well can be reasonably supposed be closed to reservoir natural state and therefore used as initial conditions for the steady state simulation (initial T and initial P of the well) are shown in **Figure 37**. Anyway in the case of calibration using PEST only the pressure profile will be considered as observation variables.

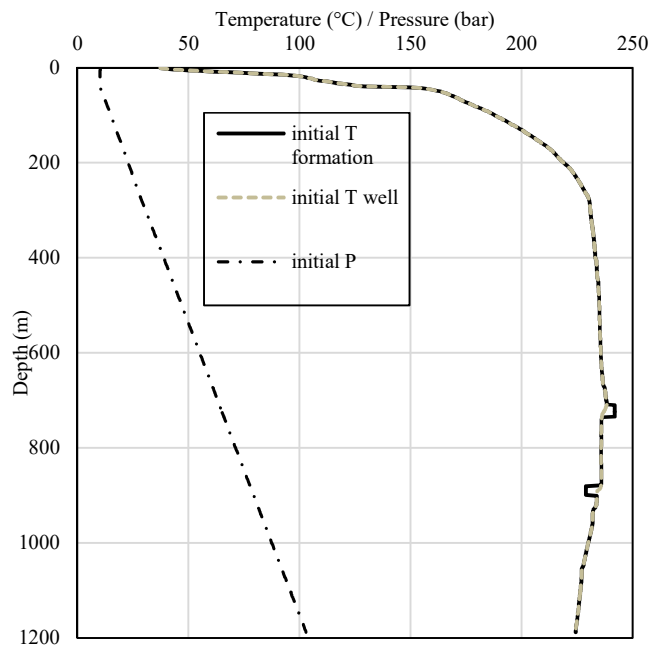


Figure 37.Initial pressure and temperature conditions assumed for the wellbore-reservoir model [Vasini, 2016].

The experimental data available consists in two downhole flowing pressure transients at depths of 800 m and 1180 m, one flowing pressure and temperature log, well-head pressure, (WHP) and enthalpy. The different rock domains permeability was calibrated in order to reproduce the

experimental results. In **Table 2** the results of the calibration of the model obtained manually are listed. In this preliminary study, possible skin effects for both producing feed zones have been neglected.

Table 2.Reservoir formation horizontal permeability as obtained by manual calibration of the model.

Rock type	Perm XY (m ²)
FEED1	$1.8 \cdot 10^{-14}$
RESV1	$1.8 \cdot 10^{-15}$
FEED2	$0.18 \cdot 10^{-12}$
RESV2	$0.5 \cdot 10^{-15}$
FEED3	$30 \cdot 10^{-15}$
BOTTM	$0.02 \cdot 10^{-15}$

As showed in **Figure 38** the measured flowing pressure profile and the simulated one are in good agreement. The percentage difference value for the pressures is about 2.12% [Vasini, 2016].

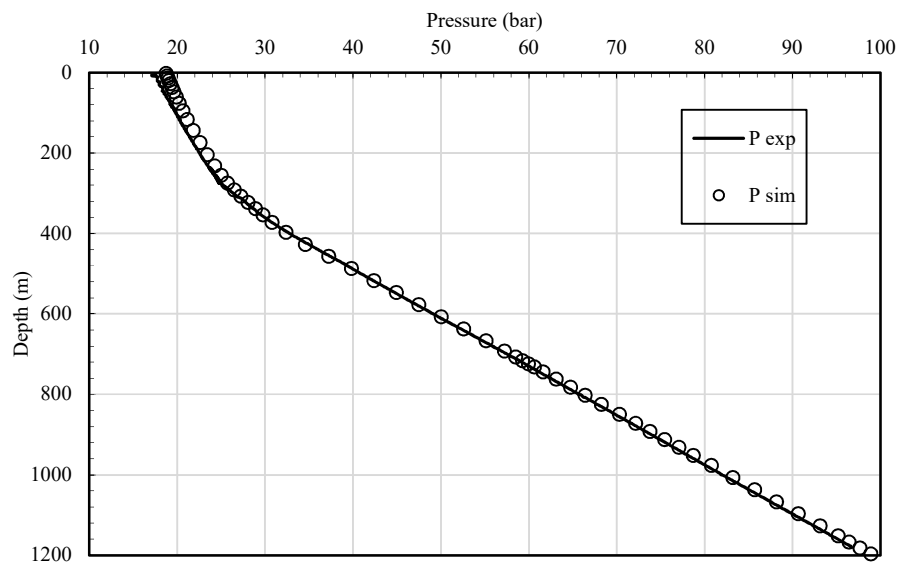


Figure 38. Comparison between measured and simulated flowing pressures. The two set of data show a good agreement.

4.3 T2Well-PEST

It is necessary to specify that for a matter of time and precision, it has been chosen to calibrate only the permeability of the FEED1, RESV1 and FEED2 layers and the output data relative to the dynamic pressure profile have been compared with the experimental data and the ones obtained via trial-and-error approach.

The initial values of the permeability for the three layers of interest have been chosen equal to the one obtained by manual calibration and the values of the lower limit and the upper interval in which varying the permeability parameter to be estimated, have been calculated through a sensitivity analysis. The limit values were found by increasing and decreasing the starting values of a percentage value for which T2Well worked properly. The starting point was 20% and has been decreased iteratively trying to run the executable T2Well until the output file was completely and properly written by the simulator. In **Table 3** all the working boundary values are reported. These values were properly checked by modifying the input file of T2Well and running the executable file called *T2Well-EWASG_12.exe*. If T2Well ran with success the simulation, as previously said, the percentage selected to calculate the range of variation can be considered correct, otherwise the value of the interval has to be decreased.

Table 3. Correct values for the variation interval of the parameters.

Rock type	Initial value	Lower limit bound	Upper limit bound
FEED1	$1.8 \cdot 10^{-14}$	$1.0 \cdot 10^{-16}$	$1.0 \cdot 10^{-12}$
RESV1	$1.8 \cdot 10^{-15}$	$1.0 \cdot 10^{-16}$	$1.0 \cdot 10^{-12}$
FEED2	$0.18 \cdot 10^{-12}$	$1.0 \cdot 10^{-16}$	$1.0 \cdot 10^{-12}$

After this operation, all the input files for PEST have been prepared scrupulously. Starting from the complex input file, only the parameters values to estimate have been substituted with the values obtained and reported in **Table 3**, leaving the rest of the file unchanged. In **Figure 39** the input file prepared for PEST is shown. The initial permeability values have been highlighted in red. After that, also the template file has been prepared following the same procedure illustrated in the example paragraph 4.1. So, it was created by simply copying the input file and substituting the numeric value of the parameters with their name. For simplicity, the names of the permeability of FEED1, RESV1 and FEED2 that have been assigned are p3x, p4x, p5x respectively. The template file called *WW-01_0.9.tpl* is reported in **Figure 40**.

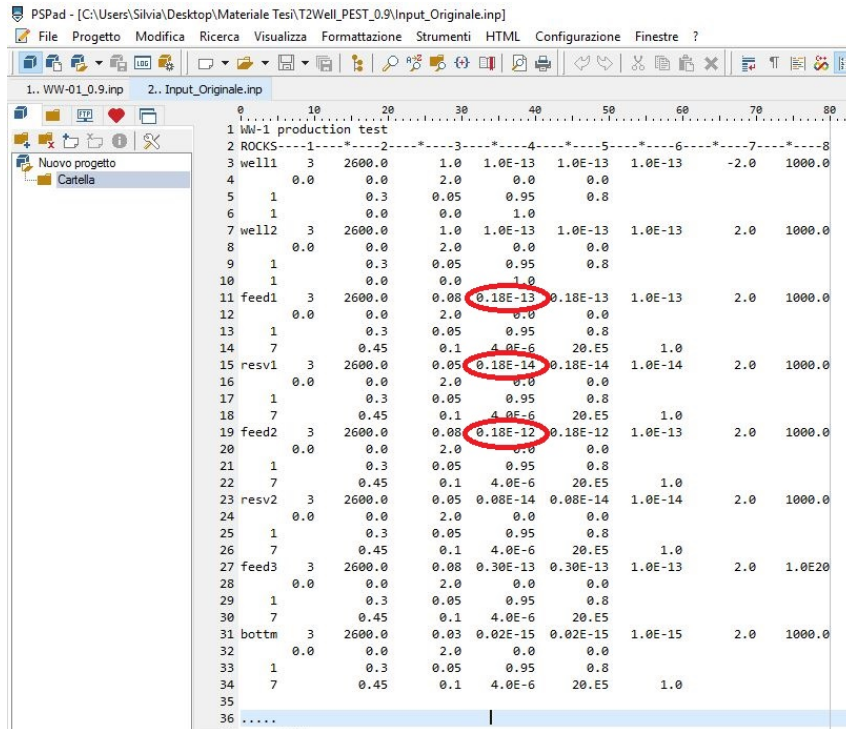


Figure 39. Input file for T2Well-PEST application.

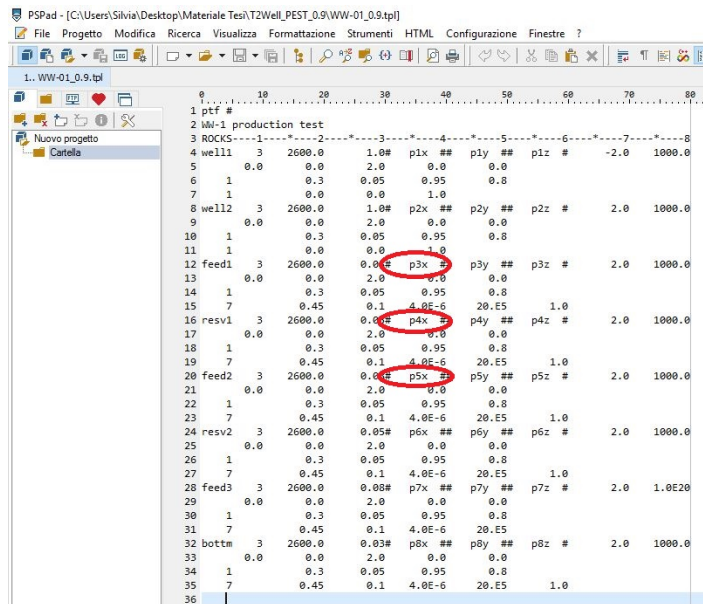


Figure 40. PEST template file called WW-01_0.9.tpl.

Always following step by step the example in paragraph 3.4, the template file has been checked with the utility tool TEMPCHEK and then using again TEMPCHEK the files .pmt and .par has been generated (Figure 41 and 42).

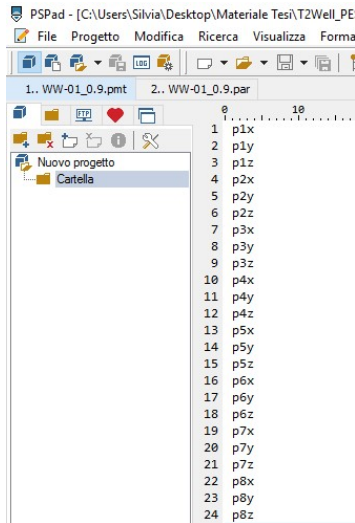


Figure 41. File *.pmt*, where all the parameters names are listed, generated by using the utility tool TEMPCHEK.

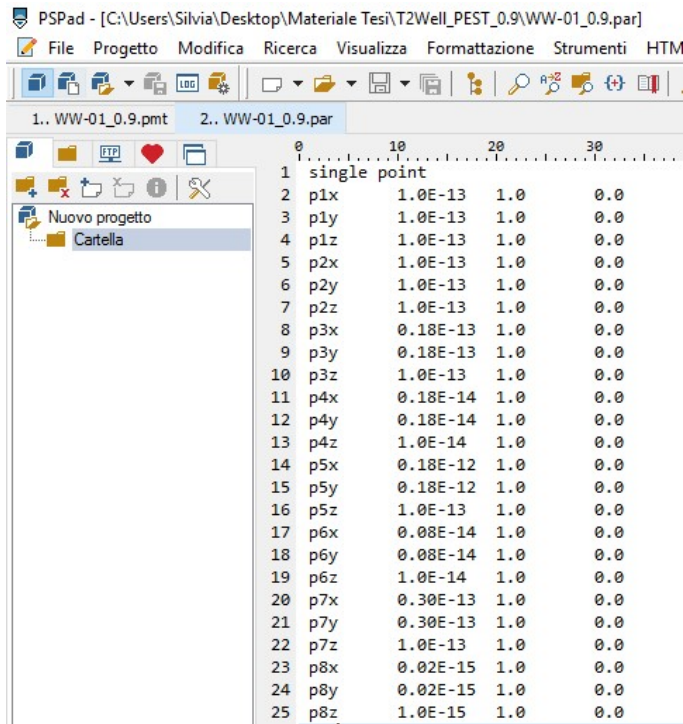


Figure 42. File *.par*, where all the parameters names and values are listed, generated by using the utility tool TEMPCHEK.

Next the instruction file has to be prepared. This is done using a text editor and by writing the instructions for reading correctly only the values of the pressures from the T2Well output file. This instruction file is named *FStatus_2.ins* and is shown partially for a space reason below in **Figure 43**.

```

1 bif @
2 @ Title="Wellbore status"@
3 @ Pres,T@
4 11 (p1)4:23
5 11 (p2)4:23
6 11 (p3)4:23
7 11 (p4)4:23
8 11 (p5)4:23
9 11 (p6)4:23
10 11 (p7)4:23
11 11 (p8)4:23
12 11 (p9)4:23
13 11 (p10)4:23
14 11 (p11)4:23
15 11 (p12)4:23
16 11 (p13)4:23
17 11 (p14)4:23
18 11 (p15)4:23
19 11 (p16)4:23
20 11 (p17)4:23
21 11 (p18)4:23
22 11 (p19)4:23
23 11 (p20)4:23
24 11 (p21)4:23
25 11 (p22)4:23
26 11 (p23)4:23
27 11 (p24)4:23
28 11 (p25)4:23
29 11 (p26)4:23
30 11 (p27)4:23
31 11 (p28)4:23
32 11 (p29)4:23
33 11 (p30)4:23
34 11 (p31)4:23
35 11 (p32)4:23
36 11 (p33)4:23

```

Figure 43. *FStatus_2.ins* instruction file.

Using this instruction set, all model-generated observations are read as semi-fixed observations. Now, as in paragraph 4.1, the program INSCHEK should be used to check that file *FStatus_2.ins* contains a legal instruction set. If there are no errors, INSCHEK should be run again, this time directing it to read the T2Well output file using the instruction set with INSCHEK, the instruction file and the input file with the original data. After this operation, a file named *FStatus_2.obf* will be produced. In this file the values that it reads from the output file for the observations cited in the instruction file are listed. It is possible to view an extract of the file *FStatus_2.obf* in **Figure 44**.

Parameter	Value
1 p1	1816130.
2 p2	1829676.
3 p3	1842196.
4 p4	1856483.
5 p5	1873952.
6 p6	1895085.
7 p7	1920756.
8 p8	1951682.
9 p9	1989433.
10 p10	2035837.
11 p11	2093071.
12 p12	2166749.
13 p13	2253541.
14 p14	2346009.
15 p15	2438023.
16 p16	2518885.
17 p17	2598458.
18 p18	2688522.
19 p19	2761746.
20 p20	2841263.
21 p21	2920756.
22 p22	3000384.
23 p23	3107388.
24 p24	3267059.
25 p25	3480132.
26 p26	3756093.
27 p27	4012742.
28 p28	4269047.
29 p29	4525093.
30 p30	4780894.
31 p31	5036461.
32 p32	5291807.
33 p33	5546948.
34 p34	5799403.
35 p35	5886815.

Figure 44. File *FStatus_2.obf* generated from INSCHEK tool.

The PEST-T2Well interface is now complete as PEST can now generate a model input file and read the output file. Now a PEST control file has to be generated. This file has to provide to PEST an appropriate set of control variables and the set of measured values. First of all the file *FStatus_2.obs* has to be copied to file *experimental_data.obf* then the values of each model-generated observation have to be replaced with the corresponding measured values and the form of the file *measure.obf* is as in **Figure 45** here below.

Parameter	Value
1 p1	1757920.
2 p2	1733490.
3 p3	1771150.
4 p4	1772840.
5 p5	1794700.
6 p6	1836140.
7 p7	1852730.
8 p8	1876310.
9 p9	1915420.
10 p10	1972870.
11 p11	2019450.
12 p12	2100200.
13 p13	2178020.
14 p14	2262810.
15 p15	2347550.
16 p16	2433250.
17 p17	2475030.
18 p18	2585260.
19 p19	2668900.
20 p20	2781110.
21 p21	2867540.
22 p22	2955300.
23 p23	3075200.
24 p24	3256690.
25 p25	3494550.
26 p26	3741220.
27 p27	3987510.
28 p28	4228230.
29 p29	4472850.
30 p30	4721840.
31 p31	4970970.
32 p32	5225970.
33 p33	5477400.
34 p34	5686640.
35 p35	5812250.

Figure 45. File *experimental_data.obf*.

Then run the utility program PESTGEN to generate the PEST control file named *tough2_WW01.pst*; see **Figure 46**.

```

0      10      20      30      40      50      60      70      80
1 pcf
2 * control data
3 restart estimation
4 24 59 24 0 1
5 1 1 single point 1 0 0
6 5.0 2.0 0.3 0.03 10
7 3.0 3.0 0.001 0
8 0.1
9 30 0.01 3 3 0.01 3
10 1 1 1
11 * parameter groups
12 p1x relative 0.01 0.0 switch 2.0 parabolic
13 p1y relative 0.01 0.0 switch 2.0 parabolic
14 p1z relative 0.01 0.0 switch 2.0 parabolic
15 p2x relative 0.01 0.0 switch 2.0 parabolic
16 p2y relative 0.01 0.0 switch 2.0 parabolic
17 p2z relative 0.01 0.0 switch 2.0 parabolic
18 p3x relative 0.01 0.0 switch 2.0 parabolic
19 p3y relative 0.01 0.0 switch 2.0 parabolic
20 p3z relative 0.01 0.0 switch 2.0 parabolic
21 p4x relative 0.01 0.0 switch 2.0 parabolic
22 p4y relative 0.01 0.0 switch 2.0 parabolic
23 p4z relative 0.01 0.0 switch 2.0 parabolic
24 p5x relative 0.01 0.0 switch 2.0 parabolic
25 p5y relative 0.01 0.0 switch 2.0 parabolic
26 p5z relative 0.01 0.0 switch 2.0 parabolic
27 p6x relative 0.01 0.0 switch 2.0 parabolic
28 p6y relative 0.01 0.0 switch 2.0 parabolic
29 p6z relative 0.01 0.0 switch 2.0 parabolic
30 p7x relative 0.01 0.0 switch 2.0 parabolic
31 p7y relative 0.01 0.0 switch 2.0 parabolic
32 p7z relative 0.01 0.0 switch 2.0 parabolic
33 p8x relative 0.01 0.0 switch 2.0 parabolic
34 p8y relative 0.01 0.0 switch 2.0 parabolic
35 p8z relative 0.01 0.0 switch 2.0 parabolic

36 * parameter data
37 p1x fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p1x 1.0000 0.0000 1
38 p1y fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p1y 1.0000 0.0000 1
39 p1z fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p1z 1.0000 0.0000 1
40 p2x fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p2x 1.0000 0.0000 1
41 p2y fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p2y 1.0000 0.0000 1
42 p2z fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p2z 1.0000 0.0000 1
43 p3x none relative 1.80000E-14 1.00000E-16 1.00000E-12 p3x 1.0000 0.0000 1
44 p3y fixed relative 1.80000E-14 -1.00000E+10 1.00000E+10 p3y 1.0000 0.0000 1
45 p3z fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p3z 1.0000 0.0000 1
46 p4x none relative 1.80000E-15 1.00000E-16 1.00000E-12 p4x 1.0000 0.0000 1
47 p4y fixed relative 1.80000E-15 -1.00000E+10 1.00000E+10 p4y 1.0000 0.0000 1
48 p4z fixed relative 1.00000E-14 -1.00000E+10 1.00000E+10 p4z 1.0000 0.0000 1
49 p5x none relative 1.80000E-13 1.00000E-16 1.00000E-12 p5x 1.0000 0.0000 1
50 p5y fixed relative 1.80000E-13 -1.00000E+10 1.00000E+10 p5y 1.0000 0.0000 1
51 p5z fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p5z 1.0000 0.0000 1
52 p6x fixed relative 8.00000E-16 -1.00000E+10 1.00000E+10 p6x 1.0000 0.0000 1
53 p6y fixed relative 8.00000E-16 -1.00000E+10 1.00000E+10 p6y 1.0000 0.0000 1
54 p6z fixed relative 1.00000E-14 -1.00000E+10 1.00000E+10 p6z 1.0000 0.0000 1
55 p7x fixed relative 3.00000E-14 -1.00000E+10 1.00000E+10 p7x 1.0000 0.0000 1
56 p7y fixed relative 3.00000E-14 -1.00000E+10 1.00000E+10 p7y 1.0000 0.0000 1
57 p7z fixed relative 1.00000E-13 -1.00000E+10 1.00000E+10 p7z 1.0000 0.0000 1
58 p8x fixed relative 2.00000E-17 -1.00000E+10 1.00000E+10 p8x 1.0000 0.0000 1
59 p8y fixed relative 2.00000E-17 -1.00000E+10 1.00000E+10 p8y 1.0000 0.0000 1
60 p8z fixed relative 1.00000E-15 -1.00000E+10 1.00000E+10 p8z 1.0000 0.0000 1
61 * observation groups
62 obsgroup
63 * observation data

```

```

0      10      20      30      40      50      60      70      80      90      100     110
64 p1      1.757920E+06 1.0 obsgroup
65 p2      1.733490E+06 1.0 obsgroup
66 p3      1.771150E+06 1.0 obsgroup
67 p4      1.772840E+06 1.0 obsgroup
68 p5      1.794700E+06 1.0 obsgroup
69 p6      1.836140E+06 1.0 obsgroup
70 p7      1.852730E+06 1.0 obsgroup
71 p8      1.876310E+06 1.0 obsgroup
72 p9      1.915420E+06 1.0 obsgroup
73 p10     1.972870E+06 1.0 obsgroup
74 p11     2.019450E+06 1.0 obsgroup
75 p12     2.100280E+06 1.0 obsgroup
76 p13     2.178020E+06 1.0 obsgroup
77 p14     2.262810E+06 1.0 obsgroup
78 p15     2.347550E+06 1.0 obsgroup
79 p16     2.433250E+06 1.0 obsgroup
80 p17     2.475030E+06 1.0 obsgroup
81 p18     2.585260E+06 1.0 obsgroup
82 p19     2.668980E+06 1.0 obsgroup
83 p20     2.781110E+06 1.0 obsgroup
84 p21     2.867540E+06 1.0 obsgroup
85 p22     2.955300E+06 1.0 obsgroup
86 p23     3.075200E+06 1.0 obsgroup
87 p24     3.256690E+06 1.0 obsgroup
88 p25     3.494550E+06 1.0 obsgroup
89 p26     3.741220E+06 1.0 obsgroup
90 p27     3.987510E+06 1.0 obsgroup
91 p28     4.228230E+06 1.0 obsgroup
92 p29     4.472850E+06 1.0 obsgroup
93 p30     4.721840E+06 1.0 obsgroup
94 p31     4.970970E+06 1.0 obsgroup
95 p32     5.225970E+06 1.0 obsgroup
96 p33     5.477460E+06 1.0 obsgroup
97 p34     5.686640E+06 1.0 obsgroup
98 p35     5.812250E+06 1.0 obsgroup

99 p36     5.890210E+06 1.0 obsgroup
100 p37    5.955940E+06 1.0 obsgroup
101 p38    6.019700E+06 1.0 obsgroup
102 p39    6.116400E+06 1.0 obsgroup
103 p40    6.262980E+06 1.0 obsgroup
104 p41    6.426810E+06 1.0 obsgroup
105 p42    6.596380E+06 1.0 obsgroup
106 p43    6.779210E+06 1.0 obsgroup
107 p44    6.984540E+06 1.0 obsgroup
108 p45    7.166610E+06 1.0 obsgroup
109 p46    7.329330E+06 1.0 obsgroup
110 p47    7.491500E+06 1.0 obsgroup
111 p48    7.652000E+06 1.0 obsgroup
112 p49    7.813600E+06 1.0 obsgroup
113 p50    8.016760E+06 1.0 obsgroup
114 p51    8.284210E+06 1.0 obsgroup
115 p52    8.529700E+06 1.0 obsgroup
116 p53    8.775290E+06 1.0 obsgroup
117 p54    9.024020E+06 1.0 obsgroup
118 p55    9.270450E+06 1.0 obsgroup
119 p56    9.432690E+06 1.0 obsgroup
120 p57    9.599930E+06 1.0 obsgroup
121 p58    9.599930E+06 1.0 obsgroup
122 p59    9.599930E+06 1.0 obsgroup
123 * model command line
124 T2Well-EWASG_12.exe<MW-01_0.9.inp>MW-01_Output_0.9.out
125 * model input/output
126 MW-01_0.9.tpl MW-01_0.9.inp
127 FStatus_2.ins FStatus_2
128 * prior information

```

Figure 46. Control file generated with PESTGEN, called *tough2_WW01.pst*.

This control file has been modified fixing all the parameters that are not of interest for this simulation and leaving free to vary in the previously defined intervals the permeability parameters p3x, p4x and p5x. Moreover the last part of the control file, in detail lines 124, 126 and 127 of **Figure 46** have been modified inserting the correct names of the executable file of the model, the input file name, the template file name and the instruction and output file names. Now the preparation of PEST run is complete and as final global check, the utility program PESTCHEK should be run using the command:

```
pestchek tough2_WW01
```

Finally, if all is correct, it is possible to run PEST using the command:

```
pest tough2_WW01
```

All the procedure resulted to be correct and at the end, it is possible to see the results in the final output file *FStatus_2* (small extract in **Figure 47**) and in files generated from the simulation *tough2_WW01.rec* and *tough2_WW01.par* (**Figure 48**) containing the estimated parameters set.

```
1 | Title="Wellbore status"
2 | Pres,T
3 | 0.1794632831265E+07 0.2065016537583E+03
4 | 0.1808025244515E+07 0.2068613258294E+03
5 | 0.1820396968559E+07 0.2071925224521E+03
6 | 0.1834486971760E+07 0.2075677905563E+03
7 | 0.1851713335511E+07 0.2080233694186E+03
8 | 0.1872547439027E+07 0.2085697541072E+03
9 | 0.1897846366508E+07 0.2092264854802E+03
10 | 0.1928312342687E+07 0.2100079506328E+03
11 | 0.1965483712242E+07 0.2109473334098E+03
12 | 0.2011147018839E+07 0.2120814416576E+03
13 | 0.2067422728085E+07 0.2134493638052E+03
14 | 0.2139794370943E+07 0.2151617842218E+03
15 | 0.2224943475162E+07 0.2171175744148E+03
16 | 0.2315533105523E+07 0.2191273665477E+03
17 | 0.2405544823525E+07 0.2210586684245E+03
18 | 0.2484584736475E+07 0.2227024633557E+03
19 | 0.2562651263533E+07 0.2242650968558E+03
20 | 0.2651997571358E+07 0.2259831273379E+03
21 | 0.2724193059374E+07 0.2273187235172E+03
22 | 0.2802230440966E+07 0.2286528304674E+03
23 | 0.2879970604047E+07 0.2299699796076E+03
24 | 0.2961052405917E+07 0.2312770285681E+03
25 | 0.3061966796207E+07 0.2327476432924E+03
26 | 0.3217242457616E+07 0.2347062023651E+03
27 | 0.3432561194690E+07 0.2367382465484E+03
28 | 0.3699897821208E+07 0.2378111272281E+03
29 | 0.3956573196726E+07 0.2379070671689E+03
30 | 0.4212832280756E+07 0.2379991367839E+03
31 | 0.4468839699481E+07 0.2380896677439E+03
32 | 0.4724606636506E+07 0.2381791282106E+03
33 | 0.4980144853851E+07 0.2382663092661E+03
34 | 0.5235467112717E+07 0.2383513408536E+03
35 | 0.5490587357305E+07 0.2384321663683E+03
36 | 0.5703027126752E+07 0.2385076834638E+03
```

Figure 47. Final output file *FStatus_2*.

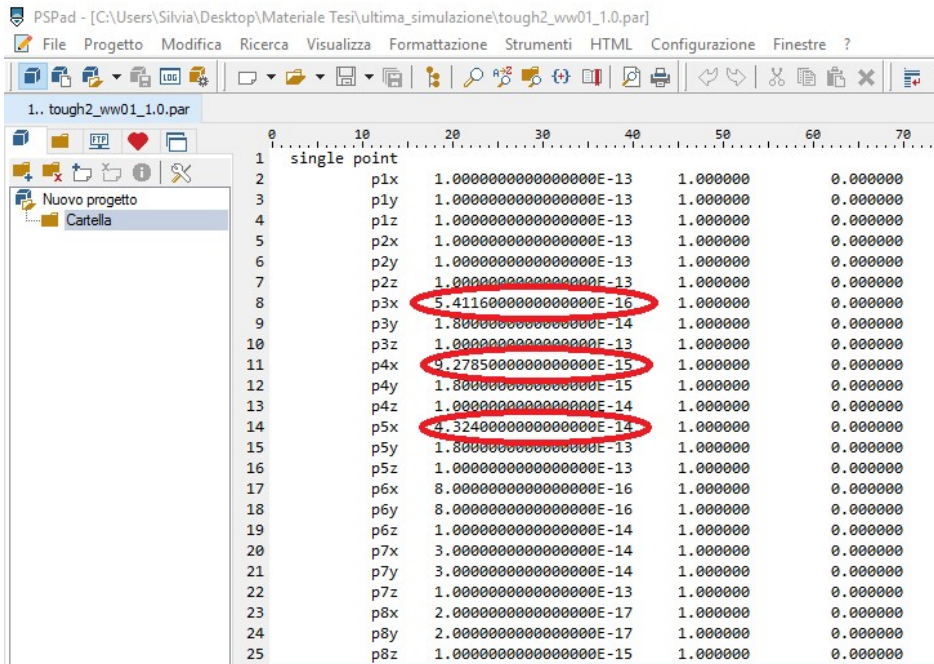


Figure 48. *tough2_WW01.par* containing the estimated parameters set.

As final step, the values of the pressures have been extracted from the output file *FStatus_2* and have been copied to an Excel file in order to compare these values obtained with PEST simulation with the measured ones and with the manually calibrated set. These values have been plotted all together in a unique graph shown in **Figure 49** to see the matching.

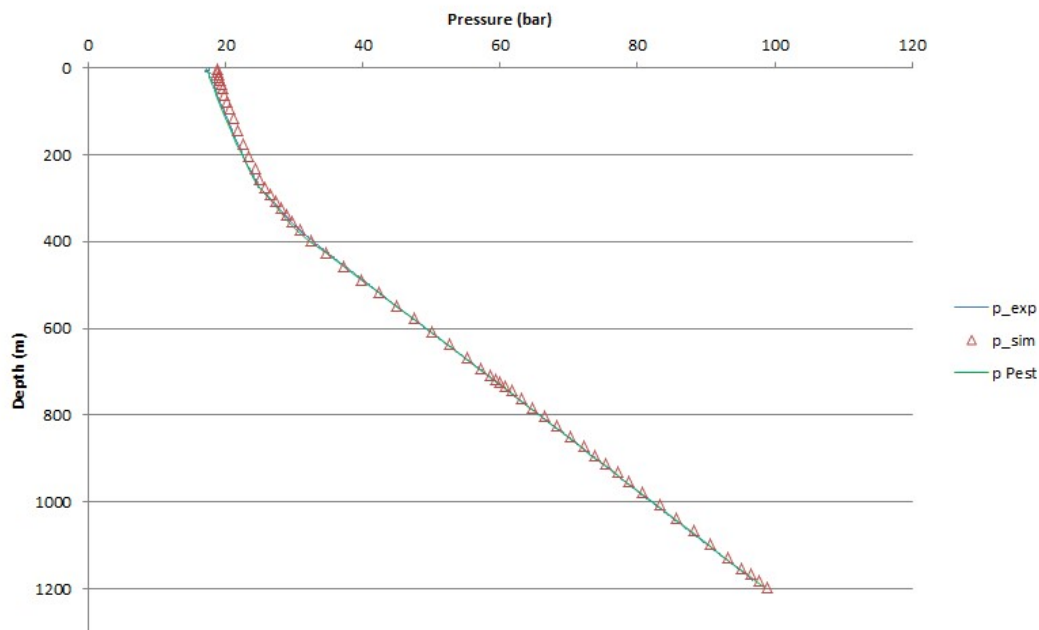


Figure 49. Graph showing the matching between the experimental data, the manually simulated dataset and the values obtained from PEST calibration. All these data are obviously pressures.

The graph shows qualitatively a very good match between the experimental data of the pressure profile and the data obtained from the PEST run. As last check the absolute value of the percentage ratio between the residual and the measured pressure data has been calculated obtaining an average difference value of about 0.5 % that is a better value than the one obtained with *trial-and-error* manual calibration. From this point of view, the use of PEST for the model calibration is convenient because it leads to a more accurate results with lower time consuming respect to the trial and error approach.

CONCLUSIONS

In order to improve the reservoir engineering activities, it has been decided to use PEST (*Model-Independent Parameter Estimation*), a tool for automatic parameter estimation and analysis of the uncertainties of environmental models and in general of complex numerical models, in combination with T2Well-EWASG, a coupled wellbore-reservoir flow simulator, which uses the equation of state EWASG dedicated to work with multiphase-multicomponent geothermal high enthalpy reservoirs.

The calibration platform PEST-T2Well-EWASG, has been then used to calibrate a model concerning a coupled wellbore-reservoir flow geothermal model, referring to a real geothermal reservoir located in Commonwealth of Dominica.

This model was already previously calibrated manually with a *trial-and-error* method in the work for a PhD thesis, and the parameters obtained from this first calibration were used as initial tentative value. Once all the files necessary to PEST were constructed, PEST has been successfully run to automatically calibrate three feed zone (i.e. the permeability of layers) of the model.

The comparison of the simulation output (in particular a pressure profile) obtained from the model calibrated with PEST with the measured pressure profile displayed a good qualitative match. A residual analysis showed an average percentage difference between simulated and measured pressure profile of about 0.5 % that is a better value than the one obtained with the previous *trial-and-error* manual calibration (about 2%).

Thus it is possible to say that, even if the used model is not well representative of the real system, because it is too simple respect to the real system and we do not have a large number of measured data, PEST has proven to be surprisingly a very powerful tool for the automatic calibration of models with a lot of potentialities to be explored. In particular, PEST-T2Well-EWASG could be used to improve the interpretation of well-tests performed in geothermal reservoirs.

References

Battistelli, A., Calore, C., Pruess, K., 1997, *The simulator TOUGH2/EWASG for modelling geothermal reservoirs with brines and a non-condensable gas*. Geothermics, Vol. 26, No. 4, pp. 437-464, 1997.

Battistelli, A., 2012 *Improving the treatment of saline brines in EWASG for the simulation of hydrothermal systems*, In: Proceedings of TOUGH Symposium 2012. Lawrence Berkeley National Laboratory, Berkeley, California. September. 17-19, 2012.

Baveye P. C., Laba M., Mysiak J., 2007, *Uncertainties in Environmental Modelling and Consequences for Policy Making*, Springer in cooperation with NATO Public Diplomacy Division, Vrsar, Croatia, 30 September-11 October 2007.

Bedient, P., 1999, *Ground water contamination: transport and remediation*, Prentice Hall PTR: Upper Saddle River NJ.

Berry, P., Bonduá, S., Bortolotti, V., Cormio, C., Vasini, E.M., 2014, *A GIS-based open-source pre-processor for georesources numerical modeling*, Environmental Modelling and Software, Volume 62, December 2014, Pages 52–64.

Bonduà, S., Berry, P., Bortolotti, V., Cormio, C., 2012, *TOUGH2Viewer: A post-processing tool for interactive 3D visualization of locally refined unstructured grids for TOUGH2*, Computer & Geosciences, 46, pp. 107-118.

Bortolotti, V., 2013, *Lectures on Numerical Simulations of Acquirer*, Lectures notes.

Bortolotti, V., 2015, *Lectures on Petroleum Geosystems-Discretization chapter*, Lectures notes.

Calore, C., Battistelli, A., 2003, *Application of TOUGH2/EWASG to the modelling of salt water injection into a depleted geothermal reservoir: preliminary results*. Proceedings, TOUGH Symposium 2003, Lawrence Berkeley National Laboratory, Berkeley, California, May 12-14, 2003.

Chierici, G.L. *Principi di ingegneria dei giacimenti petroliferi*. ENI, 2004.

Doherty, J., 2016, *PEST, Model-Independent Parameter Estimation, User Manual Part I*, SENSAN and Global Optimisers, 16th Edition, Watermark Numerical Computing.

ELC Electroconsult, *Wotten Waven Geothermal Field, Commonwealth of Dominica, West Indies: Feasibility Study*. Report for the Ministry of Public Utilities, Energy and Ports, Commonwealth of Dominica, 2013 (*unpublished*).

Finsterle, S. 2007. *iTOUGH User's Guide*, Earth Science Division, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA 94720.

Institute of Transportation Engineers, *Transportation Planning Handbook*, 1992, p. 116.

Intel®, 2003-2004, *Intel® Fortran Language Reference*, Document Number: 253261-002, World Wide Web: <http://developer.intel.com>, Intel corporation, 2003-2004.

Osborn, W., Hernández, J., George, A., 2014, *Successful Discovery Drilling in Roseau Valley, Commonwealth of Dominica*. Proc., 39th Workshop on Geothermal Reservoir Engineering Stanford U., Stanford, CA, Feb. 24-26, 2014 SGP-TR-202.

Pan, L., Oldenburg, C.M., Wu, Y. and Pruess, K., 2011, *TWell/ECO2N Version 1.0: Multiphase and Non-Isothermal Model for Coupled Wellbore-Reservoir Flow of Carbon Dioxide and Variable Salinity Water*, Earth Sciences Division, Lawrence Berkeley National Laboratory, University of California, Berkeley, California 94720.

Pan, L., Oldenburg, C.M., 2012, *T2Well – An integrated wellbore-reservoir simulator*, TOUGH Symposium, Lawrence Berkeley National Laboratory, Berkeley, California, September 17-19, 2012.

Pruess, K., Oldenburg, C., Moridis, G., 1999, *TOUGH2 USER'S GUIDE, VERSION 2.0*, Earth Sciences Division, Lawrence Berkeley National Laboratory, University of California, Berkeley, California 94720.

Sheldon, P., *Earth's Physical Resources: An Introduction (Book 1 of S278 Earth's Physical Resources: Origin, Use and Environmental Impact)*, The Open University, Milton Keynes, 2005.

Smith, S., *Water: The Vital Resource (Book 3 of S278 Earth's Physical Resources: Origin, Use and Environmental Impact)*, The Open University, Milton Keynes, 2005.

Taylor D., Pandya A., Thompson D., 2012, *Fundamentals of Model Calibration: Theory & Practice*, OPTUMInsight™, ISPOR 17th Annual International Meeting, Washington, DC USA, 4 June 2012.

Vasini, E.M., Battistelli, A., Berry, P., Bonduà, S., Bortolotti, V., Cormio, C., Pan, L., 2015, *Interpretation of production tests in geothermal wells with T2Well-EWASG*, Proceedings of the TOUGH Symposium 2015, LBNL, Berkeley, California, September 28-30.

Vasini, E.M., *PhD Dissertation.*, *Numerical modelling and simulation optimization of geothermal reservoirs using the TOUGH2 family of codes*, Supervisor: Bortolotti, V, coadvisor: Battistelli, A., Alma Mater Studiorum-University of Bologna, 2016.