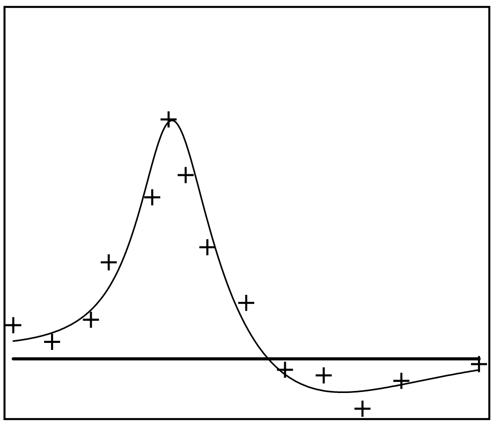
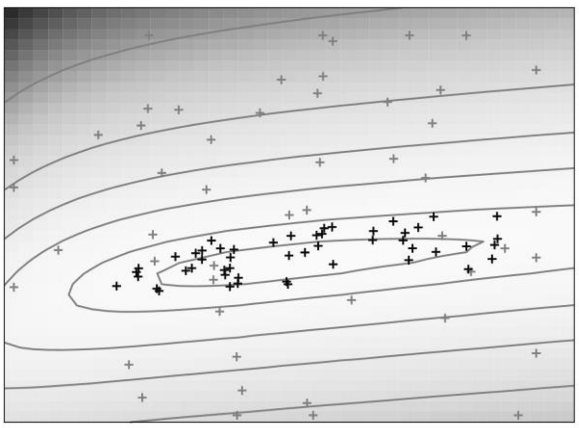
PEST & PEST++

An Overview





John Doherty Jeremy White and David Welter

Watermark Numerical Computing

Supported by Queensland Water Modelling Network

October 2018

Table of Contents

1. Introduction

1.1 Adding Value to Models

Environmental models are used ubiquitously to support environmental management. Ideally, using a model, a management strategy can be tested on a computer before it is implemented in the real world. Different management strategies can be compared, and the best selected. The "best" strategy is often that which balances, in an acceptable manner, the conflicting requirements of cost reduction and resource protection.

If a model is built to simulate future system states and fluxes then, ideally, it should be calibrated against historical system states and fluxes. History-matching reduces the uncertainties of model parameters. This may, or may not, reduce the uncertainties of decision-critical model predictions. These uncertainties must be taken into account when the model is used to support environmental management, for without quantification of uncertainty, there can be no assessment of the risks associated with a particular course of management action. Risk assessment is fundamental to decision-making. See Doherty (2015), Doherty and Vogwill (2015) and Doherty and Simmons (2013) for a detailed discussion of this issue.

History-matching, and post-history-matching uncertainty quantification, are numerically intensive procedures. They require that a model be run many times. In general, the more complex is the system that is simulated, the greater is the number of parameters that a model must possess in order to represent the physical, chemical and hydraulic properties of that system. The greater then becomes the numerical burden of estimating minimum error variance values for these parameters by matching model outputs to historical system behaviour, and of allowing these parameters to vary subject to history-matching constraints so that the uncertainties of predictions of management interest can be explored.

Environmental management (actually, any kind of management) requires explicit or implicit solution of an optimization problem. More often than not, solution of this problem requires minimization of the cost of achieving a management objective subject to certain constraints. These constraints may be the protection of environmental, physical and/or economic assets. If we accept the uncertainties associated with model predictions, then we must recognize that protection of these assets can only be guaranteed at a certain level of confidence. Environmental management may then need to be adaptive so that it can be altered if necessary following re-appraisal of these confidence limits once the repercussions of a particular course of management action begin to be felt. Where management is complex and decision variables are many, solution of such "optimisation-under-uncertainty" problems can be extremely numerically demanding, requiring that many thousands of model runs be undertaken.

Use of models in management of natural systems thus requires the use of complementary software that can run a model many times in order to undertake tasks such as history-matching and optimization. Other model-run-intensive tasks may also be required, these including local and global sensitivity analysis, experimental design, assessment of the worth of new and/or existing data, construction of a proxy model, and more. The model-partner packages which undertake these tasks are just as important to environmental management as the simulators themselves. In the present document, software of this kind is referred to as "model-value-adding software".

A fundamental specification for a model-value-adding package is that it has the ability to run a numerical simulator many times. Two design considerations can enhance the efficiency and flexibility of such a package. These are:

- that it can undertake these model runs in parallel, and
- that it possesses a non-intrusive interface with any model that is under its control.

The benefits of model run parallelization are obvious. The wall time required for completion of a calibration or optimization process can be reduced in proportion to the number of nodes available for conducting model runs. Opportunities for model run parallelization are increasing on an almost daily basis. Modellers have access to multiple CPU's on their own machines, on office networks, on high performance computing clusters and on the cloud.

A "non-intrusive" interface with a model is one that requires no changes to the model's source code, nor re-compilation of the model. It allows a model-value-adding package to test the effects of different parameters or decision variables on model outputs by writing these values to a model's normal input files, and then running the model using a system call. Model outcomes that are of interest to the value-adding package are then read by the package from the model's usual output files on completion of each model run.

The convenience of communicating with a model through the model's own input and output files has a number of importance consequences. One of them is that it allows extreme flexibility in definition of "the model". A model can, in fact, be comprised of a series of executable programs listed in a batch or script file. This series of executables can include more than one simulator. Thus, for example, a water flow simulator may be followed by a contaminant transport simulator; parameters pertaining to both of these simulators may then be jointly estimated using field measurements of water pressure, flow and constituent concentrations. A geothermal reservoir model may be followed by a model which computes alterations to the local force of gravity as steam and water are withdrawn from the porous rocks which comprise the geothermal system. The model can then be calibrated against borehole measurements of enthalpy, temperature and surficial measurements of changes to the earth's gravitational attraction. All simulators that are featured in the batch or script file may be preceded by pre-processors and followed by post-processors. The addition of new features to the overall model only requires that the names of pertinent executable programs be listed in the model batch or script file, and that the value-adding package which runs the model be directed to write additional input files and/or read additional output files as necessary.

The present document describes model-value-adding software that comprises the PEST and PEST++ suites, as well as utility software that expedites the use of these suites in everyday modelling. Members of the PEST and PEST++ suites possess the dual capabilities listed above that together comprise fundamental design criteria for model-value-adding software, namely an ability to undertake model runs in parallel and a non-intrusive model interface. These specifications are, in fact, associated with the run management component of these programs, a subject to which we now briefly turn.

1.2 Non-Intrusive Model Run Management

Members of the PEST and PEST++ suites communicate with a model by writing numbers that represent the current values of parameters or decision variables to the locations that the model expects to find them on its native input files. PEST and PEST++ suite programs are informed of these locations through user-prepared templates of these files. They read numbers from model output files using a set of instructions that are listed in so-called “instruction files”; one such file must be provided for each model output file that must be read. The use of instructions rather than templates preserves flexibility in reading these files; there is no guarantee that a model output file will not change in length, and in content, from model run to model run.

The construction and syntax of template and instruction files is explained in detail in the manuals for both PEST and PEST++. Both of these packages employ these same files.

1.3 Parallel, Non-Intrusive Model Run Management

The principle distinguishing feature between early versions of PEST and parameter estimation software that preceded it was its non-intrusive model interface. An operating system command (i.e. a “system call”) was (and still is) used to run the model. In early versions of PEST, model runs were undertaken on an as-needed basis; only one incidence of the model could be run at any time.

Parallel PEST introduced parallelization of model runs. For Parallel PEST the “manager” (previously named “master”) was also the inversion engine. So-called “workers” (previously named “slaves”) could reside on the same computer as the manager, or on computers to which the manager had access through a network. The working folder of each worker was different from that of any other worker. To commission a model run, the manager first wrote one or a number of model input files to the folder of the worker that it selected to undertake the model run. It then wrote a “semaphore file” to that same folder. This informed the worker that it must now issue the system call to run the model on its local machine. On completion of the model run, the worker wrote its own semaphore file in its local folder to inform the manager that model output files were ready for its perusal. Template and instruction files resided in the manager’s working folder. Model input and output files were written and read across the network.

While this mode of parallel run management precipitated a quantum leap in PEST’s ability to calibrate complex, slow-running models, it was also cumbersome. Continuous checking for the presence of semaphore files is a numerically intensive activity. The writing and reading of model input and output files across a network can be slow if the network is busy. It also requires that the run manager be given permission by the network manager to do this. To add to the difficulties, preparation for a Parallel PEST run was complicated. It required that the user prepare a special file to inform the manager of the folders in which all of its workers were operating.

The advent of BEOPEST alleviated these problems. BEOPEST uses “smart” workers. They, rather than the manager, write model input files and read model output files; hence these files are written and read locally. The manager communicates with each of its workers using the TCP/IP protocol. As BEOPEST commissions a model run, it first sends to the pertinent worker the numbers that it wishes the model to use on that particular run. The worker then writes these numbers to pertinent model input files using templates of these files; the latter reside in the worker’s folder. When the model has finished running, the worker reads numbers of interest from model output files using its own local copy of pertinent instruction files. It then transmits these numbers back to the manager using the TCP/IP protocol. PEST_HP uses a similar run manager to that of BEOPEST. It operates under the same principle. (Note that the original version of the BEOPEST run manager was developed by Willem Schreüder.)

It is apparent from the above description that the decision-support roles played by model-value-adding software requires implementation of at least two complementary, and relatively independent, tasks. One of these is run management. The package that performs this role must be linked to the numerical engine which performs the other of these roles. The latter package commissions model runs yet can remain relatively aloof from the book-keeping and communication tasks that are required to manage these runs. This package can comprise inversion software such as PEST, or a package which undertakes some other kind of model-based decision support such as decision optimisation. Meanwhile the run manager selects workers for the supervision of model runs, sends numbers to those workers for the model to use, and receives numbers back from each worker on completion of a model run. At the same time the manager monitors the performance of each of its workers, assigning runs preferentially to idle workers that reside on machines that can complete those runs in the smallest time. Optionally, the manager can re-assign a model run to a different worker if it loses communication with the worker that is supervising that run, or if an alternative worker suddenly becomes available that can complete the model run in a shorter time.

In the design of PEST++ the important, yet independent, role played by the non-intrusive run manager was recognized by formally separating the run manager from the software package that commissions model runs. All programs of the PEST++ suite employ a stand-alone run manager that can be accessed by any numerical engine engaged in any task that requires the commissioning of parallelized model runs. The run management principles of BEOPEST and PEST_HP are preserved, these requiring the use of smart workers that communicate with a manager using TCP/IP, and that interact with a model using template and instruction files (thus preserving the non-intrusive model interface).

A modular, object-oriented, parallel run manager named GENIE was written to accompany the original version of PEST++; see Muffels et al (2012). Another run manager named YAMR was added to later versions of PEST++; see Welter et al (2015). Run management capabilities offered by both GENIE and YAMR were accessible through a series of function calls from programs written in FORTRAN, C and C++. The availability of these modular non-intrusive parallel run managers marked an important step forward in model-based environmental decision-support. They enabled writers of model-value-adding software to devote their attention to the numerical tasks that this software must perform, without having to worry about the details of parallel model run management.

Programs comprising version 4 of the PEST++ suite (the most recent version at the time of writing) employ a new, modular run manager named PANTHER; see Welter et al (2018). PANTHER is a direct descendent of YAMR. Its principles of parallel run management are the same as those of YAMR, GENIE and the BEOPEST/PEST_HP run managers. However, like GENIE and YAMR it is separated from the numerical engine which it serves; non-intrusive, parallel run management can be accessed by any numerical engine through pertinent function calls. Meanwhile, PANTHER offers more advanced run management functionality than that offered by any previous run manager, this including “dynamic” run management. It is also callable from more programming languages than any of its predecessors.

2. PEST: A Short History

2.1 Initial Release

The first version of PEST was released in 1994. It was characterized as a "model-independent parameter estimator". As was discussed in chapter 1 of this document, model independence was achieved through the use of template and instructions files to communicate with a model.

The first version of PEST was accompanied by a handful of utility programs. These were named INSCHEK, TEMPCHEK, PESTCHEK and PESTGEN. As is apparent from these names, most of them performed error-checking on part or all of a PEST input dataset. Error checking was undertaken by programs other than PEST because of the need to limit PEST functionality to that of parameter estimation and non-intrusive communication with a model. Separation of these tasks was necessitated by the limited amount of memory that was accessible using compilers and computers of that time.

Early versions of PEST implemented the Gauss-Marquardt-Levenberg (GML) method of parameter estimation. Use of this method assumes inverse problem well-posedness. If an inverse problem is not-well-posed, the GML method is incapable of minimizing an objective function that reduces with model-to-measurement misfit. This is because matrices which must be inverted if using this method are singular (and hence non-invertible) if parameters cannot be estimated uniquely.

Unlike the present version of PEST, early versions of PEST were commercial products. With the growing use of groundwater models, and the need for "automatic calibration" of these models, PEST was adopted by commercial groundwater modelling graphical user interfaces – particularly those which supported the popular United States Geological Survey MODFLOW model. In order to encourage its adoption by the groundwater modelling community, a suite of supporting utility programs were written to facilitate use of PEST in conjunction with MODFLOW, as well as with MODFLOW-related programs such as MT3D and SEAWAT. These MODFLOW-support programs comprised the initial members of the Groundwater Data Utility suite. (The term "groundwater data utilities" was used in preference to "groundwater modelling utilities" as a software suite with the latter name already existed at the time.) Programs comprising the Groundwater Data Utility suite served two purposes. Firstly, they made PEST more attractive to developers of groundwater modelling graphical user interfaces, for their existence obviated the need for them to develop utility software to facilitate interaction between PEST and MODFLOW/MT3D/SEAWAT themselves. Secondly, they expedited PEST setup for those modellers who preferred the flexibility of working outside the graphical user interface environment.

2.2 Early PEST Developments

Implementation of the GML method for parameter estimation was both a strength and a weakness of early versions of PEST. Where

- an inverse problem is well-posed, and
- a model is numerically stable so that its outputs are continuous with respect to its parameters,

the GML method of parameter estimation is more efficient than any other. This is an important consideration where model run times are long (as they commonly are for groundwater models), and where computers are slow (as they were in the early days of PEST). A valuable ancillary benefit of GML-based parameter estimation is that, provided the above conditions are met, it can provide linear estimates of post-calibration parameter uncertainty and correlation. These statistics allow a modeller to draw loose inferences on the estimability, or otherwise, for parameters to which he/she is attempting to ascribe values.

Unfortunately, well-posedness of inverse problems through which environmental models are calibrated requires implementation of manual, pre-calibration regularisation. As is discussed by Doherty (2015), this is often a difficult and somewhat subjective, task, which is not always successful. A considerable amount of early PEST development was devoted to heuristic enablement of GML-based parameter estimation in circumstances of incipient inverse problem ill-posedness. These developments included the introduction of user intervention, and then so-called "automatic user intervention", functionality to PEST. Because of these features, and a Marquardt lambda selection and testing methodology that allowed PEST to accommodate near-singular matrices by strategically increasing their diagonal elements, PEST's implementation of the GML methodology was more robust than other GML-based methods of parameter estimation that were available at the time. However if an inverse problem was too ill-posed because of inadequate manual regularisation, PEST simply would not work. Either the objective function would not fall, or PEST would estimate values for parameters which lacked credibility. Because of the prevalence of these problems, and the fact that many modellers did not realize the importance of manual regularisation as a precursor to model calibration, the groundwater modelling industry began to lose confidence in computer-based model calibration.

Similar problems were being encountered in surface water model calibration to those that were being experienced in groundwater model calibration. In the surface water modelling context, problems associated with automatic calibration are compounded by severe model nonlinearity (this sometimes giving rise to local objective function minima), and poorly designed simulation algorithms that create discontinuities between model outputs and parameters (Kavetski et al, 2006). Unfortunately, the surface water modelling culture of the day generally failed to differentiate between non-uniqueness and local optima. This resulted in the development and widespread adoption of so-called "global optimisation methods" for calibration of surface water models. An early (and still popular) global method is the "shuffled complex evolution" (SCE) method developed by Duan et al (2003). An advantage of global methods is that they are often able to provide a good fit between model outputs and field data even when an inverse problem is ill posed and highly nonlinear (provided the problem does not feature too many parameters). Drawbacks of global methods include the following:

- They require a large number of model runs to achieve a good fit between model outputs and a calibration dataset;
- They are generally unable to calculate inference statistics for estimated parameters, nor provide other indications of inverse problem ill-posedness;
- The solution that they achieve to an ill-posed inverse problem cannot be guaranteed to be of minimized error variance; estimated parameters, and predictions which are sensitive to them, may therefore be biased.

Throughout the late nineties, functionality was continuously added to PEST that attempted to improve its speed and stability. These included:

- The ability to use derivatives calculated by a model instead of, or in addition to, those which it calculates itself using finite parameter differences;
- Improved reporting of parameter estimation outcomes;
- Forgiveness of model run failure;
- Sensitivity re-use;
- Methodologies such as "split run analysis" to improve its performance with models whose numerical behaviour is problematical.

2.3 Predictive Uncertainty Analysis

1998 saw the addition of "predictive analysis" mode to normal PEST parameter estimation behaviour. In order to differentiate it from the former, the latter was named "estimation" mode. When run in "predictive analysis" mode, PEST maximizes or minimizes a prediction of interest, subject to the constraint that the calibration objective function rises no higher than a user-specified value. Presumably, this value is slightly higher than that achieved through a previous calibration exercise. The ability to undertake constrained predictive maximization/minimization required only moderate modification of the objective function minimization algorithm that was already implemented by PEST; the algorithm used by PEST to undertake constrained maximization/minimization was developed by Vecchia and Cooley (1987).

The addition of predictive analysis capabilities to PEST's calibration functionality recognized the importance of looking critically at predictions made by environmental models, and of analysing their uncertainties. Practical experience gained through using PEST in predictive analysis mode soon revealed that the uncertainties associated with predictions made by a calibrated model could be high, notwithstanding the attainment of a good fit between model outputs and a calibration dataset. The ramifications for model-based decision-support are obvious. There is now an expectation within the environmental industry that the values of decision-critical predictions made by environmental models be accompanied by estimates of the uncertainties associated with these predictions if models are to be used with integrity for environmental management.

Over the years since it was first introduced to PEST, the constrained maximization/minimization procedure that forms the basis of PEST's behaviour when run in "predictive analysis" mode has undergone many enhancements. Nevertheless, its use has waned somewhat in recent years. There are a number of reasons for this. They include the following:

- The algorithm does not work well where parameter numbers are high. Nor does it easily accommodate parameter nonuniqueness. Unfortunately, especially in groundwater modelling contexts, it is post-calibration parameter nonuniqueness that is often responsible for the lion's share of predictive uncertainty.
- Prediction maximization/minimization subject to calibration constraints is a process that can be applied to only one prediction at a time. Where the uncertainties of many predictions must be explored, the numerical cost of using this methodology can be prohibitive.

- The methodology is numerically “delicate” in that it requires that derivatives of model outputs with respect to parameters be precise. Where model numerical granularity compromises finite-difference calculation of derivatives, it cannot be guaranteed that PEST can indeed find the maximum or minimum value of a prediction subject to calibration constraints.

2.4 Parallelization

Model run parallelization was also introduced to PEST in 1998. As was discussed in chapter 1 of this document, Parallel PEST achieved non-intrusive parallelization of model runs through the use of semaphore files. Parallelisation of model runs considerably reduced the time required to solve inverse problems, making computer-based model calibration possible where it had previously been considered intractable.

With parallelization came the need for management of model runs on different CPUs and on different computers, and for optimization of computing recourses in implementing these runs. The design of the parallel run manager used by Parallel PEST included functionality for maximization of computing efficiency and minimization of computer latency. Many elements of the run manager developed for Parallel PEST have been retained (with some improvements) in BEOPEST and PEST_HP.

2.5 Regularisation

An important milestone in PEST development was the introduction of Tikhonov regularisation to PEST in 2000. This was accessed through a new mode of PEST behaviour, appropriately named “regularisation” mode. PEST’s original implementation of Tikhonov regularisation was based on an algorithm described by De Groote-Hedlin and Constable (1990). Its task when operating in this mode bears some relationship to its task when operating in “predictive analysis” mode, although the underlying algorithm is somewhat different. When run in “regularisation” mode, PEST is asked to minimize a regularisation objective function subject to the constraint that the measurement objective function rises no higher than a user-specified value. The regularisation objective function is a measure of the extent to which parameters adhere to a user-specified preferred set of values or conditions (for example smoothness or homogeneity). As the name implies, the measurement objective function reflects model-to-measurement misfit.

The introduction of Tikhonov regularisation to PEST marked the crossing of a threshold in environmental model calibration. The need for pre-calibration, manual regularisation was eliminated. A modeller could now, in theory at least, obtain a minimum error variance solution to the inverse problem of model calibration, regardless of the number of parameters that the inverse problem.

With regularized inversion came the need for a better parameterization device for groundwater models than zones of piecewise constancy that had been the standard up until that time. Pilot points were introduced to groundwater model calibration by Certes and de Marsily (1991). When deployed in a regularisation framework, the number of pilot points employed by a model can rise from a few tens, to hundreds or even thousands. This introduced unprecedented flexibility to calibration of complex spatial models. The calibration process itself, rather than the modeller, could now introduce heterogeneity to a model domain where its existence proved necessary for model outputs to match field observations. At the same time, the use of Tikhonov regularisation ensures that heterogeneity is introduced to a model's parameter field only to the minimum extent required to achieve a user-specified level of fit with a calibration dataset, and (if regularisation is appropriate) only in ways that make geological sense.

Following the introduction of Tikhonov regularisation functionality to PEST, the Groundwater Data Utility suite was expanded considerably to facilitate and expand the use of pilot points as a spatial parameterization device for groundwater and other models. Not only did this require the development of utilities that undertake interpolation from pilot points to a model grid or mesh. It also required that utility programs be developed for automatic implementation of geologically reasonable regularisation. The development of software to undertake both of these tasks is still ongoing.

In late 1999, PEST ceased to become a commercial product, and was placed in the public domain. With the exception of PEST_HP it, together with all of its utility support software, has remained in the public domain to this day.

2.6 Surface Water Utilities

While models which simulate routing of water to and through rivers on a catchment scale are often numerically simpler than groundwater and subsurface reservoir models, they are often more difficult to calibrate. Reference has already been made to the occurrence of local objective function minima that accompany the calibration of these types of models. Further problems arise from the nature of the calibration dataset itself, this often being comprised of one or a number of time series of stream or river flow. As Doherty and Johnston (2003) and Doherty and Welter (2010) point out, optimal transfer of information from a stream flow time series to the parameters employed by a model requires formulation of a multi-component objective function. The formulation of each component of this objective function requires that the original time series be processed in a different way. Ideally, each objective function component should be rich in information pertaining to a certain aspect of the behaviour of the system, and hence should inform a particular subset of model parameters. Each component of the thus-formulated objective function should be weighted for visibility in the overall objective function which PEST must minimize. (It is worthy of note that this philosophy now forms the centrepiece of approximate Bayesian computation (ABC) methods that are being routinely applied to the calibration of surface water models. See, for example, Nott et al (2011), Sadegh and Vrugt (2013) and papers cited therein.)

To automate construction of complex, multi-component objective functions for use in calibration of surface water models, the TSPROC time series processor was developed in 2001. TSPROC comprises the flagship of the PEST Surface Water Utility suite. Development of TSPROC was partially funded by the University of Idaho. It comprises the PEST interface to the HSPF surface water quantity/quality model supported by the United States Environmental Protection Agency (USEPA) BASINS package. Since 2012 TSPROC has been under the ownership of the Wisconsin Office of the United States Geological Survey (USGS). Staff from that office have added functionality to TSPROC that supports construction of multi-component objective functions for use in a variety of calibration contexts. See Westenbroek et al (2012).

In recognition of the need to accommodate local objective function minima, and of the proclivity of surface water modellers to adopt "global optimization" software for calibration of their models rather than so-called "gradient-based" schemes such as those provided by PEST, two global optimizers were introduced to the PEST suite in 2003. These can be used interchangeably with PEST as they use the same nonintrusive model interface as PEST and read inverse problem specifications from a PEST control file. The first of these is the popular SCE method of Duan et al (1993). The second is the "Covariance Matrix Adaption – Evolutionary Scheme (CMAES) described by Hansen and Ostermeier (2001) and Hansen et al (2003). The latter algorithm is more readily parallelizeable than the former algorithm. Experience also suggests that it is more robust over a broader range of numerical circumstances than SCE. PEST-suite versions of these packages include support for parallelized model runs using the Parallel PEST protocol.

Unfortunately, at the time of writing, neither PEST, nor other inversion packages that support regularised inversion, are widely used in calibration of surface water models. In the author's opinion this situation does not serve the surface water modelling community well. Modern inversion methodologies can support highly parameterized calibration of difficult models, while avoiding entrapment in local objective function minima. This is a matter of some importance when datasets that are available for calibration of surface water models are expanding in complexity and abundance. These datasets include airborne and satellite data, as well as measurements of water quality. Calibration of regional models that are used to maintain water flow and water quality for the protection of delicate ecosystems requires a highly parameterized approach to inversion supported by Tikhonov regularisation in order to enable regionalization of model parameters, while recognizing the complex nature of spatially distributed natural systems. Members of the PEST and PEST++ suites have much to offer in this regard, both in achieving a minimum error variance solution to the inverse problem of calibrating these kinds of models, and in sampling the posterior probability distributions of their parameters.

2.7 Living in a Highly Parameterized World

The addition of Tikhonov regularisation to the PEST inversion algorithm introduced environmental modellers to the possibilities offered by a highly parameterized approach to inversion and uncertainty analysis. At the same time, the possibilities of model run parallelisation offered by Parallel PEST overcame some of the numerical cost of living in a highly parameterized world. Nevertheless many problems remained. The numerical stability of Tikhonov regularisation could not always be guaranteed. The high numerical burden of handling many parameters, although accommodated by model run parallelization, still remained.

Singular value decomposition and LSQR (Paige and Saunders; 1982a, 1982b) were introduced to PEST as solution devices for ill-posed inverse problems in 2003 and 2004 respectively. The "SVD-assist" methodology (Tonkin and Doherty, 2005), was developed in 2005. The latter methodology supported a quantum leap in the numerical efficiency of calibrating highly parameterized environmental models by enabling estimation of so-called "super parameters" instead of parameters themselves. The number of super parameters which require estimation need only be as large as the dimensionality of the calibration solution space as defined by singular value decomposition. This number is generally much smaller than the number of parameters that feature in the calibration process.

Meanwhile the theoretical insights offered by singular value decomposition into what calibration of an environmental model can achieve and, just as importantly, what calibration of an environmental model cannot achieve, were profound. These were explained by Moore and Doherty (2005; 2006) and demonstrated by Gallagher and Doherty (2006; 2007). Various types of linear analysis that were documented and demonstrated in these papers were programmed into new PEST support utilities. This allowed PEST users to conduct similar analyses in their own working environments. These new tools supported not only quick quantification of parameter and predictive uncertainty. They also enabled exploration of data worth and parameter contributions to predictive uncertainty; both of these comprise useful model-complementary analyses in most decision-making contexts. In all of these analyses the inherent complexity of natural systems is acknowledged through an ability to handle as many parameters as is required to represent that complexity. Previous precepts underpinning environmental model calibration that stressed the need for model parameter parsimony could be abandoned. This is a matter of some importance as environmental model usage in the decision-making context requires the opposite of parameter parsimony. Predictive uncertainty analysis has no integrity unless it is accomplished with software that can manipulate the parameters that are required for representation of any system complexity to which a prediction may be sensitive. When quantifying parameter and predictive uncertainty, and when assessing the worth of data acquisition strategies that can reduce this uncertainty, it is obvious that the parameters that cannot be estimated are just as important as those that can. It is the former parameters that are generally responsible for most of the uncertainty that is associated with a prediction of management interest.

PEST’s null space Monte Carlo (NSMC) methodology, released in 2008, offered major new functionality for analysis of the post-calibration uncertainty of predictions made by complex environmental models. Though not strictly Bayesian, NSMC supports generation of a suite of random parameter fields, all of which promulgate good fits between model outputs and the calibration dataset, and all of which are reasonable from an expert knowledge point of view. Collectively they express parameter variability that reflects anticipated real-world heterogeneity, while reflecting the need to maintain the model in a calibrated state when expressing this variability. Once a model has been calibrated, a suite of complex, calibration-constrained parameter fields can be generated with little extra computational cost. See Tonkin and Doherty (2009) for details.

Concurrent with these developments was the development of BEOPEST, for whom Willem Schreüder is primarily responsible. Willem introduced the concept of “smart workers” to nonintrusive run management, while undertaking the programming work required to implement manager-worker communication using both the TCP/IP and MPI protocols. For the everyday modeller, the task of parallelizing model runs under the control of PEST became much easier with the advent of BEOPEST. With BEOPEST deployed to implement SVD-assisted inversion and NSMC-based manipulation of calibration-constrained random parameter fields, highly parameterized inversion and post-calibration uncertainty analysis for large complex models became routine.

2.8 The PEST Conference

A PEST conference was held in Potomac, Maryland from 2nd to 4th November, 2009. Following the developments of the preceding 10 years there was much to talk about. However conversations turned to the future more than the past. A presentation that particularly captured the imagination of conference attendees was that of Luchette at al (2009). This paper documented experiences gained in using BEOPEST in the computing cloud – the latter being a relatively new phenomenon at the time. It quickly became apparent to conference attendees that the large model run requirements that emerge from the need to work in a highly parameterized world can be met quickly and cheaply by running these models in the cloud, especially if a modeller's need for high levels of model run parallelization is only intermittent. See Hunt et al (2010) for a further discussion.

2.9 Pareto Analysis

A fourth mode of behaviour, denoted as "Pareto" mode was added to PEST in 2009. This is designed for use in contexts where parameters are subject to two opposing forces, represented by two opposing objective functions. In the calibration context these are the measurement and regularisation objective functions. In predictive uncertainty analysis, one objective function may denote proximity of a predictive model output to a user-specified value which is deemed to be significant, while the other objective function may specify model-to-measurement misfit. In both of these cases, PEST's "Pareto mode" provides a modeller with information that allows him/her to make a subjective assessment of the appropriate trade-off between the two competing objective functions.

Unfortunately, the running of PEST in "Pareto" mode is a numerically-intensive undertaking. Nevertheless, it introduces direct predictive hypothesis-testing to the suite of PEST's predictive analysis tools. At the same time, it provides a modeller with the information that he/she requires to undertake a subjective evaluation of a predictive hypothesis. This recognizes the important role that model-enhanced subjectivity plays in environmental decision-support.

2.10 Utility Support Software

The years since 2009 have seen more effort devoted to the development of PEST utility support software than to the development of PEST itself. The vision of highly parameterized inversion and uncertainty analysis as an enabler for environmental decision-making can be fully realized only if PEST is complemented by software that supports two- and three-dimensional spatial parameterization and regularisation of complex groundwater, surface water and land use models. The need for more flexible parameterization and regularisation was made more urgent with the development, and immediate acceptance by the groundwater modelling community, of MODFLOW-USG, an unstructured grid version of MODFLOW. While utility support for FEFLOW (which is finite-element based) was already available through a number of Groundwater Data Utility programs which were dedicated to this purpose, the need for more general parameterization software that could support flexible spatial parameterization of any model, regardless of its structure, was obvious.

The response was PLPROC. "PLPROC" stands for "parameter list processor". PLPROC provides its users with a dedicated programming language through which they can endow any model with flexible parameterization based on a variety of parameterization devices. These include pilot points, both as directly representative of environmental system spatial properties, or as devices through which existing parameterization can undergo local refinement. New methodologies for interpolation from pilot points to a model's grid were introduced with PLPROC. These include innovative kriging methodologies based on spatially-varying variograms, as well as radial basis functions.

The development of PLPROC was complemented by further additions to the PEST Groundwater Data Utility suite. New members of the suite supported enhanced regularisation capabilities based on spatially varying variograms. About twenty new utility programs were added to the suite specifically to expedite PEST usage with MODFLOW-USG. Other utilities that were written to support MODFLOW-USG had little to do with PEST; they were added simply because of the utility which they provide. Of particular note in this regard are those that support visualization and display of MODFLOW-USG parameters and outputs using the public domain PARAVIEW visualization platform.

2.11 Model Defects

When undertaking model calibration and calibration-constrained uncertainty analysis, it is incumbent on a modeller to specify the level of the fit which he/she desires between model outputs and field measurements. The need for formulation of a multi-component objective function in calibration of surface water models, and the response to that need embodied in the writing of TSPROC, has already been discussed. While no formal counterpart to TSPROC exists to expedite calibration of groundwater models (though there is no reason why TSPROC cannot be used in this role), a number of new programs were added to the Groundwater Data Utility suite during the early 2010's to provide some measure of flexibility in this regard.

The need for innovative design of an objective function recognizes the fact that even the "best" numerical model is only a simplistic and defective simulator of real-world environmental behaviour. This does not invalidate the important role that models can play in decision-support by virtue of their ability to provide receptacles for environmental information embodied in expert knowledge on the one hand, and measurements of historical system states on the other hand. However it does require that recognition be made of the defective nature of these receptacles, and that steps be taken to preclude history-matching-induced parameter and predictive bias. These important matters were investigated in a series of papers; see Doherty and Welter (2010), Doherty and Christensen (2011) and White et al (2014). Theory developed in these papers, through which the potential for calibration-induced predictive bias can be analysed using linear subspace methods, was encapsulated in new utility support programs added to the PEST suite.

2.12 Tidying Up

By 2015, 20 years after its inception, PEST had been programmed to implement inversion tasks that are far more complex than those for which it was originally designed. This made code maintenance difficult. It also suggested the need to rewrite PEST with highly-parameterized inversion as its principle design consideration. This motivated the development of PEST++, the first version of which was released in 2012. The original version of PEST++ was developed by Dave Welter under sponsorship of the USGS.

Just as urgent as the need to rewrite PEST was the need to rewrite its documentation, and to record in one place all of the insights into environmental model calibration and deployment that had been gained over the previous 20 years of PEST usage. The "PEST Book" (Doherty, 2015) was written in 2015. Meanwhile 2016 was dedicated to a comprehensive rewriting of PEST documentation, this resulting in version 6 of the PEST manual. This version (as well as version 7 of the PEST manual – the most recent at the time of writing) is comprised of two parts. The first part documents PEST itself, as well as the PEST-compatible versions of the SCE and CMAES global optimizers that are supplied with PEST. The second part documents the PEST utility support suite, this suite being comprised of those programs that are downloadable with PEST, and that implement PEST pre- and post-processing, error checking, linear analysis and other problem-independent functionality; see Doherty (2018). These are to be distinguished from the PEST Groundwater Data Utility suite which has a three part manual, and the PEST Surface Water Utility suite which has a one part manual. A separate manual documents PLPROC.

2.13 PEST.cloud and PEST_HP

BEOPEST is now widely used on computing clouds. However those who wish to use BEOPEST on the cloud are faced with two major problems. These are:

- Learning the protocols and functionality of a selected cloud platform;
- Automation of PEST manager and worker setup on computing nodes across that platform.

The second problem also besets those who use BEOPEST on office networks and on high performance computing clusters. A number of high-end PEST users have developed their own software to address these issues. See, for example, Karanovic et al (2012) and Schumacher et al (2018).

S.S. Papadopoulos and Associates of Maryland have developed a commercial, cloud-based service named *PEST.cloud* targeted at occasional PEST users. It runs on the Microsoft Azure cloud platform. When using *PEST.cloud* a modeller is guided through the process of uploading his/her files to the cloud, and of selecting the number of computing nodes on which to conduct model runs. Model calibration is implemented by a version of PEST named PEST_HP whose inversion functionality has been enhanced specifically for use in highly parallelized environments. A modeller pays for usage of *PEST.cloud* on a cpu-minute basis.

PEST_HP can also be purchased as a stand-alone item. It uses a similar run manager to that of BEOPEST, but with some enhancements. For example, file transfer is allowed between the manager and its agents. PEST_HP has its own documentation, separate from that of PEST. It can be used interchangeably with PEST, and hence is supported by PEST utility software.

The commercial PEST_HP suite includes three programs in addition to PEST_HP itself. These are:

- CMAES_HP, a parallel version of the CMAES global optimizer that employs the TCP/IP protocol for communication between manager and agents. (The parallel version of CMAES supplied with PEST employs the semaphore file protocol for communication between manager and agent.) CMAES_HP also supports “file parameters”. These facilitate the use of stochastic parameter fields during an optimization process that seeks the best values for a set of decision variables according to a user-supplied objective function. It can thus implement optimization under uncertainty in a manner similar to that described by Bayer and Finkel (2008) and Bayer et al (2010).
- PWHISP_HP, a “PEST whisperer”. PWHISP_HP reads all PEST_HP output files after completion of a PEST_HP run. It comments on the performance of PEST_HP and

provides voluminous advice on whether, and how, settings of control variables could be changed to improve this performance.
- PCOST_HP, an inversion cost estimator.

Easy-to-use PEST cloud services are also provided by ALGOCOMPUTE, owned by HydroAlgorithmics.

2.14 PEST++ Compatibility

Publication of this document coincides with the release of version 15 of PEST. One of the enhancements that marks the release of version 15 of PEST is the inclusion of a high level of inter-operability between PEST-suite programs and programs of the PEST++ suite. Full details are provided in appendix A of this document. A short description is provided hereunder.

As is discussed in the PEST++ manual (White et al, 2018), members of the PEST++ suite read a PEST control file in order to acquire information on parameters and observations, template and instruction files that support a non-intrusive model interface, prior information equations that constrain the inversion process, and variables that control the operations of inversion, optimization and uncertainty analysis algorithms. Values of control variables that are specific to a particular member of the PEST++ suite are suppled following pertinent keywords on any line within a PEST control file that begins with the characters "++". Comments can also be provided anywhere in a PEST control file; these begin with the "#" character.

All versions of PEST, and a significant number of its utility support programs, have been modified to accept the presence of these "++" lines within a PEST control file. PEST-support utilities which read a PEST control file and write another PEST control file have been programmed to transfer "++" lines to the new PEST control file. PEST and all of these utilities will also accept blank lines within a PEST control file. In addition to this, any text on any line that follows a "#" character is treated as a comment.

In version 4 of PEST++ (whose release also coincides with publication of this document) the "model input/output" section of the PEST control file is split into two sections, these being named "model input" and "model output". PEST also tolerates this change in protocol.

2.15 Programming Language

Most of PEST, and all PEST utility support software, is written in FORTRAN. The only exceptions to this are parcels of code employed by BEOPEST and PEST_HP that handle TCP/IP communication between managers and agents. These are written in C++.

As is discussed in chapter 4 of this document, members of the PEST++ suite are programmed in C++ (as the name suggests).

3. PEST Utility Support Software

3.1 Introduction

Three suites of utility software have been written to support the use of PEST. Because they support the use of PEST, they also support the use of programs belonging to the PEST++ suite.

The first of the three PEST-support utility suites accompanies PEST itself. This suite includes programs CMAES_P and SCEUA_P, the PEST-suite versions of the CMAES and SCE global optimizers. These can be used interchangeably with PEST, and are documented in part 1 of the PEST manual, along with PEST itself. Part 1 of the PEST manual also includes documentation of Parallel PEST and BEOPEST. Meanwhile, part 2 of the PEST manual documents all of the other programs that comprise this first utility suite. Like PEST, they are all run from the command line. Many of them follow a similar initiation protocol to PEST in that they receive information required for their execution through command-line arguments. Users of the UNIX version of PEST receive source code and *makefiles* for these utilities; they are compiled automatically when PEST is compiled.

The second set of PEST-support utilities is the Groundwater Data Utility suite; the PLPROC parameter list processor can be considered to be a member of this suite. Many of the programs comprising this suite were written to expedite the use of PEST with popular groundwater models such as MODFLOW, MODFLOW-USG and FEFLOW. However some members of this suite have nothing to do with parameter estimation; they simply implement common, model-related tasks such as processing of field data and plotting/visualization of model outputs.

The Surface Water Utility suite comprises the third set of PEST-support utilities. The flagship of this suite is the TSPROC time series processor. As was stated above, this is now owned and maintained by the United States Geological Survey. Nevertheless, an older (and useful) version of TSPROC can be downloaded from the PEST web pages.

Each of these utility suites is now briefly discussed. This discussion does not purport to provide usage details of the programs which comprise these suites, nor even mention all of them by name. These tasks are accomplished by the comprehensive manuals that accompany these suites. Instead, the purpose of the discussion presented herein is to allow a modeller to understand how use or PEST and PEST++ can be facilitated through use of the programs which comprise these suites.

Before beginning this description, two other software suites should be mentioned. The first of these is FloPy (Bakker et al, 2016; 2018). This is a Python package for creation and post-processing of datasets pertaining to the MODFLOW family of models. Of greater importance in the present context is PyEMU (White et al, 2016). Python-accessible functionality offered by PyEMU replicates most of that available through linear analysis utilities that accompany PEST. PyEMU functionality also replicates (and in some cases enhances) that provided by some members of the Groundwater Data Utility suite, including automatic construction and manipulation of PEST input datasets, various types of parameter pre-processing and observation post-processing (including plotting), and basic pilot point functionality. If a modeller uses FloPy for MODFLOW model setup, then he/she may find that PyEMU is more suitable for complementary use of PEST with MODFLOW than software comprising the Groundwater Data Utility suite described herein.

3.2 PEST-Support Utilities

3.2.1 General

The term “PEST-Support Utilities” is used herein to refer to over 150 utility programs that are supplied with PEST itself. These, together with PEST, are documented in parts 1 and 2 of the sixth edition of the PEST manual. As stated above, the purpose of this section is not to document these utilities. It is to indicate the types of functionality that different subgroups of these utilities provide. The reader can thus be made aware of their existence, and can understand how they fit into the broader PEST/PEST++ family of software.

Two versions of many of the utilities described below are provided with the WINDOWS version of PEST. One version is named in accordance with descriptions provided below. The names of the alternative versions are prefixed by “*i64*”. Thus, for example, two versions of the PREDUNC7 executable program are named *predunc7.exe* and *i64predunc7.exe*. The “*i64*”prefix denotes the fact that the pertinent executable program is compiled using the INTEL 64 bit compiler. Executable programs with this prefix run considerably faster than their 32 bit counterparts.

Many of the utilities discussed below can be used equally well with PEST++ as with PEST. In general, if a utility program reads a PST file (i.e. a PEST control file), a PAR file (i.e. a parameter value file), and/or a JCO file (i.e. a binary Jacobian matrix file), then it can be used with both PEST++ and PEST. However, if it reads a REC file (i.e. a run record file), then it is not useable with programs of the PEST++ suite.

3.2.2 PEST

Two WINDOWS executable versions of PEST are provided with the PEST suite. These are *pest.exe* and *i64pest.exe*. These comprise the serial version of PEST. Parallel PEST also comes in two versions, namely *ppest.exe* and *i64ppest.exe*. Parallel PEST uses semaphore files to communicate with a slave program named PSLAVE. PSLAVE has no intelligence. It does not write model input files nor read model output files. It simply issues a system call to run a model on receiving a message from the Parallel PEST master through a semaphore file; when the model has finished execution, it signals the master of this occurrence by writing another semaphore file.

Like Parallel PEST, the WINDOWS version of BEOPEST comes in two versions – a 64 bit version named *beopest64.exe* and a 32 bit version named *beopest32.exe*. As is documented in part 1 if the PEST manual, the (smart) worker (i.e. slave) program is also BEOPEST, but run with a slightly different command so that BEOPEST knows whether it should act as the manager or a worker when it commences execution. Either *beopest64.exe* or *beopest32.exe* can serve as a worker for a *beopest64.exe* or *beopest32.exe* manager.

The only WINDOWS version of PEST_HP is a 64 bit executable named *pest_hp.exe*. The worker program is the same. BEOPEST cannot be used as a worker for PEST_HP as the latter includes some run management functionality that is not available in BEOPEST.

3.2.3 Global Optimisers

CMAES_P and SCEUA_P are PEST-compatible versions of the CMAES and SCE global optimizers. If run using the “/p” command-line switch they undertake model runs in parallel. They do this using the Parallel PEST protocol; PSLAVE is their slave. Because of the nature of its optimization algorithm, SCEUA_P is difficult to parallelize. Hence parallelization takes place at the so-called "complex" level rather than at the model run level. In SCE parlance, a "complex" is a group of evolving parameters. Information about model outputs calculated using these parameters is exchanged by members of the complex to form the basis for parameter improvement. At regular intervals complexes are "shuffled", and new complexes are formed.

CMAES_HP is supplied with PEST_HP. This conducts model runs in parallel. It uses TCP/IP for communication between the manager and its workers. CMAES_HP is the manager while PEST_HP (whose execution is initiated with an appropriate command-line switch) is the (smart) worker; there can be many workers.

3.2.4 Checking Utilities

The TEMPCHEK, INSCHEK and PESTCHEK utilities were supplied with the original version of PEST. "C" is missing before "K" in the names of these utilities because they were written when the DOS operating system restricted filename lengths to 8 characters plus a three character extension. TEMPCHEK checks the integrity of a template file, and optionally writes a model input file. INSCHEK checks the integrity of an instruction file, and optionally reads a model output file. PESTCHEK checks the entirety of a PEST input dataset (the PEST control file, all template and instruction files cited therein, and all covariance matrix files cited therein) for correctness and consistency.

3.2.5 Building and Altering a PEST Control File

Where many parameters are being manipulated, and/or where the number of observations comprising the calibration dataset is large, manual manipulation and editing of the contents of a PEST control file can be time-consuming and error-prone. A number of utility programs assist in these tasks.

The PARREP utility replaces parameter values in an existing PEST control file (i.e. PST file) with a new set of values read from a parameter value file (i.e. a PAR file). Often these values would have been estimated in a previous inversion exercise. OBSREP does for observations what PARREP does for parameters. ADDREG1 and ADDREG2 read a PEST control file in which PEST is asked to run in "estimation" mode; they supplement the inversion process with "preferred value regularisation", thereby creating a new PEST control file in which PEST is asked to run in "regularisation" mode, and in which a suite of prior information equations define regularisation constraints. As the name implies, SUBREG1 performs the opposite task.

PWTADJ1 adjusts weights in a PEST control file so that the contribution made to the initial objective function by all observation groups is about the same; it reads a run record file based on current weights to ascertain current values for objective function components so that it can formulate new weights appropriately. PWTADJ2 prepares a PEST control file for uncertainty analysis. Like PWTADJ1 it reads a run record file to obtain the values of objective function components corresponding to current weights. It then adjusts these weights so that the value of the objective function component corresponding to each observation group is equal to the number of non-zero-weighted observations in the group. Each weight is thus roughly equal to the inverse of the standard deviation of "measurement noise" as it affects the observation group to which it belongs.

PSTCLEAN removes all PEST++ features from a PEST control file (including "++" lines and comments), writing another PEST control file in the process. The new PEST control file is readable by all PEST utilities, and not just those that have been modified for compatibility with PEST++.

SVDAPREP is a utility program of some importance. It writes a PEST input dataset for estimation of super parameters based on a PEST input dataset that features base (i.e. normal) parameters. This new PEST input dataset includes not just a super parameter PEST control file; it also includes the batch file that PEST must run as "the model" when undertaking SVD-assisted inversion. This batch file cites the PARCALC and PICALC utilities. PARCALC calculates current values of base parameters from those of super parameters. PICALC evaluates prior information equations that feature base parameters, despite the fact that it is provided with the values of super parameters. SVDAPREP writes template files for the input files required by both of these programs.

3.2.6 JCO File Construction and Manipulation

A "JCO file" is a binary file that contains a Jacobian (i.e. sensitivity) matrix. This type of file is integral to parameter estimation, and to parameter/prediction sensitivity/uncertainty analysis. JCO files written by PEST and PEST++ are compatible with each other. JCO file manipulation utilities that are part of the PEST suite can therefore be used with programs of the PEST++ suite.

Perhaps the most useful JCO file manipulation utility is JCO2JCO. This utility reads two PEST control files, one of which has a complementary JCO file and the other of which does not. JCO2JCO writes a new JCO file to complement the second PEST control file. The second PEST control file may feature fewer parameters and/or observations than the first; its parameters may have different transformation status and/or different values from those of the first. JCO2JCO undertakes the necessary sensitivity conversions.

Part or all of the contents of a binary JCO file can be recorded in ASCII format for user-inspection using utility programs such as JACWRIT, JCO2MAT, JROW2VEC and JCOL2VEC.

JCOPCAT builds a complete JCO file from fragments of a JCO file. This can be useful if parameters are added to an existing PEST input dataset and a modeller does not wish to re-compute sensitivities for existing parameters in building a new JCO file to complement the new PEST control file. JCOORDER, JCOADDZ, JCOSUB, JCOZERO and JCOCOMB manipulate the contents of an existing JCO file. JCOSUM and JCODIFF form a new JCO file based on the sum or difference of existing JCO files.

WTSENOUT calculates $\mathbf{Q}^{1/2}\mathbf{J}$, where $\mathbf{Q}$ is the weight matrix associated with an inversion problem.

JCOCHEK ensures that a JCO file (possibly written by one of the utilities discussed above) is fully compatible with its corresponding PEST control file.

3.2.7 The Integrity of Finite-Difference Derivatives

PEST calculates derivatives of model outputs with respect to parameters using finite differences. (Other options are available; this is the option that is most commonly employed.) In carrying out this task, each adjustable parameter in sequence is varied incrementally. The partial derivative of a particular model output with respect to the parameter is approximated by dividing the corresponding model output difference by the parameter difference; if a parameter is denoted as being log-transformed in the PEST control file, derivatives with respect to that parameter take this transformation into account. Where the numerical performance of a model is hampered by problems such as incomplete convergence of its numerical solver, finite-difference derivatives may be compromised. This can result in a deterioration of PEST's performance.

JACTEST checks the integrity of finite-difference derivatives by varying a user-specified parameter a number of times in succession, and then recording a table which contains the values of all model outputs calculated using the incrementally-varied values of that parameter. This table can be imported into a spreadsheet such as Microsoft EXCEL for plotting. The POSTJACTEST utility can be used to quickly identify problematical derivatives.

MULJCOSEN reads a sequence of JCO files recorded over the course of a PEST run (if PEST is asked to record iteration-specific JCO files). It identifies discrepancies between sensitivities contained in these files which may be indicative of poor numerical derivatives.

3.2.8 Model Pre- and Post-Processing

There are many occasions where an inversion process is well served if the parameters that are adjusted by PEST are transformed counterparts of those employed by the model. The same considerations often apply to model outputs and the observations to which they are matched. The PAR2PAR and OBS2OBS utilities were written to implement the necessary transformations. PAR2PAR undertakes manipulation of PEST parameters based on user-supplied equations of arbitrary complexity. It then writes model input files using templates of these files. OBS2OBS reads numbers from model output files using instruction files. It manipulates these outputs using modeller-supplied equations of arbitrary complexity, and records the outcomes of its calculations in a file that is easily readable by an instruction file. OBS2OBS writes this instruction file itself.

Note that functionality provided by PAR2PAR is directly accessible through the "secondary parameter" functionality of PEST_HP. Use of PAR2PAR is thus unnecessary when using PEST_HP.

3.2.9 Statistical Postprocessing of a PEST Run

After PEST has been run in "estimation" mode, the EIGPROC utility can be used to provide a "report card" of the health (or otherwise) of the parameter estimation process. After reading a number of PEST output files, EIGPROC summarizes its findings in a set of tables that are easily human-readable. Meanwhile INFSTAT can be used in the same context to calculate so-called "influence statistics". These quantify the effects that different observations exert on values estimated for parameters.

In highly parameterized contexts where inverse problems are generally ill-posed, different statistics must be employed for characterization of these problems. These generally emerge from singular value decomposition of the weighted Jacobian matrix. SUPCALC estimates the dimensions of the calibration solution and null spaces, while IDENTPAR calculates parameter identifiabilities. SSSTAT provides a complete range of subspace statistics, while INFSTAT1 attempts to quantify observation influence in highly parameterized contexts.

"Super parameters" and "super observations" (orthogonal combinations of parameters, and the orthogonal combination of observations which inform them, as calculated through singular value decomposition of the weighted Jacobian matrix) can be formulated using the

SUPOBSPAR and SUPOBSPAR1 utilities. These can be useful for tracking the flow of information from a calibration dataset to model parameters. As has already been discussed, the SVDAPREP utility builds a PEST control file which is formulated to estimate super-parameters directly. This is the basis for SVD-assisted inversion. Super parameter adjustment is also an important component of null space Monte Carlo analysis of parameter and predictive uncertainty.

3.2.10 Linear Uncertainty and Error Analysis

A suite of PEST utilities was developed to undertake linear uncertainty and error analysis. Input files for most of these programs include a PEST control file, a corresponding JCO file, and a parameter uncertainty file (which specifies pre-calibration parameter uncertainties). Optionally, the PWTADJ2 utility can be used to construct an analysis-ready PEST control file in which measurement weights are the inverse of “measurement noise” back-calculated from model-to-measurement fit achieved through a previous inversion process. Uncertainty calculations are based on the linearized version of Bayes equation. Error calculations are based on singular value decomposition.

Uncertainty and uncertainty-dependent calculations are implemented by members of the PREDUNC* suite of utilities. PREDUNC1 calculates the pre- and post-calibration uncertainty of a parameter or prediction. PREDUNC5 calculates how much an existing or proposed measurement (or suite of measurements) can reduce this uncertainty; this is useful for evaluation of data worth. PREDUNC4 calculates the contribution made to the uncertainty of a user-specified prediction by a particular parameter, or by a group of parameters. PREDUNC7 calculates a posterior covariance matrix.

The PREDVAR* utilities perform similar roles to their PREDUNC* counterparts. However they calculate parameter/predictive error rather than uncertainty. In general, it is better to work with uncertainty than error; the latter can be somewhat arbitrary, its value depending on the singular value index at which demarcation of the solution and null spaces is deemed to occur. There are two members of the PREDVAR* suite, however, that have no PREDUNC* counterparts. These are PREDVAR1B and PREDVAR1C. These are used to explore model-simplification-induced parameter and/or predictive bias. Simplification is inherently a subspace concept that is difficult to express in Bayesian terms.

GENLINPRED is a kind of “umbrella utility” that runs many of the PREDUNC* and PREDVAR* utilities behind the scenes after asking the user a series of questions. This can simplify use of these utilities in certain contexts.

As has already been mentioned, most of the functionality provided by both the PREDVAR* and PREDUNC* suites is also provided by PyEMU.

The PEST suite includes a number of older linear analysis utilities which are retained in the suite despite the fact that their use is no longer recommended. These includes RESPROC, RESWRIT, PARAMERR, REGERR, and the PREDERR* suite of programs.

3.2.11 Nonlinear Uncertainty and Error Analysis

As was discussed in chapter 2 of this document, PEST has a mode of behaviour known as “predictive analysis” mode. This mode of operation allows a modeller to explore the extent of post-calibration predictive uncertainty by direct maximization/minimization of that prediction subject to the calibration objective function rising no higher than a user-specified value. Model linearity is not assumed; hence the method is general. Unfortunately, use of this methodology for exploration of predictive uncertainty becomes numerically intractable where parameter numbers are high. PEST can also be run in “Pareto” mode to explore post-calibration predictive uncertainty; the numerical burden incurred by running PEST in this mode can be high.

Monte Carlo analysis of parameter and predictive uncertainty requires generation of many random parameter fields. In the post-calibration context, these parameter fields must retain as much of their randomness as possible, while promulgating a good fit between pertinent model outputs and members of the calibration dataset. The RANDPAR and RANDPAR1 utilities generate random parameter fields based on user-supplied parameter standard deviations and/or one or more covariance matrices. For a linear model, calibration constraints on these parameter fields are automatically enforced if RANDPAR or RANDPAR1 sample a PREDUNC7-calculated posterior parameter covariance matrix. Alternatively (or as well), the PNULPAR utility can be used to establish calibration constraints on these parameter fields through null space projection.

Monte Carlo analysis requires that a model be run many times in succession, using a different parameter field on each occasion. PEST, BEOPEST and PEST_HP can be run using the “/h” switch to supervise these runs. (BEOPEST and PEST_HP conduct these model runs in parallel.) PESTPP-SWP, supplied with the PEST++ suite, can be used for the same purpose; it also parallelizes model runs. Assistance in the reading and collation of multiple model output files which record model-calculated predictions is available through the RDMULRES, COMFLENME, COMFLENME1 and MULPARTAB utilities.

Where PEST is used in “Pareto” mode to undertake direct predictive hypothesis-testing, the ASSESSPAR utility can be used to quantify the likelihood of a particular parameter field.

Latin hypercube sampling is implemented by a program named LHS developed by Sandia National Laboratories. This is downloaded with the PEST suite. The PEST2LHS, LHS2PEST and PHISTATS utilities facilitate data exchange between PEST and LHS.

3.2.12 Matrix Manipulation

A set of over 20 utilities is provided with PEST that manipulate the contents of “matrix files”. Each of these files contains a matrix or vector. Matrix manipulation functionality provided by these utilities includes the following:

- matrix-matrix and matrix-vector products;
- matrix transposition;
- singular value decomposition;
- formation of a correlation matrix from a covariance matrix;
- production of a JCO file from a matrix file, and vice versa;
- extraction of matrix diagonal elements;
- formulation of matrices of the type $\mathbf{X}^t\mathbf{X}$ and $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}$;
- extraction of rows and columns from a matrix.

3.2.13 Manipulation of RRF Files

An “RRF file” is a run results file. PEST, BEOPEST and PEST_HP write this type of file if run using the “/f” command-line switch. When execution of any of these programs is initiated with this switch, they read a set of parameter value files (i.e. “PAR files”); they then run the model using the values of parameters recorded in each of these files. Parameter and corresponding model output values are recorded in the RRF file.

Because it records the outcomes of many model runs that are based on many parameter values, the contents of an RRF file can serve many purposes. These include sensitivity and uncertainty analysis, proxy model construction, and calculation of a randomized approximation to a Jacobian matrix.

RRFCALCPSI calculates objective functions based on parameter and model output values recorded in an RRF file. RRF2JCO calculates an approximate Jacobian matrix based on covariances between parameters and model outputs exhibited by the contents of an RRF file. RRFCAT, RRFCULL and RRFCLEAN manipulate the contents of RRF files in ways suggested by their names. RRF2PAR, RRF2PAROBS and RRF2TAB rewrite the contents of an RRF file in other formats.

3.2.14 Advice

PWHISP_HP, the "PEST whisperer", accompanies PEST_HP. It cannot be used in conjunction with other versions of PEST as it was programmed to read a run record file written by PEST_HP; there are some important differences between the PEST_HP run record file and that produced by PEST and BEOPEST, this reflecting some of the unique features of its inversion algorithm, particularly its ability to conduct a line search along parameter upgrade vectors calculated using different values of the Marquardt lambda.

As well as reading a PEST_HP run record file, PWHISP_HP reads the PEST control file on which an inversion process is based, as well as the JCO file, the SEN file and the RES file produced by PEST_HP. After considering the contents of all of these files, it offers advice on whether PEST_HP performed according to its potential, and (if necessary) what features of its input dataset should be adjusted to improve its performance.

3.3 Groundwater Data Utilities

3.3.1 General

The roles played by members of the Groundwater Data Utility suite are now briefly described. In all, over 140 utilities comprise this suite. Only some of them are mentioned by name in the following description. For more details see their manual. This manual is in three parts. Part A provides an overview of the utilities, and presents specifications for the various file types that they employ. Part C documents utilities that were written specifically for use with MODFLOW-USG. Part B describes all other utilities.

Many of the Groundwater Data Utilities were written to serve as simulator pre- or post-processors. Thus they are likely to feature in batch files that are run by programs of the PEST or PEST++ suites as "the model". Each such utility contributes to the passage of information between PEST parameters and simulator inputs on the one hand, or between simulator outputs and field observations on the other hand.

3.3.2 Spatial/Temporal Interpolation of Model Outputs

Heads measured in one or a number of observation wells comprise a significant component of the calibration dataset of most groundwater models. Finite-different models such as MODFLOW and MODFLOW-USG calculate heads at the centres of cells which comprise their grids. These heads must be spatially interpolated to the sites of observation wells, and temporally interpolated to the times at which head measurements were made in these wells. For structured MODFLOW, this role is performed by MOD2OBS. Spatial interpolation is bilinear between cell centres, while temporal interpolation is linear between MODFLOW output times. For MODFLOW-USG this role is played by USGMOD2OBS and USGMOD2OBS1 (the latter can read both single and double precision binary heads files). Spatial interpolation from an unstructured grid to an arbitrary point in space is more difficult than for a structured grid. USGMOD2OBS and USGMOD2OBS1 request that the modeller supply his/her own interpolation factors. These can be generated automatically for structured and quadtree-refined MODFLOW-USG grids using the USGQUADFAC utility.

(USG)MOD2OBS(1) record the outcomes of their calculations in so-called "bore sample file" format. They expect borehole measurements to be supplied in this same format. A number of Groundwater Data Utilities undertake spatial interpolation of model-generated heads to a set of user-supplied locations, recording the outcomes of this interpolation procedure in bore sample file format. (As these utilities read no measurement data, temporal interpolation to measurement times is not required.) MOD2SMP and USGMOD2SMP perform these roles for MODFLOW and MODFLOW-USG respectively. DAR2SMP performs this role for FEFLOW. MOD2SMPDIFF undertakes subtraction as well as spatial interpolation, so that differences between heads calculated at two different locations can be recorded in bore sample file format. LAYDIFF performs a similar function.

MODFLOW-calculated fluxes can be extracted from the binary files in which they reside, and then recorded in bore sample file format, using BUD2SMP and BUD2SMP1 for structured MODFLOW, and USGBUD2SMP and USGBUD2SMP1 for MODFLOW-USG.

Because MT3DMS-suite programs employ the same protocol for storage of concentrations as MODFLOW does for heads, the same utility programs that extract heads (and drawdowns) from MODFLOW output files can extract concentrations from MT3DMS output files. Similarly, programs that extract heads from MODFLOW-USG output files can also extract concentrations from these files.

3.3.3 Processing of Borehole Time Series Data

A bore sample file can contain any type of data, sampled at regular or irregular intervals. It can hold field data or model-calculated data, including model-calculated counterparts to field data. A single bore sample file may hold data pertaining to many observation wells. A number of members of the Groundwater Data Utility suite process time-series data that is stored in bore sample file format. (Note that the TSPROC program from the Surface Water Utility suite provides comprehensive functionality for processing time series data stored in many formats, including bore sample file format.)

SMPCHEK checks the integrity of a bore sample file. SMP2SMP undertakes time-interpolation from the times that are featured in one bore sample file to the times that are featured in another bore sample file. SMPDIFF and SMPTREND assist in transient model calibration by taking differences between items within a bore sample file that pertain to the same well but that pertain to different times; it writes a new bore sample file based on these data differences. SMPCAL adjusts the contents of one bore sample file to accord with those of another. It can be used to calibrate logger data against sporadic manual water level measurements taken in wells in which water level recorders have been placed. SMP2HYD extracts data pertaining to a single well from a bore sample file, re-writing this data in tabular format suitable for use by any graphing package. SMP2INFO time-interpolates data from many wells to a user-specified time. It records the outcomes of this interpolation in a table along with the coordinates of respective wells. The file that it writes is suitable for use by a commercial contouring package.

3.3.4 Building and Manipulating a PEST Control File

In many model calibration contexts, the most difficult tasks encountered in assembling a PEST input dataset are those related to construction of the "observation groups" and "observation data" sections of a PEST control file, and the building of instruction files to read complementary data from model output files. As is extensively discussed by Doherty (2015), Doherty and Welter (2010) and White (2014), it is important that the objective function that is formulated to quantify the misfit between field data and its model-generated counterparts be designed to ensure that all types of information that are resident in the calibration dataset find expression, and thus have the capacity to inform estimable model parameters. At the same time, formulation of the required multi-component objective function must be such as to protect parameters from incurring calibration-induced bias as they adopt surrogate roles in order to compensate for model defects in promulgating a good fit between model outputs and the calibration dataset.

The PESTPREP, PESTPREP1 and PESTPREP2 utilities build a PEST control file based on the contents of bore sample files that contain field data on the one hand, and model-generated counterparts to field data on the other hand. Both of these datasets may be the outcomes of various types of processing, implemented using utilities described in previous subsections. Not only do these PESTPREP* utilities write and append information to the "observation groups" and "observation data" sections of a PEST control file; they also write instruction files to read bore sample files that contain model-generated data, naming observations in the process.

Once a PEST control file is constructed, the ADJOBS utility can be used to assign weights assignment to the members of different observation groups featured in this file. If desired, weights can be calculated as a function of observation values in group-specific ways.

3.3.5 MODFLOW Array Extraction and Manipulation

Arrays are the building blocks of the structured version of MODFLOW. A number of Groundwater Data Utilities read, write and manipulate the contents of MODFLOW-compatible integer and real arrays.

INT2REAL assists in implementation of zone-based parameterization of a structured grid model; elements of a real array are assigned values using elements of a corresponding integer array as keys. REAL2INT builds an integer array using real number intervals in a real array. TWOARRAY adds, subtracts, multiplies and divides corresponding elements of two different MODFLOW-compatible real arrays. LOGARRAY calculates a new array by taking the log (to base 10) of elements of an existing array.

MOD2ARRAY extracts real or integer arrays from a MODFLOW input file, recording these arrays in separate files. The contents of real and integer arrays can be written in tabular form using REAL2TAB. Conversely, the contents of tabular data files can be recorded in array format using TAB2INT and TAB2REAL. ARRAYOBS allows array data recorded in tabular form to be incorporated into a PEST control file as a set of observations that comprise all or part of a calibration dataset.

The SECTION utility undertakes bilinear interpolation from one or a number of arrays to an arbitrary line through the model domain; the outcomes of this interpolation are recorded in a format that is suitable for use by any plotting package. ARR2BORE undertakes bilinear interpolation from a single real array to user-specified points.

Some utilities rewrite all or part of the contents of MODFLOW-generated binary files in ASCII format so that these data can be viewed, manipulated and displayed. MANY2ONE extracts individual arrays from a MODFLOW binary heads/drawdown file or an MT3D binary concentration file. GETMULARR and GETMULARR1 extract multiple arrays from these files. ARRDET lists the specifications of arrays that are recorded in a structured MODFLOW binary output file.

3.3.6 MODFLOW-USG Data Extraction and Manipulation

MODFLOW-USG does not use arrays. Hydraulic properties, heads, concentrations and other MODFLOW-USG-generated datasets are better manipulated as tables of node-specific data than as two dimensional arrays. Some Groundwater Data Utilities perform similar functions to those described in the previous subsection, but operate on so-called "node data tables" rather than on arrays. However, where parameter pre-processing is required, the PLPROC package provides far greater flexibility than do individual utility programs. PLPROC is discussed below.

USGPROP2TAB1 and USGPROP2TAB2 exact hydraulic properties and other grid-related data from MODFLOW-USG input files; they re-write these data as node data tables. Programs USGBIN2TAB_H, USGBIN2TAB_H1 and USGBIN2TAB record heads, drawdowns or concentrations pertaining to a particular output time, as read from a MODFLOW-USG binary output file, in node data table format. The contents of a MODFLOW-USG binary output file can be listed using the USGARRDET utility.

3.3.7 Display of MODFLOW and MODFLOW-USG Data

None of the programs comprising the Groundwater Data Utility suite include any plotting or display functionality. However some of them include functionality to re-write MODFLOW, MT3DMS and MODFLOW-USG property/activity/head/drawdown/concentration and other data in a form that is easily read by GIS, plotting and display software.

Programs INT2MIF, REAL2MIF (for MODFLOW) and USGNDT2MIF (for MODFLOW-USG) write model data in "Mapinfo exchange format", an ASCII protocol that can be read by most GIS packages, including the public domain QGIS package. TAB2NDTF, USG2VTK, USG2VTK1 and PT2VTK record MODFLOW-USG datasets in so-called "legacy VTK" format that can be read by the public domain PARAVIEW visualization package. GRID2BLN, ZONE2BLN, ZONE2DXF, GRID2PT, REAL2SRF, SRF2REAL, ROTBLN, ROTDAT, ROTDXF and USGGRIDLAY facilitate data exchange between MODFLOW, MT3DMS and MODFLOW-USG on the one hand, and the popular SURFER contouring package on the other hand.

3.3.8 Pilot Points as a Parameterization Device

Over 20 members of the Groundwater Data Utility suite are dedicated to pilot point parameterization and regularisation. Utilities which implement pilot point parameterization include those which calculate interpolation factors from pilot points to a model grid, and those which use these factors to undertake interpolation to that grid once a set of values has been assigned to pilot points. These programs (which have now been largely superseded by PLPROC) include the PPK2FAC* suite of programs which calculate kriging factors that are used for interpolation of values assigned to two- and three-dimensional collections of pilot points to the grids/meshes of MODFLOW, FEFLOW and other models, and the FAC2* suite of programs that use these factors to conduct kriging as the model is actually run.

3.3.9 Regularisation for Pilot Points

Two members of the Groundwater Data Utilities suite add so-called “preferred difference” regularisation to a PEST control file based on kriging factors calculated by the PPK2FAC* suite of programs; these are the PPKREG and PPKREG1 utilities. “Preferred difference” regularisation defines homogeneity as a default parameter condition in prior information equations that implement Tikhonov regularisation.

While the idea that parametric heterogeneity should “earn its right to exist” in achieving a minimum error variance solution to an inverse problem makes sense, the number of prior information equations that are used to implement this form of regularisation can be high enough to constitute a significant computational burden when calibrating highly parameterized two- and three-dimensional models. “Preferred value” regularisation requires fewer prior information equations than preferred-difference regularisation. Furthermore, if these are applied to multiplier parameters acting on an underlying zone-based or uniform parameterization device, then a “preferred difference” regularisation strategy is effectively implemented at the same time. The PPCOV, PPCOV3D, PPCOV_SVA and PPCOV3D_SVA utilities facilitate this kind of regularisation. They build a covariance matrix based on a user-specified variogram for use with prior information equations which specify preferred values for pilot points. The variogram specified for the last two of these utilities can have characteristics (including anisotropy) that vary on a pilot point by pilot point basis. This allows a high degree of flexibility in defining acceptable ways for geologically realistic heterogeneity to arise within a model domain. If desired, variogram ranges can vary in proportion to local pilot point density; the MKPPSTAT and MKPPSTAT3D programs write PPCOV_SVA and PPCOV3D_SVA input files which support this type of local variogram definition.

The GENREG utility provides further regularisation options – for pilot points and for other parameterization devices.

3.3.10 Some other Programs

FIELDGEN populates a structured MODFLOW real array with a multiGaussian stochastic field based on a user-supplied variogram. Optionally, this stochastic field can be conditioned by hydraulic property measurements made within its domain. PPSAMP calculates values for pilot points such that real array values kriged from these points fit a (stochastic) real array in a minimized least squares sense. CONC2ELEV and ELEV2CONC assist in calibration of multi-layered SEAWAT models. The former calculates the position of a freshwater/saltwater interface based on a three-dimensional array of concentrations; the latter performs the reverse operation.

3.4 PLPROC

“PLPROC” stands for “parameter list processor”. It was written shortly after the release of MODFLOW-USG to complement the flexible grid design afforded by this simulator with flexibility in model parameterization. However use of PLPROC is not restricted to parameterization of MODFLOW-USG. Because it uses “embedded functions” in templates of model input files to introduce hydraulic property data to these files, it can be used in conjunction with any model, regardless of the nature of its grid or mesh, and regardless of data formatting protocols employed by model input files. PLPROC is an integral component of the FEPEST interface with FEFLOW.

Much of the parameterization functionality of PLPROC focusses on pilot points. However zone based parameterization is also supported, either on its own or in combination with pilot points. Interpolation from pilot points to a model grid or mesh can be implemented using kriging, radial basis functions or inverse power of distance. In the former two cases, the variables which control spatial interpolation can vary in space according to local pilot point density.

PLPROC functionality is accessed through a series of function calls that collectively constitute a native programming language. This provides a high degree of flexibility in designing innovative parameterization schemes. Parameter and model property lists can undergo arbitrary mathematical manipulation defined by user-supplied equations which apply to all or part of a parameter list. Interpolation from pilot points to a grid or mesh can exhibit spatially varying anisotropy, as may be required in those parts of a model domain that represent meandering alluvial channels, or local regions of directional geological structure.

3.5 Surface Water Utilities

Prior to 2001 a small suite of surface water utilities had been written to expedite the use of PEST in conjunction with surface water and land use models. However it was realized that the demands of surface water model calibration require a more coherent response than this. In particular, much depends on innovative formulation of an objective function that, on the one hand, defines model-to-measurement misfit and, on the other hand, ensures that information within a calibration dataset that pertains to different groups of parameters actually gets to inform those parameters.

“TSPROC” stands for “Time Series PROCessor”. It reads an arbitrary number of time series datasets (representing flow, water constituent concentrations, etc) from a variety of file types. It undertakes temporal interpolation of model-calculated quantities to the times at which measurements were made in the field to produce two complementary time series – a field measurement dataset and its model-calculated counterpart. These two time series can then be processed in a number of different ways, with the same processing applied to both. The outcomes of these different types of processing can then comprise the components of a multi-component objective function whose value is reduced as the model is calibrated. Weights associated with processed measurements can be calculated as functions of the measurements themselves. Time series can undergo mathematical manipulation using equations of arbitrary complexity.

Using TSPROC, the objective function employed in calibration of a model may include any or all of the following components:

- flows or constituent concentrations (mathematically transformed according to a user’s specifications);
- flow volumes or constituent masses over a sequence of discrete events or over a sequence of regular, contiguous time spans;
- flow or constituent exceedance durations;

- flow or constituent concentration statistics;
- high and low pass filtered flows;
- extracted approximations to baseflow.

Different aspects of TSPROC functionality are activated through a series of processing blocks in which variables which control TSPROC processing are supplied through keywords. (In retrospect, it may have been better to employ a programming language like that used by PLPROC).

In expediting calibration of surface water models, TSPROC serves two purposes. It acts as a model post-processor. It also acts as a PEST input file generator. Once a modeller has built a TSPROC input file containing blocks that specify the sequence of data and model post-processing tasks that he/she requires, he/she can then add another block to this file that instructs TSPROC to write a PEST input dataset based on the multi—component objective defined through this processing sequence. TSPROC also writes an instruction file to read the output file that it writes in acting as the model post-processor.

4. PEST++: A Short History

4.1 Rationale for Development of PEST++

It is apparent from the brief description of PEST provided in chapter 2 of this document, that the breadth of its functionality is large. Whenever, over the years since its inception, it became obvious that environmental model calibration and uncertainty analysis required more sophistication than was included in the current version of PEST, or that its existing inversion algorithm needed improvement, more code was added accordingly. Once PEST was placed in the public domain, much of this had to be done on a financial shoestring. While PEST remained remarkably bug-free, parts of its coding became more and more tortuous. This made the addition of further enhancements more and more difficult. It became obvious that it needed to be re-written.

In 2009 development of PEST++ began. This was funded by the USGS Great Lakes Restoration Initiative. As related by Welter et al (2010), its purpose was to:

1. Lower the barriers of entry for new users of parameter estimation software;
2. Provide to the modelling community access to efficient parameter estimation tools and algorithms for implementing highly parameterized inversion; and
3. Employ an object-oriented framework to support future cooperative development by multiple programmers.

The original version of the PEST++ suite included only the PESTPP program. Like PEST, this program undertakes highly parameterized inversion. The first version of PESTPP did not include all of the functionality that was available in the version of PEST which prevailed at that time. (Nor does it now; however, it includes functionality which the present version of PEST does not possess.) However its original specifications included those aspects of PEST functionality that were used most often.

All programs comprising the PEST++ suite are written in C++. Internally, programming is object-oriented and designed for team-based code development. It has been written in the Microsoft Visual Studio environment. Source code and Visual Studio project files are freely available.

4.2 Parallel Run Management

An important specification of PEST++ was that its parallel run manager be separated from the inversion engine. Because of this, the run manager can be used by programs other than those of the PEST++ suite that require non-intrusive parallel run management. It was specified that the run manager adopt similar principles to the BEOPEST run manager, specifically that it communicate with “smart workers” using the TCP/IP protocol, and that these workers employ template and instruction files to communicate with a model. Its original run manager was named “GENIE” (Muffels et al, 2012). Also written in C++, GENIE made run management functionality available to programs written in C, C++ and FORTRAN.

4.3 Relationship between PEST and PEST++

4.3.1 Functionality

As stated above, it was not intended that PESTPP replicate all of the functionality of PEST. However it was intended that some commonly implemented tasks be done better by PESTPP than by PEST, and that default values be available for all control variables, so that ease of use could be enhanced. Particular emphasis was placed on facilitating SVD-assisted inversion.

When undertaking SVD-assisted inversion using PEST, a user must implement the following steps:

1. Calculate a Jacobian matrix to complement a PEST control file that defines the inverse problem using native model parameters;
2. Run the SVDAPREP utility to build a complete input dataset for SVD-assisted parameter estimation in which PEST estimates the values of super parameters;
3. Run PEST using this new super parameter dataset;
4. On completion of the parameter estimation process, use the PARREP utility to build a new PEST control file in which initial native parameter values are replaced by estimated native parameter values.

While SVD-assisted inversion can reduce the number of runs required for highly parameterized inversion considerably, it encounters problems where the relationship between parameters and model outputs is highly nonlinear. In some calibration contexts, progress of SVD-assisted inversion may falter, or cease altogether, before an acceptable fit is attained between model outputs and a calibration dataset. These problems can often be overcome by periodical re-definition of super parameters as the values of native parameters change through the inversion process. If using PEST, rebuilding of a PEST input dataset for SVD-assisted inversion must be done manually. That is, the existing super parameter estimation process must be halted; then a new Jacobian matrix must be calculated using updated base parameters. SVDAPREP must then be re-run to re-build a super parameter estimation dataset, then PEST execution must be re-initiated to advance the SVD-assisted inversion process.

A major design consideration for PESTPP was that SVD-assisted inversion be made easier. In PEST++, SVD-assisted inversion takes place “behind the scenes”. If requested to do so, PEST++ calculates a base Jacobian matrix and then automatically estimates super parameters for a few iterations without the need to run SVDAPREP; it does not construct a PEST control file which features super parameters. The modeler simply nominates how many iterations of SVD-assisted inversion must take place before a base Jacobian matrix is re-computed so that super-parameters can be re-defined. No other human intervention is required.

(It is worth pointing out that while the benefits of automating SVD-assisted inversion are obvious, separation of super parameter estimation from base parameter estimation can sometimes be beneficial when exerting calibration constraints on random parameter fields using the null space Monte Carlo methodology.)

The original version of PESTPP introduced the PROPACK package (Larsen, 2001) as an alternative to singular value decomposition as a solution methodology for ill-posed inverse problems. This package is as fast as the LSQR methodology used by PEST, but is numerically superior.

4.3.2 File Protocols

An important specification for programs of the PEST++ suite in general, and PESTPP in particular, was that of partial file compatibility with PEST so that PESTPP and PEST could be used interchangeably. This required that PEST read a PEST control file, while ignoring variables in this file which it does not use. PEST++ control variables are placed in a PEST file on lines that begin with the "++" string. In contrast to PEST control variables which are identified by their positions in a PEST control file, PEST++ control variables are recognized by a variable-specific keyword. Keywords can be supplied in any order. Where a keyword is not supplied, PEST++ programs supply a default value for the associated control variable.

Some other input files are also compatible between PEST and programs of the PEST++ suite. PEST++ programs that undertake uncertainty analysis are able to read parameter uncertainty files and covariance matrices files that accord with PEST specifications for these file types.

There are two output file types that were specified to be identical between PEST and programs of the PEST++ suite. These are the JCO and PAR files. The first is a binary file which holds the Jacobian matrix; the second is an ASCII file which holds best parameters achieved up to any point in the inversion process.

4.4 PEST++ Version 3

Despite undergoing continuous improvements since its first release, it was only in 2015 that a new version of the PEST++ manual was released; see Welter et al (2015). Enhancements to PEST++ that collectively comprised version 3 of this suite are now briefly described.

4.4.1 New Run Manager

A new run manager named YAMR (for "Yet Another Run Manager") was introduced to PEST++. Since its release with version 3 of PEST++, YAMR (rather than GENIE) has been the focus of run manager development.

A change in terminology was introduced with YAMR; the "master" become the "manager" and a slave became a "worker". In similar fashion to BEOPEST, the same executable program plays both of these roles; information supplied on its command line informs a member of the PEST++ suite the role that is must play on any occasion that it is run. When run as a worker, A PEST++ program is threaded. This allows it to maintain communications with its manager even when it is supervising a model run. It can thus respond to status queries from the manager; if requested by the manager, it can terminate the model run which it is presently supervising and initiate another in its place.

4.4.2 Improved Inversion Capabilities

Based on experience gained in using versions 1 and 2 of PESTPP, version 3 of PESTPP altered the way in which values for the Marquardt lambda are selected for calculating parameter upgrades.

PESTPP's implementation of Tikhonov regularisation was also upgraded. Restart capabilities (missing in versions 1 and 2 of PESTPP) were introduced. A number of new output options were provided, some of which are not available in PEST. One such option is the recording of CSV files that can be readily imported into packages such as Microsoft EXCEL to facilitate graphing and analysis of inversion progress and results.

Version 3 of PESTPP offered users the choice of three methods for solution of an inverse problem, all of which are closely related to singular value decomposition, and all of which can happily accommodate inverse problem ill-posedness. One is the PROPACK package inherited from version 1 of PESTPP; another uses the Jacobi module from the public domain EIGEN library (see http://eigen.tuxfamily.org); a third (suitable for very highly parameterized inverse problems) employs randomized singular value decomposition functionality provided by the REDSVD library (see https://github.com/cequencer/redsvd).

4.4.3 Linear Analysis

Version 3 and above of PESTPP undertakes basic linear analysis (also known as FOSM - first order second moment) on completion of an inversion process. It calculates posterior parameter uncertainties and, optionally, a full posterior parameter covariance matrix. These are computed using the same algorithm as that used by the PEST PREDUNC* utility programs. If requested, PESTPP calculates the posterior uncertainty of a prediction. This prediction must be provided as an "observation" in the PEST control file; as such, it should be ascribed a weight of zero.

Calculation of posterior parameter and predictive uncertainties requires knowledge of prior parameter uncertainties. PESTPP (and other programs of the PEST++ suite) can calculate these from parameter bounds supplied in a PEST control file. Alternatively, these programs can read them from a user-prepared parameter uncertainty file; this may, or may not, reference one or a number of parameter covariance matrix files.

4.4.4 Global Optimisation using Differential Evolution

Inversion methods based on differentials of model outputs with respect to adjustable parameters (these are often referred to as "gradient methods") can accommodate large numbers of parameters, and can include complex regularisation algorithms. However the performance of these methods can suffer if a model exhibits bad numerical behaviour, and/or if the relationship between its outputs and its parameters is extremely nonlinear, especially if this promulgates local objective function minima. So-called "global" optimisation methods can (in theory) accommodate these types of model behaviour. Furthermore, because global methods perform optimization rather than inversion, they can also be used for decision optimization. Under these circumstances, the model inputs that they adjust in order to minimize an objective function are referred to as "decision-variables", rather than as parameters. Unfortunately, however, global methods generally require a much larger number of model runs than do gradient methods to minimize an objective function. Furthermore, the model run burden incurred by use of these methods generally increases dramatically as the number of adjustable parameters increases. Nor can global methods assimilate expert knowledge embedded in regularisation constraints with the same alacrity as gradient methods.

If requested to do so, PESTPP implements global optimisation using the differential evolution (DE) method. This method is described in Storn and Price (1995; 1997). The DE method is relatively simple to program, is easy to parallelize, and (according to the literature) works well in a variety of optimization circumstances. It is activated by provision of a pertinent keyword in the PEST control file read by PESTPP. One of the attractions of the DE method is the small number of control variables that is required for its implementation. PESTPP offers defaults for all of these.

4.4.5 Global Sensitivity Analysis

A new program named GSA was added to version 3 of the PEST++ suite. "GSA" stands for "Global Sensitivity Analysis"; GSA has been renamed to "PESTPP-GSA". PESTPP-GSA supports two kinds of global sensitivity analysis, namely the methods of Morris and Sobol; see Morris (1991), Sobol (1993) and Saltelli et al (2004) for a description of these methods. This represented a significant addition to the suite of model-value-adding functionality available through the PEST and PEST++ suites. Global sensitivity analysis provides considerable insights not just into the roles played by the parameters of a model, but also into the model-simulated processes with which these parameters are associated.

4.5 Version 4 Enhancements of the PEST++ Suite

4.5.1 General

Version 4 of the PEST++ suite was released in 2018, together with a comprehensive manual for the suite (White et al, 2018b). Previous "manuals" comprised USGS reports; they all assumed a high level of user familiarity with PEST data input protocols. The new manual, however, made no such assumptions, and hence comprised complete, stand-alone, documentation for the suite. It also included documentation for members of the suite that had been added since the time of writing of Welter et al (2015). These members are now briefly described.

4.5.2 Optimisation under Uncertainty

PESTPP-OPT implements management optimization under a regime of parameter uncertainty. Parameters that are featured in a PEST control file are categorized as decision-variables or model parameters. The latter (i.e. model parameters) normally represent the properties of a simulated system. As such, they are assumed to be uncertain, despite the fact that they are not varied during the PEST++ optimization process. However their uncertainties are assumed to have been reduced from their prior uncertainties by a calibration process that has already taken place. The calibration dataset is included in the PEST control file, while prior parameter uncertainties are supplied by the user. Post-calibration parameter uncertainties are computed using a linearized form of Bayes equation.

PESTPP-OPT adjusts decision-variables in order to minimize a model-generated quantity (normally a management cost) subject to user-specified constraints on other model outputs. This is achieved using a sequential linear programming algorithm supplied in the open-source CLP optimization library (Forrest et al 2016). Because they are sensitive to model parameters (which are uncertain), the model outputs to which constraints are applied are also uncertain. Like the parameters to which they are sensitive, the uncertainties associated with these constraint outputs can be calculated using standard linear methods. PESTPP-OPT uses a calibration Jacobian matrix (which it can optionally calculate itself) to estimate these uncertainties. If requested by the user, this Jacobian matrix can be intermittently updated if its elements are likely to change as decision variables change through the optimisation process.

In order to accommodate model output uncertainty, post-calibration confidence intervals associated with model outputs to which constraints are applied are added or subtracted to these model outputs before imposition of these constraints in accordance with a user's level of risk aversion or risk tolerance. See White et al (2018) for further details.

4.5.3 An Iterative Ensemble Smoother

PESTPP_IES implements the iterative ensemble smoother described in detail by Chen and Oliver (2013). It adjusts a suite of random parameter fields simultaneously rather than just a single parameter field. Initially, these random parameter fields are samples from the prior parameter probability distribution. Each of them is adjusted until it promulgates a good fit between model outputs and members of the calibration dataset. At the end of the iterative process, the adjusted parameter fields can therefore be considered as samples of the posterior probability distribution of model parameters.

In many modelling contexts, adjustment of these parameter fields is numerically very efficient. The same model runs that are used to calculate model outputs using adjusted parameter fields are used to calculate an approximate Jacobian matrix on which to base the next iteration of parameter field adjustment. In benign circumstances only a few iterations may be required for attainment of a good fit between the calibration dataset and model outputs calculated using all of these parameter fields. Each of these iterations requires only as many model runs as there are random realizations which comprise the ensemble. In most practical cases, this is far fewer than the number of parameters which comprise a realization. Theoretically the number of realizations need not be any larger than the number of dimensions of the solution space of the inverse problem, regardless of the number of parameters that each field contains. Hence, if desired, each parameter field can cite thousands, or even hundreds of thousands of parameters; parameters can therefore be numerous enough to represent hydraulic property heterogeneity at the model cell level. See White (2018) for details.

4.5.4 Runs for any Reason

PESTPP-SWP runs a model using arbitrary sets of parameter values supplied in a CSV file. Model output sets corresponding to these parameter sets are recorded in another CSV file. Model runs can be undertaken sequentially or in parallel.

4.5.5 Universal Worker

PESTPP-WRK is a “smart worker” that can be used in conjunction with any member of the PEST++ suite. However its use is not restricted to members of this suite. It can be used in conjunction with any program that uses the PANTHER run manager, whether or not that program reads a PEST control file. In complementing PANTHER, PESTPP-WRK helps to facilitate the production of model-value-adding software by third party programmers who, like the authors of the present document, realize the important role that this software plays in supporting models that support decisions.

5. PyEMU

5.1 Introduction

PyEMU is the name given to a set of Python modules that programmatically interact with members of the PEST and PEST++ suites. It employs the same nomenclature as that used by PEST and PEST++ programs. At the same time, it replicates much of the functionality available through PEST utility support software described earlier in this manual, while providing additional capabilities (including plotting and display) that are not available through these stand-alone programs. Its powerful PEST/PEST++ support functionality is implemented through a framework that is easily understood by Python programmers.

PyEMU makes heavy use of the scientific Python ecosystem including the Numpy, Pandas, and Matplotlib libraries. It makes the popular FloPy library (Bakker et al, 2018) an explicit dependency, this supporting direct linkage between the PyEMU and FloPy packages. Some of the capabilities that are supported by this linkage are described later in this section.

5.2 Rationale for Development

The initial version of PyEMU was developed to prototype and experiment with model-related linear analysis. It replicated the capabilities of the PEST PREDUNC and PREDVAR suites, while providing easier access to these capabilities through a common Python programming framework. Since that time, PyEMU has grown into an extensive library of Python modules that expedite PEST/PEST++ pre/post-processing, while also implementing many PEST-independent tasks such as Monte Carlo and GLUE analyses. (See Beven et al (1992), Stendinger et al (2008), and references cited therein, for a description of GLUE.) PyEMU also supports visualization of the results of these analyses. In this way it provides a reproducible, scripting interface for PEST/PEST++ analyses while also serving as a “sand box” for prototyping new algorithms (a task for which Python is well suited).

PyEMU adopts a pythonic approach to ease of use. It is invoked with minimal user input by relying on default behavior wherever possible. For example, if a user does not pass a prior parameter covariance matrix to a linear analysis class, PyEMU constructs a diagonal prior parameter covariance matrix based on parameter bounds supplied in a PEST control file. This aspect of its design promotes rapid introductory use at the same time as it offers more complex functionality required by experienced users.

5.3 PEST/PEST++ and PyEMU Interoperability

PyEMU was designed to interface smoothly with programs comprising the PEST and PEST++ suites. It can read and write most of the types of file that are used by members of these suites. These include (but are not limited to) the following:

- PEST control files;
- binary and ASCII matrix files;
- parameter value (i.e. PAR) files;
- residual (i.e. RES) files;
- Jacobian matrix (i.e. JCO) files; and
- parameter uncertainty (i.e. UNC) files.

5.4 Some Examples

5.4.1 Control File Handling using the *Pst* Class

PyEMU offers an extensively-tested class for creation and manipulation of PEST control files. The PyEMU *Pst* class has attributes that pertain to individual sections featured in a PEST control file. For example, the "parameter data" section of a PEST control file is accessible as *Pst.parameter_data*, while the "control data" section is accessible as *Pst.control_data*. Furthermore, Python variable names belonging to the *Pst* class are compatible with the names of variables that appear in pertinent sections of the PEST control file.

Along with the *Pst* class, PyEMU offers several methods to create a new PEST control file. These range from a generic *Pst* instance with dummy parameter and observation names, to creation of a PEST control file based on existing template and instruction files.

The *Pst* class also includes methods that write stand-alone LaTeX summary tables of parameters and observations appearing in a PEST control file.

5.4.2 Linear Algebra using the *Matrix* and *Cov* Classes

The *Matrix* and (derived) *Cov* classes available through PyEMU are designed to facilitate implementation of model-pertinent linear algebra. Through object oriented programming, the *Matrix* class overloads mathematical operators, this allowing linear algebra equations to be quickly coded in PyEMU without a user having to concern him/herself with row and column alignment; the *Matrix* class automatically aligns these elements based on row and column names. As an example, suppose that a user wishes to plot the singular value spectrum of the so-called "normal matrix" calculated by PEST or PEST++. This can be accomplished using the following code fragment.

```
>>>pst = pyemu.Pst("pest.pst")
>>>jco = pyemu.Jco.from_binary("pest.jco")
>>>oc = pyemu.Cov.from_observation_data(pst)
>>>xtqx = jco.T * oc**0.5 * jco
>>>plt.plot(xtqx.s)
```

The ease with which complex matrix operations can be accomplished with a few lines of code is immediately obvious.

5.4.3 Linear Analysis (FOSM) Capabilities Using the *Schur* and *ErrVar* Classes

PyEMU exposes two classes for implementation of linear algebra. These are the *Schur* and *ErrVar* classes. These replicate the behavior of the PEST PREDUNC and PREDVAR suites respectively. The *Schur* class implements the *Schur* compliment equation for conditional covariance propagation (the equation used by members of the PREDUNC suite). Like the PREDUNC suite of utility programs, the *Schur* class provides functionality for parameter and predictive uncertainty evaluation, calculating the worth of data, and evaluating the contributions made to the uncertainty of an arbitrary prediction by different parameters or groups of parameters. Similar to the PEST PREDVAR utility suite, the *ErrVar* class provides related functionality based on subspace error variance analysis.

5.4.4 Monte Carlo and GLUE Analysis

The *MonteCarlo* and associated *ParameterEnsemble* and *ObservationEnsemble* classes available through PyEMU facilitate preparation for Monte Carlo analysis within the PEST framework. The *ParameterEnsemble* class supports uniform and multivariate Gaussian draws, while respecting parameter transformation status and parameter bounds provided in a PEST control file. The *ParameterEnsemble* and *ObservationEnsemble* classes interact with the PESTPP-SWP program, allowing a user to quickly generate multiple realizations of parameters to thereby construct a parameter ensemble. Model outputs can be calculated using all members of the ensemble by conducting model runs in parallel, and then by conditioning model outputs using GLUE-type metrics pertaining to model-to-measurement misfit.

The following PyEMU code example demonstrates how a posterior parameter probability distribution (defined by best-fit parameter values and a posterior parameter covariance matrix calculated by the *Schur* class) can be used to draw 10,000 parameter realizations; these are then saved in a PESTPP-SWP compatible CSV file.

```
>>>pst = pyemu.Pst("pest.pst")
>>>pst.parrep("pest.par")
>>>sc = pyemu.Schur(jco="pest.jco")
>>> mc = pyemu.MonteCarlo(pst=pst,parcov=sc.posterior_parameter)
>>>mc.draw(10000)
>>>mc.parensemble.to_csv("sweep_in.csv")
```

5.4.5 Visualization Using the PyEMU *plot* Module

The *plot* module available through PyEMU generates many types of plots that are commonly employed for visualization of the results of parameter estimation and parameter/predictive uncertainty analysis. These include the following:

- deterministic and stochastic line-of-equality plots;
- multi-component objective function pie charts;
- multiple ensemble histograms;
- linear-analysis-implied Gaussian distributions;
- objective function progress by iteration;
- plots of observation vs modelled time series plots;
- stacked identifiability bar charts;
- prior Gaussian distributions implied by parameter bounds.

In accordance with the design philosophy of PyEMU, this plotting functionality is accessible with minimal user input; however fine-scale plotting detail can be designated using optional keyword function arguments.

5.4.6 The g*eostats* Module

The *geostats* module is a pure-Python implementation of geostatistical methods which include ordinary kriging and parameter covariance matrix construction. The *geostats* module supports nested geostatistical structures; it can read and write PEST-compatible structure files. In doing so it replicates the functionality of the PPK2FAC and FAC2REAL programs provided with the PEST Groundwater Data Utilities suite. It includes functionality for reading and writing files in GSLIB (Deutsch and Journel, 1998) and SGEMS (Remy et al, 2011) formats.

5.4.7 The *gw_utils* Module

The *gw_utils* module offers functionality for setting up template and instruction files associated with the MODFLOW family of simulators (including MT3D-USGS). Much of this functionality is used within the *PstFromFlopyHelper* class (see below).

5.4.8 The *pp_utils* Module

The *pp_utils* module assists with pilot points parameterization of a model.

5.4.9 The *PstFromFlopyHelper* Class

At the time of writing, recent PyEMU developments have been focused on decreasing the amount of time required for construction of a PEST++ interface with an existing model. The *PstFromFlopyHelper* class combines the functionality of FloPy with that of PyEMU to rapidly construct a sophisticated PEST++ interface for any FloPy-compatible MODFLOW family model. Setup is such that all parameters act as multipliers on base model input arrays. This allows users to base spatial model parameterization on complex mixtures of zones, pilot points, Karhuenen-Loève covariance eigenvector multipliers, and grid-scale parameters for any array-based model data type. Complex parameterization schemes can be constructed, and reproduced, through a suite of simple function calls. The helper also includes functionality to construct spatial and temporal list-based input multipliers. All MODFLOW list-based boundary condition datasets are supported.

On the model output side, the *PstFromFlopyHelper* class includes functionality for extracting the model-generated counterparts to observations from a variety of binary and ASCII MODFLOW output files. The *PstFromFlopyHelper* class writes the forward model run script, the prior parameter covariance matrix (based on geostatistical concepts where appropriate), and a PEST control file. Following the running of a model, users can immediately commence parameter estimation or Monte Carlo based uncertainty analysis using files created by this class and functionality available through PyEMU.

The *PstFromFlopyHelper* class is most impressive when used in conjunction with PESTPP-IES. The helper can be used to quickly generate a model-to-PEST++ interface with hundreds of thousands of cell-based parameters; decisions related to parameterization density are thereby avoided. In this context of extremely highly parameterized inversion, a user can generate high-dimensional parameter ensembles using the memory-efficient *PstFromFlopyHelper.draw* method.

6. References

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J. and Fienen, M. N., 2016, Scripting MODFLOW model development using Python and FloPy. *Groundwater*, 54, 733–739, doi:10.1111/gwat.12413.

Bakker, M., Post, V., Langevin, C.D., Hughes, J.D., White, J.T., Starn, J.J., and Fienen, M.N., 2018. FloPy v3.2.9: U.S. Geological Survey Software Release, 27 February 2018, http://dx.doi.org/10.5066/F7BK19FH

Bayer, P., Bürger, C.M. and Finkel, M., 2008. Computationally efficient stochastic optimization using multiple realizations. *Adv. Water Resour.* 31(2), 399-417. doi:10.1016/j.advwatres.2007.09.004.

Bayer, P., de Paley, M. and Bürger, C.M., 2010. Optimization of high-reliability-based hydrological design problems by robust automatic sampling of critical model realizations. *Water Resour. Res.* 46, doi:10.1029/2009WR008081.

Beven, K. and Binley, A., 1992. The future of distributed models: model calibration and uncertainty prediction. *Hydrological Processes* 6 (3): 279–298.

Chen, Y. and Oliver, D. S., 2013. Levenberg–Marquardt forms of the iterative ensemble smoother for efficient history matching and uncertainty quantification. *Computational Geosciences*, 17 (4), 689–703.

Certes, C., and de Marsily, G., 1991, Application of the pilot-points method to the identification of aquifer transmissivities: *Advances in Water Resources*, 14 (5), 284–300, doi:10.1016/0309-1708(91)90040-U.

De Groot-Hedlin, C. and Constable, S., 1990. Occam's inversion to generate smooth, two-dimensional models from magnetotelluric data. *Geophysics*, 55 (12), 1613-1624.

Deutsch, C. and Journel, A.G., 1998. GSLIB: geostatistical software library user's guide. Oxford University Press, New York.

Doherty, J., 2015. Calibration and uncertainty analysis for complex environmental models. Published by Watermark Numerical Computing, Brisbane, Australia. 227pp. ISBN: 978-0-9943786-0-6. Downloadable from www.pesthomepage.org.

Doherty, J., 2018a. Manual for PEST: Model-Independent Parameter Estimation. Part 1: PEST, SENSAN and Global Optimisers. Watermark Numerical Computing, Brisbane, Australia. Downloadable from www.pesthomepage.org.

Doherty, J., 2018b. Manual for PEST: Model-Independent Parameter Estimation. Part 2: PEST Utility Support Software. Watermark Numerical Computing, Brisbane, Australia. Downloadable from www.pesthomepage.org.

Doherty, J. and Christensen, S., 2011. Use of paired simple and complex models in reducing predictive bias and quantifying uncertainty. *Water Resourc. Res* 47, W12534, doi:10.1029/2011WR010763.

Doherty, J. and Johnston, J.M., 2003. Methodologies for calibration and predictive analysis of a watershed model, *J. American Water Resources Association*, 39(2), 251-265.

Doherty, J. and Simmons, C.T., 2013. Groundwater modelling in decision support: reflections on a unified conceptual framework. *Hydrogeology Journal* 21, 1531–1537.

Doherty, J. and Welter, D., 2010, A short exploration of structural noise, *Water Resour. Res*., 46, W05525, doi:10.1029/2009WR008377.

Doherty, J. and Vogwill, R., 2015. Models, Decision-Making and Science. In *Solving the Groundwater Challenges of the 21st Century*. Vogwill, R. editor. CRC Press.

Duan, Q., Gupta, V.K. and Sorooshian, S., 1993. A shuffled complex evolution approach for effective and efficient global optimization," *Journal of Optimization Theory and Its Applications*, 76(3): 501-521.

Forrest, J., Nuez, D. d.l., Lougee-Heimer, R., 2016. CLP: COIN-OR Linear Programming Solver. https://projects.coin-or.org/Clp.

Gallagher, M.R. and Doherty, J., 2006. Parameter Estimation and Uncertainty Analysis for a Watershed Model, *Environmental Modelling and Software*, 22, 1000-1020.

Gallagher, M. and Doherty, J., 2007. Predictive error analysis for a water resource management model. *Journal of Hydrology*, 34(3-4), 513-533.

Hansen, N. and A. Ostermeier, 2001. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.,* 9, 159-195

Hansen, N., Muller, S.D. and Koumoutsakos, P., 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol. Comput.,* 9, 159-195.

Hunt, R.J., Luchette, J., Shreuder, W.A., Rumbaugh, J., Doherty, J., Tonkin, M.J. and Rumbaugh, D., 2010. Using the cloud to replenish parched groundwater modeling efforts. Rapid Communication for *Ground Water,* doi: 10.1111/j.1745-6584.2010.00699

Karanovic, M., Muffels, C.T, Tonkin, M.J. and Hunt, R.J., 2012. Approaches in Highly Parameterized Inversion: PESTCommander, a Graphical User Interface for File and Run Management across Networks. USGS Techniques and Methods, Book7, Ch 8.

Kavetski, D., Kuczera, G. and Franks, S.W., 2006. Calibration of conceptual hydrological models revisited: 1. overcoming numerical artifacts. *J. Hydrol.,* 320, 173-186.

Larsen, R.M., 2001. Combining implicit restart and partial reorthogonalization in Lanczos bidiagnalization, SCCM, Stanford University, April 2001. PROPACK is downloadable from: http://sun.stanford.edu/~rmunk/PROPACK/

Luchette, J., Nelson, G.K., McLane, C.F. and Cecan, L.I., 2009. Unlimited virtual computing capacity using the cloud for automated parameter estimation. In *Proceedings of the 1st PEST Conference*, Potomac, Maryland, 1–3 November.

Moore, C. and Doherty, J., 2005. The role of the calibration process in reducing model predictive error. *Water Resources Research*. 41 (5), W05050.

Moore, C. and Doherty, J., 2006. The cost of uniqueness in groundwater model calibration. *Advances in Water Resources*. 29 (4), 605–623.

Morris, M.D. 1991. "Factorial Sampling Plans for Preliminary Computational Experiments". Technometrics 33(2), 161-174

Muffels, T.M., Schreüder, W.A., Doherty, J.E., Karanovic, M, Tonkin, M.J., Hunt, R.J. and Welter, D.E., 2012. Approaches to Highly Parameterized Inversion: Genie, a General Model-Independent TCP/IP Run Manager. United. States Geological Survey, Techniques and Methods, Book 7, Section C6.

Nott, D.J., Marshall, L. and Brown, J., 2012. Generalized likelihood uncertainty estimation (GLUE) and approximate Bayesian computation: what's the connection? *Water. Resour. Res.*, 48, W12602, doi:10.1029/2011WR011128.

Paige, C.C., and Saunders, M.A., 1982a. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM T Math Software*, 8 (1), 43–71.

Paige, C.C., and Saunders, M.A., 1982b. Algorithm 583 LSQR: Sparse linear equations and least squares problems. *ACM T Math Software,* 8 (2), 195–209.

Remy, N., Boucher, A. and Jianbing, W., 2011. Applied Geostatistics with SGeMS. A User's Guide. ISBN: 9781107403246.

Sadegh, M. and Vrugt, J.A., 2013. Bridging the gap between GLUE and formal statistical approaches: approximate Bayesian computation. *Hydrol. Earth Syst. Sci.,* 17, 4831-4850.

Saltelli, A., Tarantola, S., Campologno, F., and Ratto, M., 2004. Sensitivity Analysis in Practice – A Guide to Assessing Scientific Models. West Sussex, England, John Wiley and Sons Ltd., 219p.

Schumacher, J., Hayley, K., Boutin,L.C. and White, E., 2018. PPAPI: a program for groundwater modelling tasks in distributed parallel computing environments. *Groundwater*, 56 (2), 248-250.

Stedinger, J. R., Vogel, R M., Lee, S U, and Batchelder, R., 2008. Appraisal of the generalized likelihood uncertainty estimation (GLUE) method. *Water Resources Research* 44 (12). doi:10.1029/2008WR006822

Sobol, I.M., 1993. Sensitivity Estimates for Nonlinear Mathematical Models, *Mathematical Modeling and Computation*, 1(4), 407-414.

Storn, R., and Price, K., 1995. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech Report, International Computer Science Institute, Berkeley.

Storn, R. and Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11, 341-359.

Tonkin, M. and Doherty, J., 2005. A hybrid regularised inversion methodology for highly parameterised models. *Water Resources Research*. 41, W10412, doi:10.1029/2005WR003995.

Tonkin M., J. and Doherty, J., 2009. Calibration-constrained Monte Carlo analysis of highly parameterized models using subspace techniques, *Water Resour. Res*., 45, W00B10, doi:10.1029/2007WR006678.

Vecchia, A.V. and Cooley, R.L., 1987. Simultaneous confidence and prediction intervals for nonlinear regression models with application to a groundwater flow model. *Water Resour Res*, 23 (7), 1237-1250.

Welter, D.E., Doherty, J.E., Hunt, R.J., Muffels, C.T., Tonkin, M.J. and Shreüder, W.A., 2010. Approaches to Highly Parameterized Inversion: PEST++, a Parameter Estimation Code Optimized for Large Environmental Models. United States Geological Survey Techniques and Methods, Book 7, Section C5.

Welter, D.E., White, J.T., Hunt, R.J., and Doherty, J.E., 2015. Approaches to Highly Parameterized Inversion: PEST++ Version 3, a Parameter ESTimation and Uncertainty Analysis Software Suite Optimized for Large Environmental Models. United States Geological Survey Techniques and Methods 7-C12.

Welter, D.E. Doherty, J. and White, J.T., 2018. The PANTHER Model Run Manager. Published by CAELUM. Downloadable from http://www.caelumsoftware.org

Westenbroek, S.M., Doherty, J., Walker, J.F., Kelson, V.A., Hunt, R.J., and Cera, T.B., 2012, Approaches in highly parameterized inversion: TSPROC, a general time-series processor to assist in model calibration and result summarization: U.S. Geological Survey Techniques and Methods, book 7, chap. C7, 79 p., 3 apps., available only at https://pubs.usgs.gov/tm/tm7c7.

White, J.T., 2018. A model-independent iterative ensemble smoother for efficient history-matching and uncertainty quantification in very high dimensions. Submitted to *Environmental Modelling and Software*.

White, J.T., Doherty, J.E. and Hughes, J.D., 2014. Quantifying the predictive consequences of model error with linear subspace analysis. *Water Resour. Res*., 50 (2): 1152-1173. DOI: 10.1002/2013WR014767

White, J.T., Fienen, M.N. and Doherty, J.E., 2016. A python framework for environmental model uncertainty analysis. *Environmental Modelling and Software*, 85, 217-228.

White, J.T., Fienen, M.N., Barlow, P.M. and Welter, D.E., 2018. A tool for efficient, model-independent management optimization under uncertainty. *Environmental Modelling and Software*, 100, 213-221.

White, J.T., Welter, D.E. and Doherty, J., 2018. Manual for Version 4 of PEST++. Published by CAELUM. Downloadable from http://www.caelumsoftware.org

Appendix A: PEST-Suite Modifications for Improved PEST/PEST++ Inter-Operability

A.1 Introduction

This appendix briefly describes alterations that have been made to PEST-suite programs to improve their inter-operability with programs of the PEST++ suite. These changes ensure that a PEST++ control file can be read by PEST, BEOPEST and PEST_HP. The latter programs can, of course, undertake only the kinds of processing for which they were programmed. In doing so, they simply ignore variables which control the operation of PEST++, in the same way that members of the PEST++ suite ignore variables that have no bearing on tasks that they were programmed to perform.

PEST is accompanied by an extensive suite of utility support programs that implement a variety of PEST pre- and post-processing tasks. For version 15 of PEST, many (but not all) of the utilities which read a PEST control file have been modified to tolerate the presence of PEST++ control variables within that file. They also tolerate comments (see below), and an alteration to PEST++ control file specifications that is discussed below. Utility programs which read one PEST control file and write a new, modified PEST control file have been reprogramming to transfer PEST++ variables and comment lines from the original PEST control file to the new one.

Not all members of the PEST utility suite have been modified for PEST++ control file compatibility. Older and little used members of the suite have not undergone modification. Nevertheless these programs may find occasional use. To facilitate the running of these older programs, a new program named PSTCLEAN was added to version 15 of the PEST utility suite. This program modifies a PEST-control file so that it is readable by these older programs. PSTCLEAN is documented in part 2 of the PEST manual.

A.2 New Protocols

Two small additions have been introduced to the specifications of PEST and PEST++ control files coinciding with the release of version 15 of PEST and version 4 of PEST++. These are now described.

Comment lines no longer need to begin with the string "++#". They can now begin with the "#" character on its own. In fact, any text that follows a "#" character in a PEST control file is interpreted as a comment. The "#" character and ensuing comment are simply stripped from the respective line of the PEST control file before the values of any variables are read from that line. However the following conditions must be met for this to occur.

1. The "#" character must not be part of a text string that is enclosed by single or double quotes (indicating that it may be part of the name of a file).
2. If it is not enclosed by quotes, the "#" character must be located at the start of a line, or preceded by a blank character or tab. This ensures that it is not part of a filename that is not enclosed by quotes (presumably because the filename has no spaces).

A second change in protocol affects the "model input/output" section of the PEST control file. This single section can now be replaced by two sections; the name of the first is "model input" while the name of the second is “model output”. These header strings are preceded by the “*” character followed by a space when denoting respective sections in a PEST control file. Adoption of this new protocol allows a PEST control file to serve as an input control file for the PANTHER run manager (provided no file transfers are requested of the run manager).

The new protocol also accepts the presence of a blank line anywhere within a PEST control file.

A.3 Alterations to PEST-Suite Programs

PEST, BEOPEST and PEST_HP were altered to accommodate the new PEST control file protocol described in the preceding section. They were also programmed to accept (and ignore) lines that begin with the “++” string, signifying the presence of PEST++ keywords on that line.

A number of PEST-suite utility programs have been similarly altered. Table A1 lists these utilities. As is stated above, where a utility program writes a new PEST control file based on an original PEST control file, the new PEST control file inherits comments and PEST++ control variables from the original PEST control file. Exceptions to this are discussed below the table.

Table A1. Utility programs from the PEST suite that have undergone alteration to conform with new PEST control file protocols and the presence of “++” lines within a PEST control file.

Program	Purpose
ADDCOVMAT	Adds the names of files containing covariance matrices to a PEST control file.
ADDREG1	Adds “preferred value” regularisation to a PEST control file.
ADDREG2	Adds “preferred value” regularisation to a PEST control file; allows the user to assign values to some regularisation control variables.
ASSESSPAR	Assesses the credibility of a parameter set in terms of a user-supplied parameter probability distribution (provided the latter is multi-normal).
CALMAINTAIN	Computes alterations to parameter values that are required to maintain a model in a calibrated state if the values of a few user-specified parameters have been altered.
CMAES_P	A global optimizer based on the covariance matrix adaption scheme.
CMAES_HP	A version of CMAES_P that uses TCP/IP for communication between manager and workers.
GENLINPRED	Supervises the running of a suite of PEST utility programs to compute parameter and predictive uncertainty, together with related quantities.
IDENTPAR	Computes parameter identifiabilities.
INFSTAT	Computes observation influence statistics where an inverse problem is well-posed.
INFSTAT1	Computes observation influence statistics where an inverse problem is ill-posed.
JACTEST	Tests the integrity of finite-difference derivatives by undertaking successive incremental variation of a user-specified parameter while monitoring model outputs.
JCO2JCO	Writes a Jacobian matrix file (i.e. a JCO file) that complements a user-supplied PEST control file based on the contents of an existing PEST control file and complementary JCO file.
JCOCHEK	Checks that a JCO file is consistent with a PEST control file.

MULJCOSEN	Computes composite sensitivities of an observation or parameter for a sequence of Jacobian matrices; comparison of these sensitivities allows assessment of model nonlinearity.
PARREP	Writes a new PEST control file in which initial parameter values are replaced by values read from a parameter value file.
PARREP_RRF	Writes a new PEST control file in which initial parameter values are replaced by values read from a parameter set featured in a run results file.
PARVAR1	Computes the post-calibration error variance of all parameters featured in an inverse problem.
PCOST_HP	Estimates the cost of a PEST_HP run.
PESTCHEK	Checks a PEST input dataset (including the PEST control file) for correctness and consistency.
PESTRUNPREP	Prepares for a series of PEST runs in which a randomized Jacobian matrix is calculated from the outcomes of parameter upgrades.
PESTRUNPROC	Post-processes a series of PEST runs in which a randomized Jacobian matrix is calculated from the outcomes of parameter upgrades; calculates a new Jacobian matrix approximation based on the outcomes of these runs.
PNULPAR	Writes a secondary set of parameter value files from a primary set of parameter value files after members of the primary set are subjected to null space projection.
PREDUNC1	Uses a linear approximation to Bayes equation to calculate the uncertainty of a model prediction.
PREDUNC4	Computes the contribution made to the uncertainty of a model prediction by different parameters or parameter groups.
PREDUNC5	Computes the ability of individual or collective observations to reduce the uncertainty of a user-specified prediction.
PREDUNC6	Computes post-calibration uncertainties of multiple predictions.
PREDUNC7	Computes a posterior covariance matrix using a linear approximation to Bayes equation.
PREDVAR1	Computes the post-calibration error variance of a prediction where calibration is implemented using different numbers of singular values.
PREDVAR1A	Performs a similar function to PREDVAR1; however can handle multiple predictions while employing a slightly different numerical methodology to that used by PREDVAR1.
PREDVAR1B	Calculates calibration-induced predictive bias incurred by model defects.
PREDVAR1C	Similar to PREDVAR1B but includes the effect of model defects on model-to-measurement fit.
PREDVAR2	Similar to PREDVAR1; however automates use of a number of different prior covariance matrices.
PREDVAR3	Performs a similar role to that of PREDVAR4; now largely superseded by the latter program.
PREDVAR4	Similar to PREDUNC4; however focuses on predictive error variance rather than predictive uncertainty.
PREDVAR5	Similar to PREDUNC5; however focusses on predictive error variance rather than predictive uncertainty.
PWHISP_HP	This is the “PEST whisperer”; it offers voluminous advice on the outcomes of a PEST_HP run.
RANDPAR	Generates a set of parameter value files containing random parameter values.
RANDPAR1	Performs the same function as RANDPAR but uses a different methodology for random parameter set generation.

PREPESTRUNS	Prepares for a random parameter adjustment processes based on randomized Jacobian matrices.
PWTADJ1	Writes a new PEST control file that features identical parameters and observations to an existing PEST control file; weights are adjusted for equal visibility of objective function components.
PWTADJ2	Writes a new PEST control file that features identical parameters and observations to an existing PEST control file; weights are adjusted to equal inverse of standard deviation of “measurement noise”.
RRF2JCO	Computes a randomized Jacobian matrix from parameter and model output values stored in a run results file.
SCALEPAR	Creates a new PEST input dataset, and Jacobian matrix, based on scaled rather than native parameters.
SIMCASE	Designs a new inverse problem in which all redundant parameters and observations are omitted from the original inverse problem.
SSSTAT	Computes a suite of post-calibration statistics using subspace methods.
SUBREG1	Removes regularisation from a PEST control file.
SUPCALC	Computes the dimensionality of the solution and null spaces for an inverse problem.
SUPOBSPAR	Computes and lists “super parameters” and corresponding “super observations” obtained through singular value decomposition of the weighted Jacobian matrix.
SUPOBSPAR1	Similar to SUPOBSPAR; however undertakes Kahunen-Loève transformation of parameters prior to computation of super parameters and super observations.
SUPOBSPREP	Builds a new PEST control file from an existing PEST control file; in the new PEST control file super observations replace normal observations.
SVDAPREP	Prepares an input dataset for SVD-assisted inversion; see note below.
WTFACTOR	Multiplies weights in a user-specified observation group by a user-specified factor.
WTSENOUT	Transforms a number of PEST outputs in accordance with weighted sensitivities.

Note the following.

1. Many members of the PEST utility suite do not read a PEST control file, and hence do not require alteration to accommodate the new control file protocols described above. Included among these are PAR2PAR, OBS2OBS, RDMULRES, MULPARTAB, matrix manipulation utilities, and others.
2. The SVDAPREP utility will not generate a PEST input dataset for SVD-assisted inversion if it detects comments or the presence of “++” lines in a PEST control file. Instead, it ceases execution with an appropriate error message. This is because neither PEST, BEOPEST nor PEST_HP tolerate the presence of these features in a base parameter PEST control file when undertaking SVD-assisted inversion in accordance with the contents of a super parameter PEST control file. It was decided not to alter the pertinent part of the PEST code to support the new PEST control file protocol during SVD-assisted inversion for the following reasons.
    a. It avoids the possibility of introducing bugs to a complex part of the PEST code;
    b. A user has the option of using PEST++ for SVD-assisted inversion instead of PEST/BEOPEST/PEST_HP; PEST++ does not require separate base-parameter and super-parameter control files;
    c. If, however, a user wishes to employ PEST/BEOPEST/PEST_HP instead of PEST++ for SVD-assisted inversion, new-protocol features in a PEST control

file are easily removed using the PSTCLEAN utility. SVDAPREP can then be run using the PST_CLEAN-modified PEST control file to create a PEST input dataset for SVD-assisted inversion.

3. PWHISP_HP, the “PEST whisperer”, accommodates the new PEST control file protocol. However its execution fails if it is asked to comment on a PEST_HP run where PEST_HP undertakes SVD-assisted inversion and the base PEST control file contains features which belong to the new protocol. This is not seen as a problem, as PEST_HP will also fail under these circumstances; see above.
4. As is described in part 2 of the PEST manual, a “parameter uncertainty file” can cite a PEST control file as the source for some or all measurement-noise-induced uncertainties pertaining to members of a calibration dataset. PEST-suite programs will not accommodate the presence of “++” lines and comments in this PEST control file. A large number of PEST utilities read a parameter uncertainty file. However the vast majority of these utilities obtain parameter, rather than observation, uncertainties from this file. Observation uncertainties are obtained from elsewhere (normally the case-defining PEST control file). Hence use of a PEST control file cited in a parameter uncertainty file to obtain measurement uncertainties is likely to be a rare occurrence.
5. A few members of the Groundwater Data Utilities suite read a PEST control file. These have not been altered to accommodate the new protocols for this file. The PSTCLEAN utility can be employed to render a PEST control file readable by these utilities.