USGS
science for a changing world

Great Lakes Restoration Initiative

Approaches in Highly Parameterized Inversion: PEST++, a Parameter ESTimation Code Optimized for Large Environmental Models

Techniques and Methods, Book 7, Section C5

U.S. Department of the Interior
U.S. Geological Survey

Cover: Image from http://modis.gsfc.nasa.gov/gallery/individual.php?db_date=2010-09-06.

Approaches in Highly Parameterized Inversion: PEST++, a Parameter ESTimation Code Optimized for Large Environmental Models

By David E. Welter, John E. Doherty, Randall J. Hunt, Christopher T. Muffels, Matthew J. Tonkin, and Willem A. Schreüder

Great Lakes Restoration Initiative

Techniques and Methods, Book 7, Section C5

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
KEN SALAZAR, Secretary

U.S. Geological Survey
Marcia K. McNutt, Director

U.S. Geological Survey, Reston, Virginia: 2012

This and other USGS information products are available at http://store.usgs.gov/
U.S. Geological Survey
Box 25286, Denver Federal Center
Denver, CO 80225

To learn about the USGS and its information products visit http://www.usgs.gov/
1-888-ASK-USGS

Suggested citation:
Welter, D.E., Doherty, J.E., Hunt, R.J., Muffels, C.T., Tonkin, M.J., and Schreüder, W.A., 2012, Approaches in highly parameterized inversion—PEST++, a Parameter ESTimation code optimized for large environmental models: U.S. Geological Survey Techniques and Methods, book 7, section C5, 47 p.

Contents

Figure

Tables

Approaches in Highly Parameterized Inversion: PEST++, a Parameter ESTimation Code Optimized for Large Environmental Models

By David E. Welter,[1] John E. Doherty,[2] Randall J. Hunt,[3] Christopher T. Muffels,[4] Matthew J. Tonkin,[4] and Willem A. Schreüder[5]

Abstract

An object-oriented parameter estimation code was developed to incorporate benefits of object-oriented programming techniques for solving large parameter estimation modeling problems. The code is written in C++ and is a formulation and expansion of the algorithms included in PEST, a widely used parameter estimation code written in Fortran. The new code is called PEST++ and is designed to lower the barriers of entry for users and developers while providing efficient algorithms that can accommodate large, highly parameterized problems. This effort has focused on (1) implementing the most popular features of PEST in a fashion that is easy for novice or experienced modelers to use and (2) creating a software design that is easy to extend; that is, this effort provides a documented object-oriented framework designed from the ground up to be modular and extensible. In addition, all PEST++ source code and its associated libraries, as well as the general run manager source code, have been integrated in the Microsoft Visual Studio®[6] 2010 integrated development environment. The PEST++ code is designed to provide a foundation for an open-source development environment capable of producing robust and efficient parameter estimation tools for the environmental modeling community into the future.

Introduction

Because of the inherent inability of mathematical simulations to perfectly characterize a complex natural world, it is becoming well recognized that numerical models representing the natural world cannot make predictions without uncertainty. The uncertainty associated with environmental model predictions is usually much higher than is common for model predictions in sciences such as engineering and physics because the physical properties of environmental systems are highly variable and possess complex distributions that can never be known in sufficient detail. This recognition has led us to realize that forecasts of these systems need to be based on more probabilistic approaches (see, for example, Tonkin and Doherty, 2009): analysts realize that the calibration process is an inherently non-unique (underdetermined) problem and that infinitely many parameter sets can be found to calibrate a model. Hence, quantifying the accuracy of environmental models for real-world applications is becoming a larger part of the literature and, in turn, standard industry practice. Oreskes and others (1994), Saltelli and others (2004), Pilkey and Pilkey-Jarvis (2007), Beven (2009), and Doherty (2011) discuss underlying modeling and uncertainty issues in detail and offer suggestions on the appropriate roles and uses of models in environmental planning and decision making.

At the same time, parameter estimation and uncertainty analyses are transitioning to a standard component of defensible modeling. Doherty, Hunt, and Tonkin (2010) and Moore and others (2010) address this issue from the parameterization and uncertainty-analysis standpoint by use of regularized inversion and Pareto analysis techniques; Beven (2009), by use of the concept of equifinality; and Saltelli and others (2004), by use of rigorous sensitivity analysis.

PEST (Doherty, 2010a) is a widely used parameter estimation code in the environmental modeling community, and it is notable for its sophisticated tools available for highly parameterized regularized inversion (Doherty and Hunt, 2010), where "highly parameterized" refers to models having more parameters than can be uniquely estimated from the calibration dataset. Because highly parameterized models are inherently non-unique, they require additional constraints. The use of regularized inversion—by which mathematics is employed to insert soft knowledge and stability into a modeling problem—has grown in recent years as computational

[1]Computational Water Resource Engineering.

[2]Flinders University and Watermark Numerical Computing.

[3]U.S. Geological Survey.

[4]S.S. Papadopulos and Associates, Inc.

[5]Principia Mathematica, Inc.

[6]"Visual Studio" is a registered trademark of Microsoft Corporation in the United States and other countries.

power has increased. The propagation of relatively inexpensive multicore processors to the desktop arena and the advent of cloud computing (Hunt and others, 2010), in conjunction with a greater need to shift the focus of modeling from simple calibration to portraying uncertainty, has motivated more modelers to learn and apply and these tools. One inherent aspect of a regularized-inversion approach is to embrace—not artificially limit—investigation of parameters that may be important for a model prediction; in other words, "A regularized-inversion philosophy to parameterization, then, can be summarized as 'if in doubt, include it'"(Doherty and Hunt, 2010).The corresponding increase in modeling problem size has pushed many existing software tools to their limits, especially as advancements to the science make the tools more sophisticated, and in turn has created a two-faceted need: to make these tools easier and more intuitive for new users but also more robust and efficient for those working on increasingly complex and more highly parameterized problems. As such, this work is intended to fulfill a need for projects such as the Great Lakes Restoration Initiative, whereby Great Lake watershed-scale models are calibrated, climate and landuse scenarios are simulated, and uncertainty analyses are performed.

Purpose and Scope

This report describes a reformulation and expansion of the algorithms included in PEST, a widely used parameter estimation code written in Fortran. This new program has been named PEST++, which reflects its being a C++ object-oriented adaptation of the original PEST. Source code and executable of PEST++ are available for download at *http://pubs.usgs.gov/tm/tm7c5/*.

This report targets two types of readers. For the practitioner interested in applying these powerful approaches, input instructions and an example problem are given in appendix 5. Most of the report, however, presents the more advanced concepts of the program's design in order to facilitate code development by others. All parameter-estimation-related terminology, concepts, file extensions, and so forth, use the conventions and derivations presented and cited by Doherty (2010a) and Doherty and Hunt (2010) and are omitted here for brevity.

The goals of PEST++ development are to (1) lower the barriers of entry for new users of parameter estimation software, (2) develop efficient parameter estimation tools and algorithms appropriate for implementing the techniques discussed by Doherty and Hunt (2010) for solving highly parameterized problems, and (3) develop an object-oriented framework to support future development. This report documents the object-oriented design techniques to achieve these goals, a design approach not available in the coding of the original PEST. The programming language C++ was chosen for the sake of efficiency and better support for large/highly parameterized problems. Given the change in programming language and the inclusion of object-orientated design, the report is structured to aid more advanced users in extending the code by providing programming concepts in the main report body and additional details in the appendixes.

PEST++ does not attempt to reproduce all the functionality of PEST but instead focuses on implementing the most used features and improving the modeler's access to these features, making them easier to use. Thus, this report focuses on differences and enhancements with respect to the widely used PEST code of Doherty (2010a). In particular, this reformulation of PEST is geared toward making it easier to implement the parameter estimation guidelines provided by Doherty and Hunt (2010). These guidelines are founded on the use of a large number of parameters with soft-knowledge (Tikhonov) and subspace (singular value decomposition, or SVD) methods for regularization in a hybrid approach to ensure that "the twin ideals of parsimony—simple as possible, but not simpler—are fully met." The reader is directed to Hunt and others (2007), Doherty and Hunt (2009, 2010), and Doherty, Hunt, and Tonkin (2010) for detailed discussion of these concepts.

Enhancements of and Changes to PEST

Although PEST++ is based on PEST (Doherty, 2010a, b), PEST++ implements only a subset of the most used PEST functions and focuses on providing a simpler interface to these selected features. To accomplish this, the PEST++ development effort has added several new methods and has streamlined, combined, and/or enhanced several existing PEST methods. These enhancements and changes include the following:

1. PEST++ has the ability to automatically switch between native parameters and superparameters (Tonkin and Doherty, 2005) without user intervention.
2. Capabilities of the Gauss-Marquardt-Levenberg method for avoiding local minima have been supplanted by implementing an efficient and straightforward singular value decomposition approach that retains the important functionality of Marquardt lambda.
3. The PROPACK singular value decomposition routines for large and sparse matrices (Larsen, 2001) have been included for increased computational efficiency.
4. PEST++ has the ability to automatically normalize parameters.
5. The advanced calculation of superparameter derivatives is now possible.
6. An interface for the GENIE parallel run manager has been provided. GENIE (Muffels and others, 2011, 2012) is generalized run manager software that manages the task of scheduling and parallel running of models on multiple computers by using TCP/IP protocol for communication. It was developed in conjunction with this effort, and it is fully documented in Muffels and others (2012).

In the next sections, background of PEST++ code design is provided and the enhancements are described in more detail. This description notwithstanding, it is expected that this list of enhancements will expand as PEST++ is applied to more real-world problems in the future.

Design Background: Extensible Framework Based on Generic Transformations

Object-orientated design techniques were used to develop an extensible framework that can generalize the transformations operating in PEST (Doherty, 2010a) while facilitating incorporation of additional transformations in PEST++. This framework is built around three parameter states, where "state" refers to how input provided by the user is fed to the parameter estimation process (fig. 1, first row). Although the original PEST does not formally define these three parameter states, it differentiates between them internally in the code as a means to improve the numerical performance and provide convenience and flexibility. PEST++ is similar to PEST in that—if the developer does not wish to use the parameter state formulation provided here—these states can share a common set of parameters (= one parameter state).

The numerical parameter state contains the parameter values used internally by PEST++ algorithms to solve the least-squares problem. The model parameters are the parameter values PEST++ uses to run the external model being calibrated, and the control file parameters are the parameter values that appear in the PEST control files (*.pst*) and parameter files (*.par*).The control file parameter state is not strictly required for PEST++; however, it has been added to retain backward compatibility with the PEST input and output files that use this state. The transformation sequences shown in the second row of figure 1 provide a mapping between the parameter states listed in the first row. Transformation, as used here, refers to the connecting of information contained in the three parameter states in known relations useful for parameter estimation. The "control file to numeric" transformation sequence provides a mapping between control file and numeric parameters; the "control file to model" transformation sequence provides a mapping between control file and model parameters. Transformation sequences provide both a forward and a reverse transformation. Transformation sequences are constructed from individual transformations; the third row of figure 1 lists the individual transformations that build transformation sequences and reproduce the current functionality of PEST in PEST++.

A simple example demonstrating the interaction of the parameter states and the transformation sequences using actual parameter values is given in tables 1 and 2. Table 1 summarizes the information about the parameters specified in a hypothetical PEST/PEST++ control file, and table 2 shows how this information is translated into the PEST++ parameter states and transformation sequences.

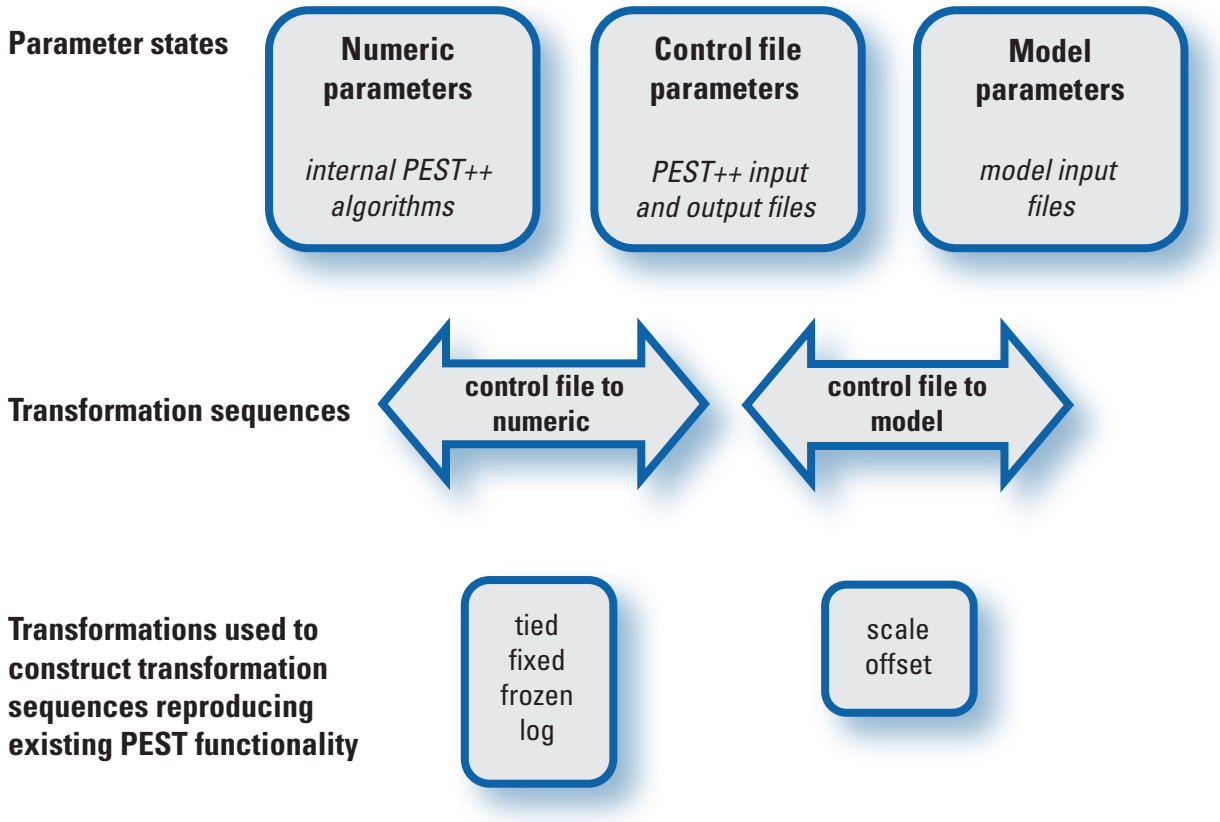

Figure 1. PEST++ parameter states and transformation sequences.

Table 1. Parameter information specified in a hypothetical PEST/PEST++ control file.

Parameter	Value in *.pst* file	Transformations
P1	2.0	scale = 2.0
P2	10.0	log
P3	4.0	none
P4	8.0	tied to P3

One-to-One Property

Accurate parameter/observation derivatives are important for proper function of the numerical methods employed in PEST and PEST++.To ensure the quality of these derivatives, PEST performs an extra step after writing the model input files and reads the parameter values back from the newly created model input files to correct for truncation errors that may have occurred while creating these files. Truncation errors can occur when the fields of an input file are of fixed width, limiting the precision in which the parameters can be written. This correction feature can greatly improve the fidelity of derivatives and improve the performance of the numerical algorithms; however, it is possible to take advantage of this feature only when all transformations map a single numeric parameter to a single model parameter and vice versa. To identify this condition, PEST++ defines an attribute associated with elementary transformations to determine whether back substitution is applicable; that is, whether the one-to-one relationship holds. If all the elementary transformations in the "control file to numeric" and "control file to model" transformation sequences are one to one, then PEST++ performs parameter back substitution.

Elementary Transformations

PEST++ uses the transformation sequences "control file to numeric" and "control file to model" to convert between parameter states. These sequences are composed of the elementary transformations that support forward and reverse transformations and contain the "one-to-one" attribute (discussed previously) to indicate whether they are compatible with parameter back substitution. A listing and short descriptions of the elementary transformations included in PEST++ are given in appendix 2.

Periodic SVD Execution During an SVD-Assist Run

Doherty and Hunt (2010) describe what is desirable in a calibration tool and how singular value decomposition can fill this role: "When large numbers of parameters are added to a model, some can be expected to be insensitive and others highly correlated with other parameters. As a result, even though a parameter *may* be estimable (therefore worth including in the calibration process), it doesn't mean that it actually *is* estimable. What is needed is an intelligent calibration tool—one that detects what can and cannot be inferred from the calibration dataset and then estimates what it can and leaves out what it can't—all automatically, without user intervention. Singular value decomposition (SVD) is such a tool." The end result is that SVD is a numerical approach that is simple to use, robust, and inherently stable (Kalman, 1996), making it appropriate for both novice and advanced users.

The power of SVD is currently available in PEST, where the parameters are estimated on the basis of a reduced set of orthogonal linear combinations of base parameters rather than by attempting to estimate the full suite of base or native parameters individually. Such an approach facilitates obtaining an unconditionally stable solution. Although additional description of SVD is beyond the scope of this report, an in-depth description of this method as implemented in PEST can be found in Doherty and Hunt (2010) and Doherty (2010a, b). SVD as implemented in PEST, however, requires that the parameter sensitivities to observations (that is, the Jacobian matrix) be calculated by using all base parameters with each PEST iteration. Thus, the overall computational burden is not reduced; rather, the parameter estimation problem is ensured to be tractable by the unconditional stability afforded by SVD.

In recognition of the need to reduce computational burdens, Tonkin and Doherty (2005) describe the use of superparameters, whereby sensitivities and associated singular values are calculated at initial parameter values that are assumed to be representative for all parameter upgrades during the parameter estimation process. This approach allows the parameter estimation process, after an initial calculation of sensitivities of all base parameters, to perform all subsequent sensitivity calculations needed to upgrade parameters on the sole basis of the reduced number of superparameters. This technique greatly reduces the number of forward model runs needed for the parameter upgrades calculated in each PEST iteration. The capability to solve the parameter estimation problem in terms

Table 2. Parameter states and transformation sequences for hypothetical example.

Parameter name	Numeric parameter value	Control file to numeric transformation sequence	Control file parameter values	Control file to model transformation sequence	Model parameter values
P1	2.0		2.0	← Scale transform →	4.0
P2	1.0	← Log transform →	10.0		10.0
P3	4.0		4.0		4.0
P4	N/A	← Tied transform →	8.0		8.0

of superparameters rather than base parameters is defined as SVD-Assist, or SVDA, by Doherty (2010a). Because SVDA works with a subset of the full spectrum of singular values, it can greatly reduce the computational burden associated with solving large highly parameterized problems. Moreover, singular values in the null space (Moore and Doherty, 2005; Doherty, Hunt, and Tonkin, 2010) are defined to have no significant effect on the simulation of the weighted observations and thus can be dropped from the solution to reduce the dimensionality of the problem. Because the optimal number of superparameters can only be estimated at the beginning of the parameter estimation process, Doherty and Hunt (2010) suggest specifying more superparameters than thought to be supported by observation data, then applying SVD on the superparameters to ensure numerical stability.

Although SVDA can significantly reduce runtimes, intermediate processing is needed to translate the base parameters and associated Tikhonov regularization to the superparameters. These translations are performed by PEST utilities PICALC (Prior Information CALCulation) and PARCALC (PARameter CALCulation), which are most often constructed automatically by the PEST utility SVDAPREP (SVD-Assist PREParation). In total, the entire PEST SVDA process requires (1) building an initial Jacobian matrix of the base parameters, (2) running SVDAPREP to create a PEST control file of the superparameter problem, and (3) running PEST on the PEST control file created by SVDAPREP. PEST++ obviates these steps and thus greatly simplifies the SVDA process by reducing the entire operation to a single transformation added to the end of the "control file to numeric" transformation sequence.

Automatically Switching Between SVD and SVDA

Because switching between superparameters and base parameters is seamless in PEST++, the modeler no longer must assume that sensitivities calculated at initial values are representative of those of all possible parameter upgrades—an assumption that may or may not hold depending on the problem. Highly nonlinear problems may cause differences in sensitivities when calculated at initial and optimal values; thus, the solution space and null space of the optimized parameters can differ from those of the initial parameters. PEST++ includes an option to solve the parameter estimation problem by cycling between base parameters and superparameters, automatically addressing the potential problem that can result from the rigid SVDA assumption (Doherty and Hunt, 2010).

This ability to cycle between base parameters and superparameters has been incorporated to automatically account for these issues and is supported via the variables N_ITER_BASE and N_ITER_SUPER in the input control file (see appendix 1).When these variables are specified, PEST++ will perform N_ITER_BASE iterations using native parameters followed by N_ITER_SUPER iterations using superparameters. The overall iterative solution is similar to that used by PEST, and the PEST variables NOPTMAN, PHIREDSTP, NPHINORED, RELPARSTP, and NRELSTP, which control the iterative solution process, retain all previous functionality. The newly added variables N_ITER_BASE and N_ITER_SUPER control only the type of solution performed each iteration (that is, whether superparameters or base parameters are used).

This automatic switching feature is most efficient when a single base parameter iteration is followed a number of superparameter iterations, followed by a single base parameter iteration, followed by a number of superparameter iterations, and so on. Periodically performing a base parameter iteration allows automatic updates to the calculation and composition of the superparameters used in subsequent parameter estimation. Thus, the PEST++ parameter estimation takes advantage of the fact that superparameters are computationally efficient, but it also addresses the potential adverse results that can result from rigidly adhering to the SVDA linearity assumptions, which can become violated when the optimal parameters move away from the initial values, especially when solving nonlinear problems. We note that the current methodology that uses N_ITER_BASE and N_SUPER_BASE does not fully exploit potential gains in efficiency that can be realized by switching between base parameters and superparameters; that is, by monitoring the progress of the SVDA run independently and moving away from the single iteration loop in PEST, it may be possible to realize additional gains in performance, a potential that will be investigated in future work.

Prior Information

When using superparameters, PEST requires that the prior-information section of the control file be calculated outside of PEST and recast as observations (as performed by *PICALC.exe*). PEST++ has removed this restriction, thus simplifying the specification of prior information to be used in conjunction with superparameters. This simplification makes the SVD-Assist links between parameters and regularization more transparent and allows the same control file to be used for both native parameters and superparameters.

The Marquardt Lambda and SVD Rotation Factor

PEST provides the Gauss-Marquardt-Levenberg method and SVD as numerical solution techniques for the least-squares problem, but PEST++ retains only the SVD. This simplification reduces numerical issues associated with misspecification of the Gauss-Marquardt-Levenberg parameters that can commonly confound novice users, whose confusion in turn confounds efficient parameter estimation of highly parameterized problems. Briefly, the Marquardt lambda is a component of the Gauss-Marquardt-Levenberg method, but in practice it also functions in PEST as a solution supplement to the SVD formulation in the following ways. When the Marquardt lambda is zero, the upgrade vector is in the direction of the Gauss-Newton solution, and increasing its

magnitude has the effect of rotating the upgrade vector in the direction of the steepest gradient descent solution. This rotation is accomplished by adding terms to the diagonal of the normal equation matrix. Adding these terms tends to make the matrix better conditioned, which has a stabilizing effect and serves as a de facto form of regularization. Moreover, it can provide sufficient regularization so as to make an ill-posed problem solvable, but this regularization phenomenon is often not apparent to the user. Use of the Marquardt lambda is not an optimal regularization strategy, however, because it does not restrain the solution in a physically meaningful way. Indeed, such a regularization approach is contrary to Tikhonov regularization, which identifies one or more preferred conditions, and truncated SVD, which minimizes changes for parameters whose influence on the inverse problem is overwhelmed by noise in the solution.

Despite undesirable de facto regularization, the Marquardt lambda role for rotating the upgrade vector in the direction of the gradient descent solution is valuable for enhancing the search capability to the nonlinear least-squares solution. Testing different values of the Marquardt lambda allows PEST to explore a larger portion of parameter space, helping to prevent its being trapped in a local minimum. Because SVD incorporates subspace regularization, however, it does not benefit from the Marquardt lambda's ability to provide less-than-optimal regularization. Therefore, there is value in retaining attractive aspects of the Marquardt lambda (robust search capability to help avoid local minimum) while minimizing the undesirable de facto regularization. Towards this end, the Gauss-Marquardt-Levenberg method is supplanted in PEST++ by an alternative method of providing a robust search functionality analogous to a widely varying Marquardt lambda in the SVD solution.

For large problems, performing an SVD factorization can be computationally expensive. This makes the direct application of Marquardt Lambda undesirable because it would require a SVD factorization be performed for each value of the Marquardt lambda. However, because SVD is inherently stable, the role of the Marquardt lambda can be reduced from adding a diagonal term to simply providing a means to rotate the upgrade vector. To take advantage of this, PEST++ replaces the Marquardt lambda with a newly defined "rotation factor," which rotates the upgrade vector in the direction of the gradient descent solution. This rotation is achieved through a simple matrix multiplication and obviates SVD factorization for each value of lambda. In addition, the physical meaning is very clear, and the rotation factor must lie in the interval [0, 1], as opposed to there being no clear upper limit on Marquardt lambda. A more in-depth discussion and the mathematical derivation of the rotation factor are included in appendix 3.

PROPACK

PROPACK is a software library containing an iterative method for computing SVD factorizations (Larsen, 1998) that is very efficient for large sparse matrices, making it ideal for working with highly parameterized problems. Unlike the LAPACK implementation of SVD commonly used in PEST, which always calculates all of the singular vectors, PROPACK is able to compute a subset of the singular vectors. In the context of PEST++, this means that PROPACK can be used to calculate only the singular vectors in the solution space while ignoring unneeded calculation of remaining singular vectors in the null space. This reduces computational burden and the amount of memory necessary to perform a SVD factorization on a large problem. For example, when Muffels (2008) integrated PROPACK in the PEST utility PREDVAR1, he found that PROPACK was able to compute the first 100 singular values in several seconds, whereas LAPACK SVD took several minutes to compute the complete factorization. In PEST++, PROPACK provides a powerful and efficient numerical algorithm that can greatly improve performance when solving large highly parameterized problems. PROPACK is also an upgrade of the LSQR algorithm (Paige and Saunders, 1982a, b) included in PEST. Whereas LSQR typically does not guarantee the orthogonality of the singular vectors, PROPACK does.

Automatic Parameter Normalization

Normalizing parameters before solving the parameter estimation problem can greatly improve the efficiency of the underlying numerical techniques. In PEST++, normalization transforms the parameters to a common scale by dividing by the standard deviation. Adding normalization to the calibration process in PEST++ is accomplished by simply adding the variable AUTO_NORM(std) to the PEST++ section of the control file, where std is the number of standard deviations represented by the difference between the maximum and minimum allowable parameter values. When the upper and lower bounds are assumed to be the 95-percent confidence interval, which is their most common interpretation, std should be set to 4. In the PEST++ code, the functionality of automatic parameter normalization is implemented by adding the TranNorm transformation to "control file to numeric" transformation sequence. It is performed after all of the other transformations, with the exception of the SVDA transformation. The normalization factor is based on the transformed values of the parameter limits.

Advanced Calculation of Superparameter Derivatives

When using superparameters in the parameter estimation process, the magnitude of a singular value typically does not provide a good reference point for computing perturbations of derivatives. During the developmental stages of PEST++, it was not uncommon for base model parameters to encroach on the specified parameter bounds when the singular values were used to compute the derivative increments. To mitigate this problem and make derivative increments more consistent across large and small singular values, increments for

superparameter derivatives are based on the increment specified in the control file for the native parameter that has the highest contribution to a particular singular vector. Using this method, the native parameter with the highest contribution to a singular vector is allowed to vary by the amount it would have when computing the derivatives of native parameters.

GENIE Run Manager

GENIE (Muffels and others, 2011, 2012) is a suite of programs that manages and executes model runs in parallel across a network, using the TCP/IP protocol. It was designed for scalability and provides features to minimize bottlenecks and automatically balance loads. Detailed documentation for GENIE is provided in Muffels and others (2012) and is only cursorily discussed here. To use GENIE with PEST++, the GMAN_SOCKET variable must be specified in the PEST++ section of the control file (see appendix 1). It is likely that this utility will be the primary approach for managing the large numbers of runs required by PEST++.

Supported PEST Capabilities

In addition to enhancements to the existing PEST program of Doherty (2010a), a subset of existing functionality has been ported to PEST++ version 1.0, as listed below. The reader is referred to Doherty (2010a) for detailed description of these ported capabilities. It is expected the list of ported PEST capability will expand as PEST++ is applied to more real-world problems in the future.

Derivative Calculation Modes

Calculating derivatives of observations with respect to parameters is a fundamental part of the least-squares solution. Similar to PEST, PEST++ provides three methods for computing derivatives: (1) forward difference, (2) central difference (outer), and (3) central difference (parabolic). These methods provide the same functionality as their PEST counterparts. The forward derivative option is the most efficient and requires only one model run per parameter. Although the central difference options require two runs per parameter, they generally result in better derivatives. Doherty (2010a) explains all three methods in detail.

Derivative Switching

PEST++ provides the same support as PEST for automatically switching from forward to central derivatives as parameter estimation progress slows. This functionality is described in detail in the PEST user's manual Doherty (2010a) and is specified in the parameter groups section of the PEST control file.

Parameter Back Substitution

Accurate derivatives are important for the numerical gradient methods employed by PEST++ to function properly. To ensure the quality of these derivatives, PEST++ performs an extra step after writing the model input files and reads the parameter back in from the files to correct for truncation errors. This feature was adapted from PEST and can greatly improve the fidelity of the derivatives, thereby improving performance of PEST++ when the model input files contain truncation errors.

User Interface

To maintain compatibility with PEST, changes to the file formats and user interfaces have been minimized. This consistency facilitates switching between the enhancements available in PEST++ and the full capability of PEST and allows the user to leverage the strengths of both while providing PEST++ compatibility with the large set of utilities included in the PEST suite of tools. PEST++ retains PEST's format for the template (*.tpl*), instruction (*.ins*), parameter value (*.par*), and Jacobian matrix (*.jco*) files, thereby ensuring compatibility with the majority of the PEST's uncertainty utilities (see Doherty 2010a, b; and Doherty, Hunt, and Tonkin, 2010). PEST++ also retains support for PEST's control file (*.pst*). Where PEST++ offers additional functionality, these features can be accessed through optional lines beginning with "++" (see appendix 1). Because SVD is optional in PEST, PEST++ provides default values for the parameters in the SVD section of the PEST input file if these values are not specified by the user.

Development Environment

All the source code required to build PEST++ has been consolidated in the Microsoft Visual Studio® 2010, an integrated development environment. This includes the source codes for (1) PEST++, (2) the GENIE interface, (3) LAPACK++, and (4) the PEST Fortran code that processes template and insertion files. The project is configured to build a statically linked PEST++ executable without any external dependencies. This simplifies and facilitates sharing the source code and distributing the executable. The PEST++ Visual Studio Project, as well as source code and executable, can be obtained from *http://pubs.usgs.gov/tm/tm7c5/*.

Limitations of Version 1.0

In order to make PEST++ more accessible to new users, some features of the full suite of PEST capability are not available. Most notable are the following:

- Full observation covariance weights matrix is not supported.
- Automatic adjustment of the regularization weights via the regularization mode in PEST is not supported.
- The GENIE interface supports only a single command line. If multiple command lines are needed to run the model, then a batch file must be created.
- Output is limited to *.rec*, *.par* and *.jco* files.
- PEST++ does not perform a final run with the best parameters. A strategy needs to be developed to implement a final run when using GENIE (Muffels and others, 2011, 2012).
- The Jacobian (*.jco*) file for superparameter iterations is written in terms of the superparameters. This requires that users map this model output back to the native parameters if they wish to interrogate the contents of the Jacobian matrix in detail.
- Moreover, although this program has been used by the U.S. Geological Survey (USGS), no warranty, expressed or implied, is made by the USGS or the U.S. Government as to the accuracy and functioning of the program and related program material nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the USGS in connection therewith.

Summary

An object-oriented parameter estimation model, PEST++, was developed in C++ as a means of porting the popular PEST parameter estimation code to an object-oriented language. PEST++ does not reproduce all the functionality of PEST but instead replicates the features most commonly of use in parameter estimation of large, highly parameterized problems. Focus on the most used features results in a less complex learning curve, a more modern programming interface, and an extensible code base. As computing power has increased massively, parallel computing has become more accessible for model calibration and uncertainty exercises. As a result, problems being posed in a parameter estimation framework are rapidly increasing in size and complexity. PEST++ has been designed to account for these drivers; thus, this effort has focused developing a tool for handling large and highly parameterized problems while retaining much of the power and commonly used elements of PEST. Towards this end, PEST++ includes enhancements to PEST, such as linking to the general GENIE run manager and the PROPACK SVD/LSQR toolset. In addition, a subset of the PEST capabilities has been ported to PEST++. All the code necessary to produce a statically linked PEST++ executable has been consolidated into the Microsoft Visual Studio® 2010 integrated development environment.

References

Beven, K., 2009, Environmental modelling—An uncertain future?: New York, Routledge, 310 p.

Doherty, J., 2010a, PEST, Model-independent parameter estimation—User manual (5th ed., with slight additions): Brisbane, Australia, Watermark Numerical Computing.

Doherty, J., 2010b, Addendum to the PEST manual: Brisbane, Australia, Watermark Numerical Computing.

Doherty, J., 2011, Modeling—Picture perfect or abstract art?: Ground Water, v. 49, no. 4, p. 455, doi:10.1111/j.1745-6584.2011.00812.x.

Doherty, J., and Hunt, R.J., 2009, Response to comment on "Two statistics for evaluating parameter identifiability and error reduction": Journal of Hydrology, v. 380, no. 3–4, p. 489–496, doi:10.1016/j.jhydrol.2009.10.012.

Doherty, J.E., and Hunt, R.J., 2010, Approaches to highly parameterized inversion—A guide to using PEST for groundwater-model calibration: U.S. Geological Survey Scientific Investigations Report 2010–5169, 59 p.

Doherty, J.E., Hunt, R.J., and Tonkin, M.J., 2010, Approaches to highly parameterized inversion—A guide to using PEST for model-parameter and predictive-uncertainty analysis: U.S. Geological Survey Scientific Investigations Report 2010–5211, 71 p.

Hunt, R.J., Doherty, J., and Tonkin, M.J., 2007, Are models too simple? Arguments for increased parameterization: Ground Water, v. 45, no. 3, p. 254–262, doi:10.1111/j.1745-6584.2007.00316.x.

Hunt, R.J., Luchette, J., Schreüder, W.A., Rumbaugh, J.O., Doherty, J., Tonkin, M.J., and Rumbaugh, D.B., 2010, Using a cloud to replenish parched groundwater modeling efforts: Ground Water, v. 48, no. 3, p. 360–365, doi:10.1111/j.1745-6584.2010.00699.x.

Kalman, D., 1996, A singularly valuable decomposition—The SVD of a matrix: College Mathematics Journal, v. 27, no. 1, p. 2–23.

Larsen, R.M., 1998, Lanczos bidiagonalization with reorthogonalization: Aarhus, Denmark, Aarhus University, Computer Science Department, 90 p., accessed December 6, 2011, at *http://soi.stanford.edu/~rmunk/PROPACK/paper.pdf*.

Larsen, R.M., 2001, Combining implicit restart and partial reorthogonalization in Lanczos bidiagonalization: Stanford University, notes from a presentation given in April 2001, accessed December 6, 2011, at *http://soi.stanford.edu/~rmunk/PROPACK/talk.rev3.pdf*.

Moore, C., and Doherty, J., 2005, The role of the calibration process in reducing model predictive error: Water Resources Research, v. 41, no. 5, W05050, 14 p., doi:10.1029/2004WR003501.

Moore, C., Wöhling, T., and Doherty, J., 2010, Efficient regularization and uncertainty analysis using a global optimization methodology: Water Resources Research, v. 46, W08527, 17 p., doi:10.1029/2009WR008627.

Muffels, C.T., 2008, Application of the LSQR algorithm to the calibration of a regional groundwater flow model—Trout Lake Basin, Vilas County, Wisconsin: University of Wisconsin-Madison, Department of Geology and Geophysics, Master's thesis, 106 p.

Muffels, C.T., Schreüder, W.A., Doherty, J., Karanovic, M., Tonkin, M.J., Hunt, R.J., and Welter, D.E., 2011, GENIE—A model independent TCP/IP run manager *in* MODFLOW and More 2011—Integrated Hydrologic Modeling, Proceedings of the 10th International Conference of the International Ground Water Modeling Center: Golden, Colo., Colorado School of Mines.

Muffels, C.T., Schreüder, W.A., Doherty, J., Karanovic, M., Tonkin, M.J., Hunt, R.J., and Welter, D.E., 2012, Approaches in highly parameterized inversion—GENIE, A general model independent TCP/IP run manager: U.S. Geological Survey Techniques and Methods, book 7, section C6, 26 p.

Oreskes, N., Shrader-Frechette, K., and Belitz, K., 1994, Verification, validation, and confirmation of numerical models in the earth sciences: Science, v. 263, p. 641–646.

Paige, C.C., and Saunders, M.A., 1982a, LSQR—An algorithm for sparse linear equations and sparse least squares: ACM Transactions on Mathematical Software, v. 8, no. 1, p. 43–71.

Paige, C.C., and Saunders, M.A., 1982b, Algorithm 583LSQR—Sparse linear equations and least squares problems: ACM Transactions on Mathematical Software, v. 8, no. 2, p. 195–209.

Pilkey, O.H., and Pilkey-Jarvis, L., 2007, Useless arithmetic—Why environmental scientists can't predict the future: New York, Columbia University Press, 230 p.

Saltelli, A., Tarantola, S., Campolongo, F., and Ratto, M., 2004, Sensitivity analysis in practice—A guide to assessing scientific models: West Sussex, England, John Wiley & Sons Ltd., 219 p.

Tonkin, M.J., and Doherty, J., 2005, A hybrid regularized inversion methodology for highly parameterized models: Water Resources Research, v. 41, W10412, 16 p., doi:10.1029/2005WR003995.

Tonkin, M., and Doherty, J., 2009, Calibration-constrained Monte-Carlo analysis of highly parameterized models using subspace techniques: Water Resources Research, v. 45, no. 12, W00B10, , 17 p., doi:10.1029/2007WR006678.

Appendixes 1 through 6

Note: Although American spelling is used in the bulk of this report, British spelling is retained in some of the code, nomenclature, and examples in the appendixes.

Appendix 1: Input Instructions

The PEST++ Visual Studio Project, as well as source code and executable, are available for download at *http://pubs.usgs.gov/tm/tm7c5/*. In order to facilitate use by experienced PEST users, PEST++ adopts many of the conventions, variable names, and output formats of the original PEST (Doherty, 2010a). The intent is to make PEST++ input and output compatible with the large number of existing PEST utilities (for example, Doherty, 2011a, 2011b). However, although having a similar appearance, PEST++ does not contain all the capabilities of PEST. Most notably, the limitations of PEST++ version 1.0 are as follows:

1. Full observation covariance weights matrix is not supported.
2. Automatic adjustment of the regularization weights via the regularization mode in PEST is not supported.
3. The GENIE interface supports only a single command line. If multiple command lines are needed to run the model, then a batch file must be created.
4. Output is limited to *.rec*, *.par* and *.jco* files.
5. PEST++ does not perform a final run with the best parameters. A strategy needs to be developed to implement a final run when using GENIE (Muffels and others, 2011, 2012).
6. Jacobian *.jco* file for superparameter iterations is written in terms of the superparameters. This requires that users map this model output back to the native parameters if they wish to interrogate the contents of the Jacobian matrix in detail.

As PEST++ is developed further, it is expected that the capabilities will increase and the disparity between PEST and PEST++ will decrease.

In addition, large problems (defined as having many parameters and/or observations) will often require parallel computing. PEST++ relies on run managers to complete the forward model runs; version 1.0 provides two options. The GENIE run manager is sophisticated and capable of performing parallel runs on a single machine or over a TCP/IP-enabled network. The serial rule manager provides a simple alternative that duplicates the functionality currently in regular PEST (not Parallel PEST/PPEST). The serial run manager is the default, but the GENIE run manager can be switched on by specifying the GMAN_SOCKET variable in the control file.

The PEST Control File

For ease of reference, variables within the PEST control file are listed below, and the variables used by PEST++ are highlighted. PEST++ relies on the structure of the input file to deduce the algorithmic parameters and read only those algorithmic parameters that are absolutely necessary. For example, there is no need to read the NOBS variable because each line in the "observation data" section of the control file specifies an observation; however, it is necessary to read the NPAR variable to know where specification of parameters ends and information on tied parameters begins. This list is followed by short explanation of each variable used by PEST++.

```
pcf

* control data

RSTFLE PESTMODE

NPAR NOBS NPARGP NPRIOR NOBSGP [MAXCOMPDIM]

NTPLFLE NINSFLE PRECIS DPOINT [NUMCOM JACFILE MESSFILE]

RLAMBDA1 RLAMFAC PHIRATSUF PHIREDLAM NUMLAM [JACUPDATE] [LAMFORGIVE]

RELPARMAX FACPARMAX FACORIG [IBOUNDSTICK UPVECBEND] [ABSPARMAX]

PHIREDSWH [NOPTSWITCH] [SPLITSWH] [DOAUI] [DOSENREUSE]

NOPTMAX PHIREDSTP NPHISTP NPHINORED RELPARSTP NRELPAR [PHISTOPTHRESH] [LASTRUN]
[PHIABANDON]
ICOV ICOR IEIG [IRES] [JCOSAVE] [VERBOSEREC] [JCOSAVEITN] [REISAVEITN] [PARSAVEITN]
```

```
* automatic user intervention
MAXAUI AUISTARTOPT NOAUIPHIRAT AUIRESTITN
AUISENSRAT AUIHOLDMAXCHG AUINUMFREE
AUIPHIRATSUF AUIPHIRATACCEPT NAUINOACCEPT
* singular value decomposition
SVDMODE
MAXSING EIGTHRESH
EIGWRITE
* lsqr
LSQRMODE
LSQR_ATOL LSQR_BTOL LSQR_CONLIM LSQR_ITNLIM
LSQRWRITE
* svd assist
BASEPESTFILE
BASEJACFILE
SVDA_MULBPA SVDA_SCALADJ SVDA_EXTSUPER SVDA_SUPDERCALC SVDA_PAR_EXCL
* sensitivity reuse
SENRELTHRESH  SENMAXREUSE
SENALLCALCINT  SENPREDWEIGHT  SENPIEXCLUDE
* parameter groups
PARGPNME INCTYP DERINC DERINCLB FORCEN DERINCMUL DERMTHD [SPLITTHRESH SPLITRELDIFF
SPLITACTION]
(one such line for each of NPARGP parameter groups)
* parameter data
PARNME PARTRANS PARCHGLIM PARVAL1 PARLBND PARUBND PARGP SCALE OFFSET DERCOM
(one such line for each of NPAR parameters)
PARNME PARTIED
(one such line for each tied parameter)
* observation groups
OBGNME [GTARG] [COVFLE]
(one such line for each of NOBSGP observation group)
* observation data
OBSNME OBSVAL WEIGHT OBGNME
(one such line for each of NOBS observations)
* derivatives command line
```

```
DERCOMLINE
EXTDERFLE
* model command line
COMLINE
(one such line for each of NUMCOM command lines)
* model input/output
TEMPFLE INFLE
(one such line for each of NTPLFLE template files)
INSFLE OUTFLE
(one such line for each of NINSLFE instruction files)
* prior information
PILBL PIFAC * PARNME + PIFAC * log(PARNME) ... = PIVAL WEIGHT OBGNME
(one such line for each of NPRIOR articles of prior information)
* predictive analysis
NPREDMAXMIN [PREDNOISE]
PD0 PD1 PD2
ABSPREDLAM RELPREDLAM INITSCHFAC MULSCHFAC NSEARCH
ABSPREDSWH RELPREDSWH
NPREDNORED ABSPREDSTP RELPREDSTP NPREDSTP
* regularisation
PHIMLIM  PHIMACCEPT [FRACPHIM] [MEMSAVE]
WFINIT  WFMIN  WFMAX  [LINREG][REGCONTINUE]
WFFAC  WFTOL IREGADJ [NOPTREGADJ REGWEIGHTRAT [REGSINGTHRESH]]
* pareto
PARETO_OBSGROUP
PARETO_WTFAC_START PARETO_WTFAC_FIN NUM_WTFAC_INC
NUM_ITER_START NUM_ITER_GEN NUM_ITER_FIN
ALT_TERM
OBS_TERM ABOVE_OR_BELOW OBS_THRESH NUM_ITER_THRESH (only if ALT_TERM is non-zero)
NOBS_REPORT
OBS_REPORT_1 OBS_REPORT_2 OBS_REPORT_3..(NOBS_REPORT items)
++# This line is a comment as are all lines that begin with "++#"
++# PEST++ input is parsed using key words that can be specified in any order
++ GMAN_SOCKET(host:socket)
++ SUPER_NMAX(max_super)  SUPER_EIGTHRES(eig_thres)
++ N_ITER_BASE(base_iter)  N_ITER_SUPER(super_iter)
```

Variables in "control data" section of PEST control file.

Variable	Type	Values	Description
RSTFLE	Text	"restart" or "norestart"	Instructs PEST whether to write restart data.
PESTMODE	Text	"estimation", "prediction", "regularisation", "pareto"	PEST's mode of operation.
NPAR	Integer	greater than 0	Number of parameters.
NUMCOM	Integer	optional; greater than zero	Number of command lines used to run model.
RELPARMAX	Real	greater than 0	Parameter relative change limit.
FACPARMAX	Real	greater than 1	Parameter factor change limit.
FACORIG	Real	between 0 and 1	Minimum fraction of original parameter value in evaluating relative change.
PHIREDSWH	Real	between 0 and 1	Sets objective function change for introduction of central derivatives.
NOPTMAX	Integer	−2, −1, 0, or any number greater than 0	Number of optimization iterations.
PHIREDSTP	Real	greater than 0	Relative objective function reduction triggering termination.
NPHISTP	Integer	greater than 0	Number of successive iterations over which PHIREDSTP applies.
NPHINORED	Integer	greater than 0	Number of iterations since last drop in objective function to trigger termination.
RELPARSTP	Real	greater than 0	Maximum relative parameter change triggering termination.
NRELPAR	Integer	greater than 0	Number of successive iterations over which RELPARSTP applies.

Variables in optional "singular value decomposition" section of PEST control file.

Variable	Type	Values	Description
MAXSING	Integer	greater than 0	Number of singular values at which truncation occurs.
EIGTHRESH	Real	0 or greater, but less than 1	Eigenvalue ratio threshold for truncation.
EIGWRITE	Integer	0 or 1	Determines content of SVD output file.

Variables required for each parameter group in "parameter groups" section of PEST control file.

Variable	Type	Values	Description
PARGPNME	Text	12 characters or less	Parameter group name.
INCTYP	Text	"relative", "absolute", "rel_to_max"	Method by which parameter increments are calculated.
DERINC	Real	greater than 0	Absolute or relative parameter increment.
DERINCLB	Real	0 or greater	Absolute lower bound of relative parameter increment.
FORCEN	Text	"switch", "always_2", "always_3", "switch_5", "always_5"	Determines whether central derivatives calculation is undertaken and whether three points or four points are employed in central derivatives calculation.
DERINCMUL	Real	greater than 0	Derivative increment multiplier when undertaking central derivatives calculation.
DERMTHD	Text	"parabolic", "outside_pts", "best_fit", "minvar", "maxprec"	Method of central derivatives calculation.

Variables required for each parameter in "parameter data" section of PEST control file.

Variable	Type	Values	Description
PARNME	Text	12 characters or less	Parameter name.
PARTRANS	Text	"log", "none", "fixed", "tied"	Parameter transformation.
PARCHGLIM	Text	"relative", "factor", or absolute(n)	Type of parameter change limit.
PARVAL1	Real	any real number	Initial parameter value.
PARLBND	Real	less than or equal to PARVAL1	Parameter lower bound.
PARUBND	Real	greater than or equal to PARVAL1	Parameter upper bound.
PARGP	Text	12 characters or less	Parameter group name.
SCALE	Real	any number other than 0	Multiplication factor for parameter.
OFFSET	Real	any number	Number to add to parameter.
DERCOM	Integer	0 or greater	Model command line used in computing parameter increments.
PARTIED	Text	12 characters or less	The name of the parameter to which another parameter is tied.

Variables required for each observation group in "observation groups" section of PEST control file.

Variable	Type	Values	Description
OBGNME	Text	12 characters or less	Observation group name.

Variables required for each observation in "observation data" section of PEST control file.

Variable	Type	Values	Description
OBSNME	Text	20 characters or less	Observation name.
OBSVAL	Real	any number	Measured value of observation.
WEIGHT	Real	0 or greater	Observation weight.
OBGNME	Text	12 characters or less	Observation group to which observation assigned.

Variables in "model command line" section of PEST control file.

Variable	Type	Values	Description
COMLINE	Text	system command	Command to run model.

Variables in "model input/output" section of PEST control file.

Variable	Type	Values	Description
TEMPFLE	Text	a filename	Template file.
INFLE	Text	a filename	Model input file.
INSFLE	Text	a filename	Instruction file.
OUTFLE	Text	a filename	Model output file.

Variables in "prior information" section of PEST control file.

Variable	Type	Values	Description
PILBL	Text	20 characters or less	Name of prior information equation.
PIFAC	Text	real number other than 0	Parameter value factor.
PARNME	Text	12 characters or less	Parameter name.
PIVAL	Real	any number	"Observed value" of prior information.
WEIGHT	Real	0 or greater	Prior information weight.
OBGNME	Text	12 characters or less	Observation group name.

Variables in optional "regularization" section of PEST control file.

Variable	Type	Values	Description
PHIMLIM	Real	greater than 0	Target measurement objective function.
PHIMACCEPT	Real	greater than PHIMLIM	Acceptable measurement objective function.
FRACPHIM	Real	optional; 0 or greater, but less than 1	Set target measurement objective function at this fraction of current measurement objective function.
MEMSAVE	Text	"memsave" or "nomemsave"	Activate conservation of memory at cost of execution speed and quantity of model output.
WFINIT	Real	greater than 0	Initial regularization weight factor.
WFMIN	Real	greater than 0	Minimum regularization weight factor.
WFMAX	Real	greater than WFMIN	Maximum regularization weight factor.
LINREG	Text	"linreg" or "nonlinreg"	Informs PEST that all regularization constraints are linear.
REGCONTINUE	Text	"continue" or "nocontinue"	Instructs PEST to continue minimizing regularization objective function even if measurement objective function is less than PHIMLIM.
WFFAC	Real	Greater than 1	Regularization weight factor adjustment factor.
WFTOL	Real	Greater than 0	Convergence criterion for regularization weight factor.
IREGADJ	integer	0, 1, 2, 3, 4 or 5	Instructs PEST to perform inter-regularization group weight factor adjustment, or to compute new relative weights for regularization observations and prior information equations.
NOPTREGADJ	integer	1 or greater	The optimization iteration interval for recalculation of regularization weights if IREGADJ is 4 or 5.
REGWEIGHTRAT	Real	absolute value of 1 or greater	The ratio of highest to lowest regularization weight; spread is logarithmic with null space projection if set negative.
REGSINGTHRESH	Real	less than 1 and greater than 0	Singular value of $\mathbf{x}^t\mathbf{q}\mathbf{x}$ (as factor of highest singular value) at which use of higher regularization weights commences if IREGADJ is set to 5.

PEST++ Additions to the PEST Control File

Information in the PEST control specific to PEST++ is specified on lines starting with "++". Although the previous example places all the PEST++ input in a single section at the end of the PEST control file, this is not a requirement. This information does not need to be contiguous and can reside anywhere in the PEST control file. Lines starting with "++#" are considered comments and are ignored.

Unlike the rest of the PEST control file, PEST++ uses keywords rather than location to specify variables. Lines are parsed using the space, tab, and parenthesis characters as separators. The example uses parentheses to more clearly delineate the values assigned to the variable, but these could just as well be replaced by white spaces. The following table includes a listing and explanation of the permissible PEST++ keywords.

Variable	Type	Values	Description
GMAN_SOCKET	Text	character string containing host and port separated by ":"	Socket of the GENIE GMAN run manager. The socket contains the hostname and port of the GMAN run manager that will be used to make the model runs. For example, if GMAN is running on the computer "my_computer" listening to port 24772, then this variable should be specified as my_computer:24772.
N_ITER_BASE	Integer	1 or greater	Number of base parameter iterations performed for each superparameter iteration.
N_ITER_SUPER	Integer	0 or greater	Number of superparameter iterations performed for each base parameter iteration.
SUPER_EIGTHRES	Real	any positive number (typically should be greater than 1.0e−7)	PEST++ will not include any superparameters whose ratio with the largest superparameter is less than this ratio. This value can as small as zero if the user wants to specify the number of superparameters solely with SUPER_NMAX. Because PEST++uses SVD on the superparameter problem, a low value for this SUPER_EIGTHRES will not adversely impact the stability of the solution.
SUPER_NMAX	Integer	integer between 1 and the minimum of maximum number of parameters and the maximum number of observations	Maximum number of superparameters to use in the superparameter iterations.

References

Doherty, J., 2011a, PEST surface water utilities: Brisbane, Australia, Watermark Numerical Computing.

Doherty, J., 2011b, Groundwater data utilities: Brisbane, Australia, Watermark Numerical Computing.

Muffels, C.T., Schreüder, W.A., Doherty, J., Karanovic, M., Tonkin, M.J., Hunt, R.J., and Welter, D.E., 2011, GENIE—A model independent TCP/IP run manager *in* MODFLOW and More 2011—Integrated Hydrologic Modeling, Proceedings of the 10th International Conference of the International Ground Water Modeling Center: Golden, Colo., Colorado School of Mines.

Muffels, C.T., Schreüder, W.A., Doherty, J., Karanovic, M., Tonkin, M.J., Hunt, R.J., and Welter, D.E., 2012, Approaches in highly parameterized inversion: GENIE, a general model independent TCP/IP run manager: U.S. Geological Survey Techniques and Methods, book 7, section C6, 27 p.

Appendix 2: PEST++ Elementary Transformations

Transformation sequences are groups of (elementary) transformations. PEST++ uses the transformation sequences "control file to numeric" and "control file to model" to convert between parameter states. These sequences are composed of the elementary transformations that support forward and reverse transformations and contain the "one-to-one" attribute to indicate whether they are compatible with parameter back substitution.

TranScale

TranScale provides scaling or multiplication by a fixed value. The forward transformation is multiplication, and the reverse transformation is division. TranScale is one-to-one.

TranOffset

TranOffset provides an offset or addition by a fixed value. The forward transformation is addition, and the reverse transformation is subtraction. TranOffset is one-to-one.

TranFixed

TranFixed provides a transformation for fixing values. The forward transformation adds an additional parameter with a fixed value, and the reverse transformation removes the parameter. TranFixed is considered to be a one-to-one transformation because it is compatible with parameter back-substitution.

TranFrozen

TranFrozen provides a transformation for freezing values. The forward transformation adds an additional parameter that is assigned the frozen value specified by this transformation, and the reverse transformation removes the parameter. The functionality of TranFrozen is identical to that of TranFixed. It has been implemented as an independent transformation for organization and tracking purposes. Like TranFixed, TranFrozen is considered to be a one-to-one transformation because it is compatible with parameter back-substitution.

TranTied

TranTied provides a transformation that ties the value of one parameter to that of another parameter. The forward transformation adds an additional parameter that is assigned a value such that the ratio between the new parameter to the value of the parameter it is tied to is maintained at the level specified in transformation. The inverse transformation removes the tied parameter from the parameter set. TranTied is considered a one-to-one transformation because it is compatible with parameter back substitution.

TranLog10

TranLog10 provides a base 10 logarithmic transformation. The forward transformation is the base 10 logarithm, and the inverse transformation is the exponential. TranLog10 is one-to-one.

TranSVD

TranSVD provides a transformation between superparameters and base parameters. The forward transformation maps base parameters to superparameters, and the inverse transformation maps superparameters to base parameters. This transformation is used to implement the functionality of PEST's SVD-Assist internally in PEST++. Additional details on SVD-Assist and this approach are provided in the next section. TranSVD is not one-to-one.

TranNorm

TranNorm provides a transformation that automatically normalizes the parameters based on the assumption that the parameter range specified for each parameter in the input control file is indicative of its variance. The forward transformation divides each parameter by its variance, and the inverse transformation multiplies each parameter by its variance. TranNorm is one-to-one.

Appendix 3: The Marquardt Lambda and SVD Rotation Factor Supporting Theory

The Marquardt lambda is typically a component of the Gauss-Marquardt-Levenberg method, but its use in PEST has been extended to supplement SVD. In the Gauss-Marquardt-Levenberg method, the Marquardt lambda is a weighting factor that interpolates between the Gauss-Newton method and the method of gradient descent. The Gauss-Newton solution for the nonlinear weighted least-squares problem can be written as

$$u = \left(J^T Q J\right)^{-1} J^T Q r \tag{1}$$

where is **u** the upgrade vector, **J** is the Jacobian, **Q** is the observation weights matrix and **r** is a vector containing the residuals of the observations. The matrix $\mathbf{J^TQJ}$ in equation 1 is commonly referred to as the "normal equations matrix." A full derivation is provided in Doherty (2010a).When the Marquardt lambda is added this equation becomes

$$u = \left(J^T Q J \pm \lambda I\right)^{-1} J^T Q r \tag{2}$$

where **I** is the identity matrix and λ is the Marquardt lambda. From equation 2, it is apparent that the Marquardt lambda adds terms to the diagonal of the normal equation matrix being inverted. Adding large terms to the diagonal of a matrix tends to make it better conditioned and provides a de facto form of regularization. However, this is not a desirable regularization strategy because it does not restrain the solution in a physically meaningful way—contrary to Tikhonov regularization (which identifies one or more preferred conditions) or truncated SVD (which minimizes changes for parameters whose influence on the inverse problem is overwhelmed by noise in the solution). So, although use of the Marquardt lambda regularization is not recommended, it can nonetheless make an ill-posed problem solvable and is often unknowingly used in this capacity by new users.

Two of the Marquardt lambda's roles have been discussed: rotating the solution vector in the direction of the gradient descent solution and providing a de facto form of regularization for ill-posed problems. In the authors' view, it is also commonly used to add a more robust search capability to the nonlinear least-squares solution. Testing different values of the Marquardt lambda during each iteration allows PEST to explore a larger portion of parameter space, helping to prevent it from becoming trapped in a local minimum. When this technique is used in conjunction with parallel PEST, the impact on the overall runtime is minimal because parallelization of testing parameter upgrades computed using different values of the Marquardt lambda allows these runs to be made with no increase in overall clock time. Because SVD incorporates subspace regularization, it does not benefit from the Marquardt lambda's ability to provide suboptimal regularization; however, SVD can greatly benefit from the addition of a more robust search capability to help avoid local optima, which is why the Marquardt lambda and SVD are commonly used together in PEST.

Although PEST++ does not need to support the Gauss-Marquardt-Levenberg method, it still needs to provide an analogous functionality to a widely varying Marquardt lambda in the SVD solution. For large problems, performing an SVD factorization can be computationally expensive, and equation 2 requires that a SVD factorization be performed for each value of the Marquardt lambda because it is contained within the normal equation matrix being inverted. However, because SVD is unconditionally stable, the role of the Marquardt lambda can be reduced from adding a diagonal term to merely providing a means to rotate the upgrade vector in the direction of the gradient descent solution. To take advantage of this simplification, PEST++ replaces the Marquardt lambda with a newly defined rotation factor. The equation for the upgrade vector can be derived by using SVD and defining a unit vector pointing in the same direction to yield to following equations:

$$u_{svd} = \left(Q^{1/2} J\right)^{-} Q^{1/2} r \tag{3}$$

$$\hat{u}_{svd} = \frac{u_{SVD}}{\left\|u_{SVD}\right\|} \tag{4}$$

where the symbol "-" denotes the generalized inverse and $\|u_{SVD}\|$ denotes the L2 norm of the upgrade vector. Similarly, the upgrade vector for the direction method of gradient descent and its associated unit vector can be expressed as

$$u_{gd} = -2J^T Q r \tag{5}$$

$$\hat{u}_{gd} = \frac{u_{gd}}{\left\|u_{gd}\right\|} \tag{6}$$

Equations 4 and 6 can be used to rotate equation 3 in the direction of the gradient descent solution to produce equation 7:

$$u = \left[(1-\alpha)\hat{u}_{SVD} + \alpha\hat{u}_{gd}\right] \frac{\|u_{SVD}\|}{\left\|(1-\alpha)\hat{u}_{SVD} + \alpha\hat{u}_{gd}\right\|} \tag{7}$$

where α is the rotation factor and **u** is the rotated upgrade vector. This technique avoids SVD factorization for each value of lambda, and its physical meaning is very clear. In addition, the rotation factor must lie in the interval [0, 1], whereas there is no clear upper limit on Marquardt lambda.

Appendix 4: Considerations for Code Development

Design Goals

The initial goal of PEST++ development was to build a basic framework and create a program that makes the powerful features of PEST accessible to more users and developers while maintaining a robust and efficient design. In particular, the following items were identified as important.

1. *Portability.*—It is important that a well-established standard language be used and that the use of external dependencies be minimized.

2. *Efficiency.*—Because this program will be used for large problems, it must be efficient in two regards. First, it must be computationally efficient; and second, it must be efficient in its use of memory and maintain a "small memory footprint." In particular, it must be possible to store and access nonstandard sparse information efficiently.

3. *Extensibility.*—The code must incorporate a modular or object-oriented design that promotes code extensibility and reuse.

4. *Ease of use.*—The code should automate processes and depend on default parameters when possible, thereby requiring as little user input as possible while providing the user with a simple interface for the required input data. Providing a more seamless interface to superparameters was identified as a key objective, given that users sometimes struggle with maintaining proper intermediate files and modified control file and batch files when using SVD-Assist in PEST. Also included in ease of use is support for robust error checking and handling.

Language Selection

Selecting a language in which to build PEST++ was not obvious. During the pseudocode process, it became apparent that no single programming language was optimum in all aspects for fully attaining the design goals. Much of the initial development of PEST++ was done in Python to take advantage of its concise syntax and friendly development environment. Use of Python at the outset allowed for rapid development and testing of an initial prototype. However, it became apparent that handling many of the sparse data structures efficiently would be difficult if using Python's standard data types and packages. To avoid these issues and ensure that adequate options will be available to resolve performance issues as they arise, the code was migrated to C++. C++ was chosen for this project because it offers the following benefits: (1) It is a mature language, governed by ISO standards. (2) It is widely available on many different platforms. (3) The standard library that is included in the standard language provides a rich set of tools, including the standard template library (STL). (4) C++ includes exception handling, which can handle usage errors in an efficient manner. (5) An experienced programmer can develop fast and efficient code. (6) It is relatively easy to produce statically linked executables, which are easy to distribute.

External PEST++ dependencies have been limited to BLAS, LAPACK, LAPACK++ and PROPACK, where LAPACK++ is an object-oriented wrapper for the BLAS and LAPACK linear algebra libraries and PROPACK is an iterative solver for computing SVD factorizations.

Integration in Visual Studio Integrated Development Environment (IDE)

All the source code required to build PEST++ has been consolidated in the Microsoft Visual Studio 2010 IDE. This includes the source codes for (1) PEST++, (2) the GENIE interface, (3) LAPACK++, and (4) the PEST Fortran code that processes template and insertion files. The project is configured to build a statically linked PEST++ executable without any external dependencies. This simplifies and facilitates sharing the source code and distributing the executables. The PEST++ Visual Studio Project, as well as source code and executable, are available for download at *http://pubs.usgs.gov/tm/tm7c5/*.

Naming Convention

The naming convention used for PEST++ code is summarized in table 4–1 and follows that of the standard template library. Although PEST++ code follows this convention, the LAPACK++ library used by PEST++ adheres to a different naming convention.

Table 4–1. PEST++ naming convention.

Items	Convention	Example
Class names	Camel-case. Names start with capitals and new words are delineated by capitals.	Class MyClass
Class instances Class methods Variables Functions	Names start with lower case and boundaries are delineated with "_".	MyClass my_class void MyClass.do_something(int a) double my_value int my_function(int a)

Inheritance and Polymorphism in PEST++

Polymorphism is an object-orientated programming feature that allows data types sharing a common interface to be used interchangeably within a program. PEST++ uses polymorphism based on inheritance to define interfaces for run managers, elemental transformations, and SVD solutions. In the case of the run manager, this shields the PEST++ code from having to specify which run manager is being used and makes the code compatible with any run manager that conforms to the ModelRunManagerAbstract base class. Figures 4–1, 4–2, and 4–3 show the inheritance trees for the run manager, elementary transformations, and the SVD-based solution.

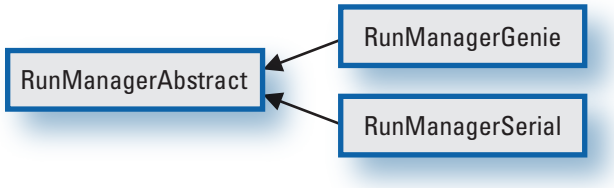

Figure 4–1. Run manager inheritance tree.

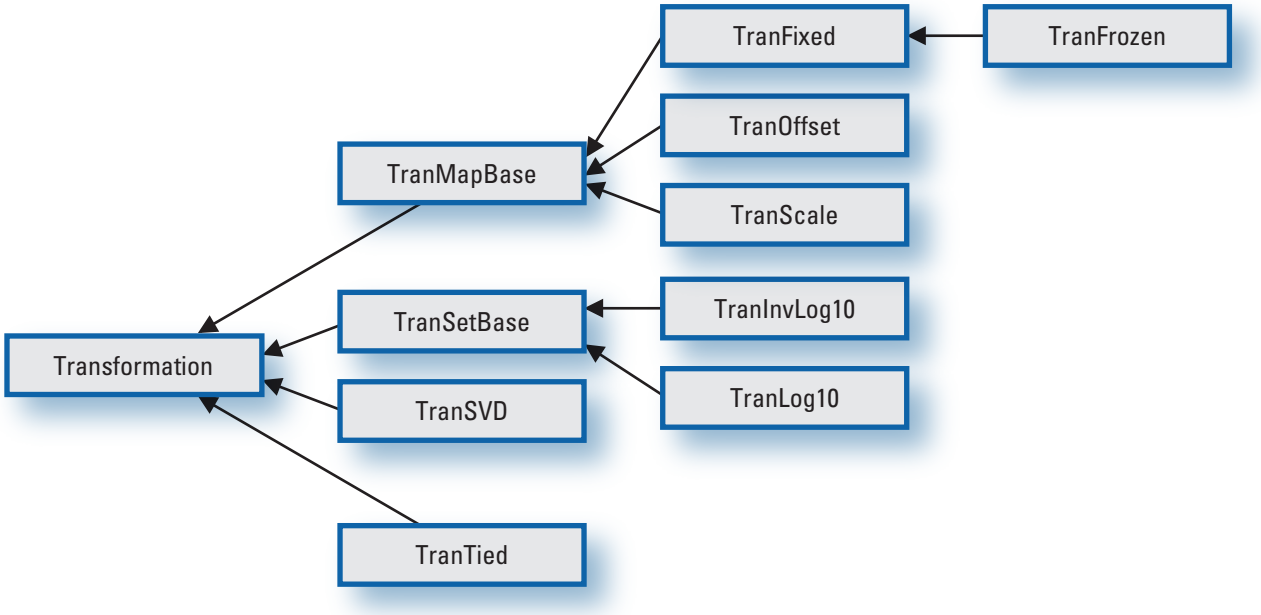

Figure 4–2. Elemental transformation inheritance tree.

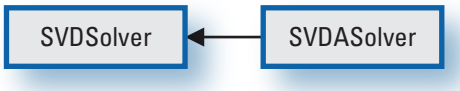

Figure 4–3. SVD Solution inheritance tree.

Overall Program Flow and Design

The following flowcharts (figs. 4–4 through 4–6) describe the overall flow of PEST++, SVDSolver::solve, and SVDSolver::iteration.

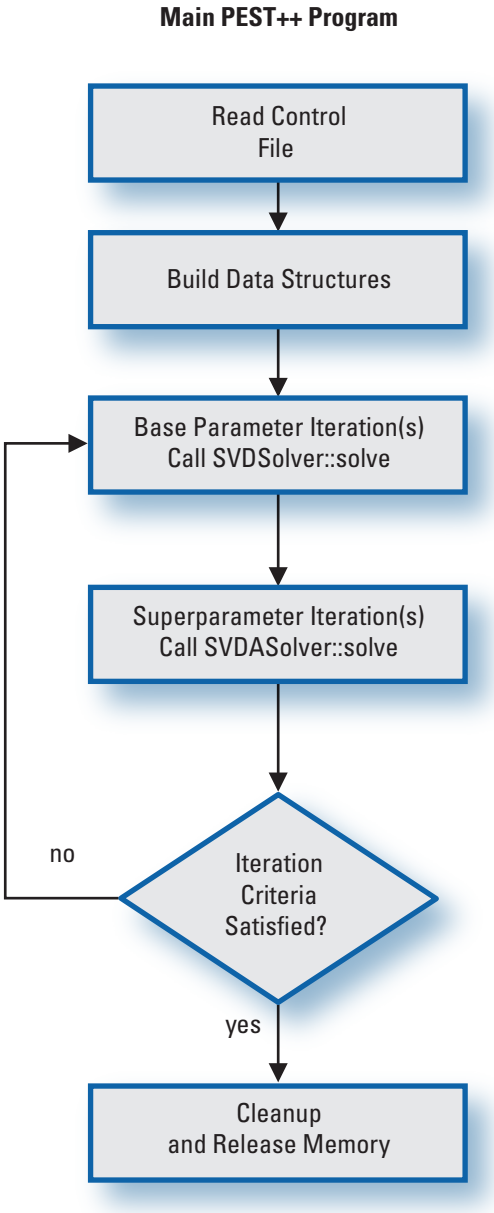

Figure 4–4. Flowchart for main PEST++ program.

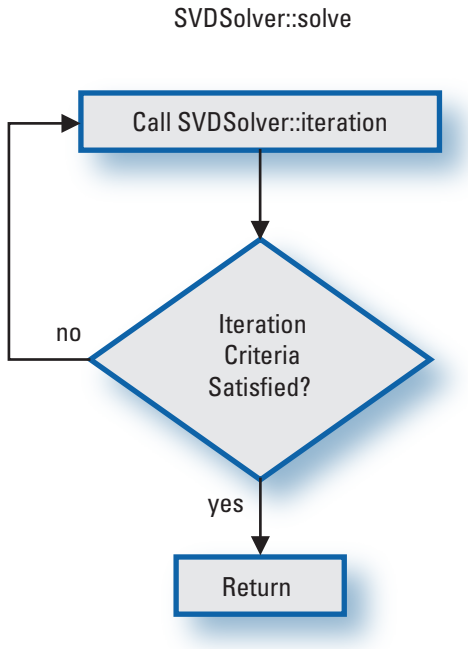

Figure 4–5. Flowchart for SVDSolver::solve.

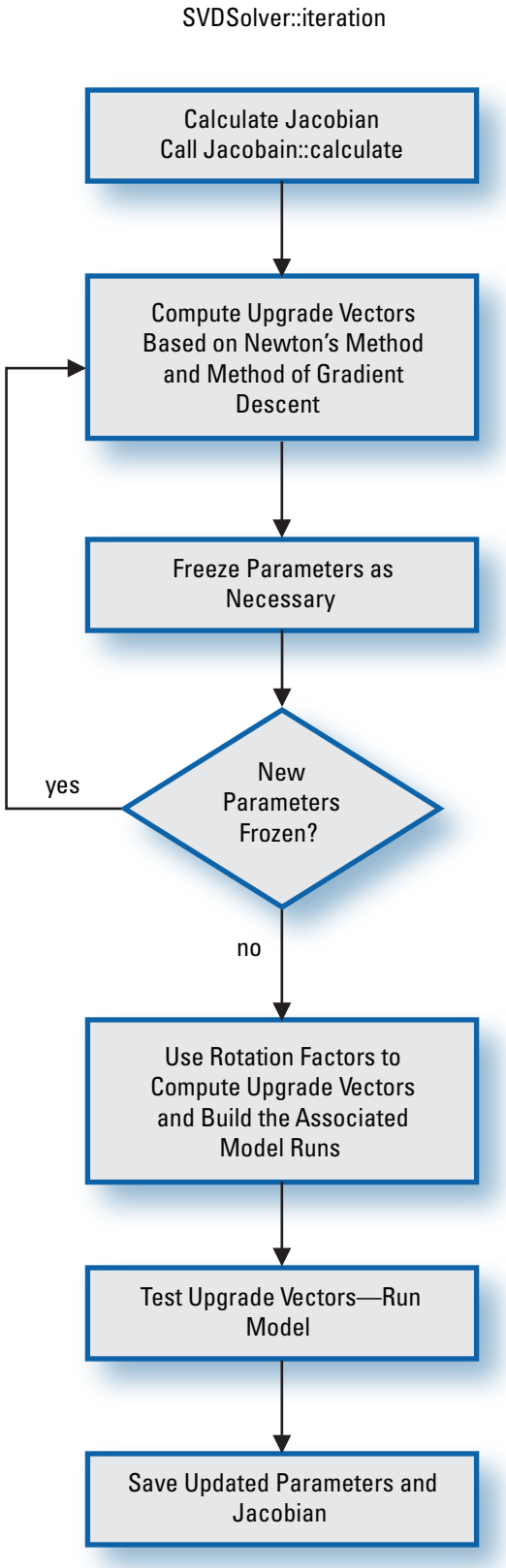

Figure 4–6. Flowchart forSVDSolver::iteration.

Appendix 5: Class List

FileManager

This class encapsulates the I/O filenames and iostream handles associated with a PEST++ run. It has a copy of the pathname in which the PEST++ simulation is running and a reference to the file stream associated with the record file. The rest of the class in PEST++ program relies on this class to supply the iostream handle to the record file, as well as the appropriate names of all PEST++ input and output files.

Jacobian

This class provides support for building and accessing the Jacobian. It includes code for calculating derivatives and provides support for forward difference, central difference(outer), and central difference(parabolic), as well as the ability to switch to from forward derivatives to central derivatives as the optimization process slows down. This class uses the GENIE interface to perform the actual model runs and is able to return the Jacobian as a LAPACK++ LaGenMatDouble matrix that can be used in numerical computations and save the Jacobian to disk PEST's *.jco* data format.

ModelExecInfo

This class contains the commands and filenames required to make a model run. It encapsulates the names of the template files, input files, insertion files, and output files. In addition, it also contains the model command lines necessary to start a model run.

ModelRun

ModelRun is a child of ModelRunAbstractBase and is designed to consolidate the information associated with a model run. Because ModelRunAbstractBase is an abstract class, it cannot be directly instantiated. ModelRun is the primary child class that is used most often to store the parameter and simulated observations associated with a model run.

ModelRunAbstractBase

This is the abstract base class for all classes associated with a model run. These classes are designed to consolidate the information associated with a model run, which includes the parameters, transformations, and the observed values, as well as their simulated counterparts. In addition, these classes also contain the information necessary to compute the objective function and contain methods that perform this task. Although ModelRunAbstractBase cannot be instantiated, it defines the methods that all classes derived from it must implement.

ModelRunShallowCopy

This is a lightweight copy of a ModelRun class. It contains its own set of parameters and simulated observations, but it uses the transformations associated with the ModelRun class it was created from.

ObjectiveFunc

This class handles all aspects of the objective function. It contains the measured observations and prior information, along with all the associated in information such as weights and groups, and is able to calculate the objective function and produce a report that includes the contributions to the objective function of all of all the observation groups.

ObservationGroupRec

This class stores the information associated with an observation group.

ObservationInfo

This class compliments the Observations class. Because the Observations class was designed to be very lightweight, it only stores the measured values of the observation. The ObservationInfo class uses an unorder_map of the ObservationRec class to store the rest of the information associated with the observations, which includes their weights and the groups that they are associated with.

ObservationRec

This class stores the weight and observation groups associated with an observation.

Observations

This class stores the measured values of the observations. It is a child of the Transformable class and is compatible the transformations used for the parameters, but the current code does not make use of this functionality.

OperSys

This class encapsulates all the features and parameters that are operating-system dependent. These include the character used to separate the different components of a pathname and the end-of-line or carriage-return character(s).

ParameterGroupInfo

This class stores the information associated with the parameter groups.

ParameterGroupRec

This class stores the information associated with a parameter group.

ParameterInfo

This class compliments the Parameters class. Because the Parameters class was designed to be lightweight, it stores only the parameter values. The ParameterInfo class uses an unorder_map of the ParameterRec class to store the rest of the information associated with parameter, which includes their bounds, change limit methodology, and the groups they are associated with.

ParameterRec

This class stores the bounds, change limit methodology, and the group for a parameter.

Parameters

This class stores parameter values. It is a child of the Transformable class and is designed be compatible the ParamTransformSeq class.

ParamTransformSeq

This class handles transformations for the parameters. It contains the transformation sequences to convert between control and numeric parameters, as well as those required to convert between control and model parameters.

Pest

This class consolidates the information contained in the control file and contains instances of the following classes: ControlInfo, SVDInfo, Parameters, ParameterInfo, ParameterGroupInfo, BaseGroupInfo, Observations, ObservationInfo, PriorInformation, ModelExecInfo, PestppOptions, and ParamTransformSeq.

PestConversionError, PestError, PestFileError, PestIndexError, PestParsingError

These classes define error states and are used to throw exceptions.

PestppOptions

This class stores the PEST++ input options that are not available in PEST.

PIAtom

This class is used by the PriorInformationRec class. It stores information associate with a single parameter in a prior information expression. This information includes the name of the parameter, whether the parameter is log transformed, and the factor by which the parameter is to be multiplied in the prior information expression.

PriorInformation

This class contains all for the prior information records associated with a PEST++ simulation.

PriorInformationRec

This class stores a complete prior information record and contains a method to compute that item's residual and contribution to the objective function. Each prior information equation in the control file is stored by using an instance of this class.

QSqrtMatrix

This class contains the weights matrix for the observations and the prior information. It contains methods that return the product of this matrix with a LAPACK++ LaGenMatDouble matrix.

RunManagerGenie

This class is a wrapper for the GENIE run manager. GENIE is a generic parallel run manager that uses TCP/IP for communication and allows runs to be performed over the Internet in remote locations. This class converts PEST++ data structures to GENIE data structures and calls GENIE to make the model runs.

SVDASolver

This class is a child of the SVDSolver class that is specialized to work with superparameters. The changes needed to accommodate superparameters are minimal; the only methods that are modified are the method that limits parameter upgrades, limit_parameters_ip, and the method that freezes parameters, freeze_parameters.

SVDInfo

This class stores the information in the SVD section of the control file.

SVDSolver

This class solves the least-squares problem by using singular value decomposition (SVD). The bulk of the work is performed in the iteration method, which sets up and solves a single iteration. This class relies on the Jacobian class to compute the Jacobian.

TerminationController

This class manages the termination criteria for the least-squares solution.

TranFixed

This class implements a transformation for fixed values.

TranFrozen

This class implements a transformation for frozen values.

TranLog10

This class implements a logarithmic transformation.

TranMapBase

This is a base class for transformations built around the standard template library (STL) map class.

TranOffset

This class implements the offset transformation.

TranScale

This class implements a scaling transformation.

TranSetBase

This is a base class for transformation that use the STL Set container to store a list of items.

Transformable

This is an abstract base class that provides compatibility with the Transformation class and all of the class derived from it. The Parameters and Observation classes are derived from this class; however, at present, only the Parameters class makes use of the functionality it provides.

TransformableValueError

This class defines an error state and is used to throw an exception.

Transformation

This is the abstract base class for all transformations. It cannot be instantiated, but it defines the methods that all of its children must possess.

TranSVD

The Transformation implements SVD-A or a transformation between superparameters and base parameters.

TranTied

This transformation ties one parameter to another. The tied parameter is maintained at a fixed ratio to the parameter it is tied to.

Appendix 6: Simple Storage Model Example

Introduction

This example demonstrates the use of PEST++ with a simple storage model developed by John Doherty as a workshop problem for use in PEST training classes. The brief description contained herein is taken from that exercise. The outputs of both PEST and PEST++ are presented so that users can review and verify the results.

Description of the Model

Figure 6–1 shows a simple storage. The storage is filled with a porous medium of storage coefficient *S*. The storage receives water as recharge at a constant rate *R*. Water is able to drain from the storage at a rate that is proportional to the head of water in the storage. Rate of water outflow is thus given by

$$q = Kh \tag{1}$$

where

q is the outflow rate,
h is the head of water in the storage, and
K is the conductance of the storage outlet. Initial head in the storage is designated as h_1.

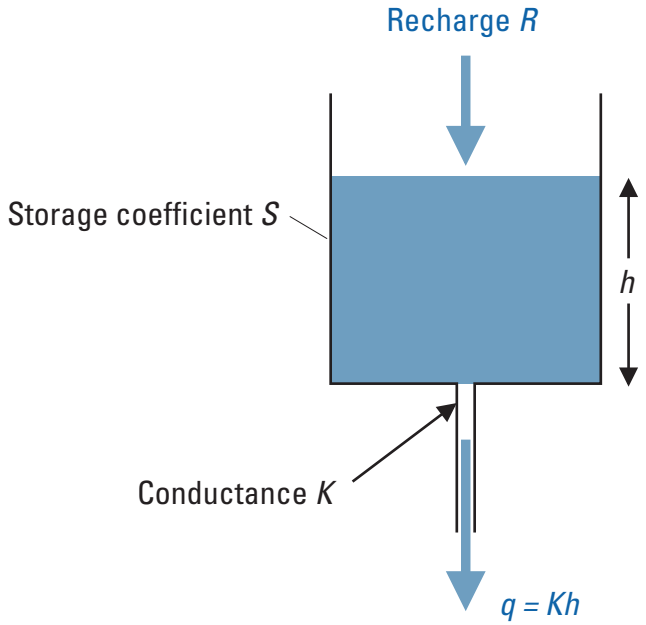

Figure 6–1. Graphical depiction of a model of a storage volume filled with a porous medium.

The rate of change in the amount of water held in storage at any time is equal to the difference between inflow and outflow. Mathematically, this is expressed by the equation

$$S\frac{dh}{dt} = R - Kh \tag{2}$$

For constant *R*, the solution of equation 2 is

$$h = h_1 + \left(\frac{R}{K} - h_1\right)\left(1 - e^{\frac{-Kt}{S}}\right) \tag{3}$$

where, as mentioned previously, h_1 is the head in the storage when *t* (the elapsed time) is zero. It is apparent from equation 3 that the equilibrium storage water level is given by

$$h = \frac{R}{K} \tag{4}$$

where *h* in equation 4 is the water level in the storage at which recharge inflow is exactly balanced by drainage outflow, the latter being given by equation 1.

When *t* is small, equation 3 asymptotically approaches the equation

$$h = h_1 + \left(\frac{R}{K} - h_1\right)\frac{Kt}{S} \tag{5}$$

If the initial head is zero, equation 5 becomes

$$h = \frac{Rt}{S} \tag{6}$$

It is apparent from equations 3 to 6 that when a new recharge regime is introduced to the storage, the level of water in the storage changes to its new level in a manner that is linear at first but, with time, assymptotically approaches its final level. The time constant pertaining to the water-level adjustment process is given by the ratio of *S* to *K*. Notice that the recharge rate *R* does not figure in calculation of the time constant and that the storage coefficient *S* is not represented in the equation for the final, equilibrium water level in the storage.

Figure 6–2 shows the variation of water level in the storage with time under the assumption that the initial water level is 20 units, the recharge rate is 10^{-1} units, the outlet conductance is 10^{-3} units, and the storage coefficient is 0.2. From equation 4, it is easily established that the final equilibrium water level in the storage under these conditions is 100 units.

The basic storage unit pictured in figure 6–1 forms an important building block of many lumped-parameter environmental models. Hence, lessons learned in this practical session will be applicable in many circumstances where much more complex environmental models are deployed to simulate the behavior of natural systems.

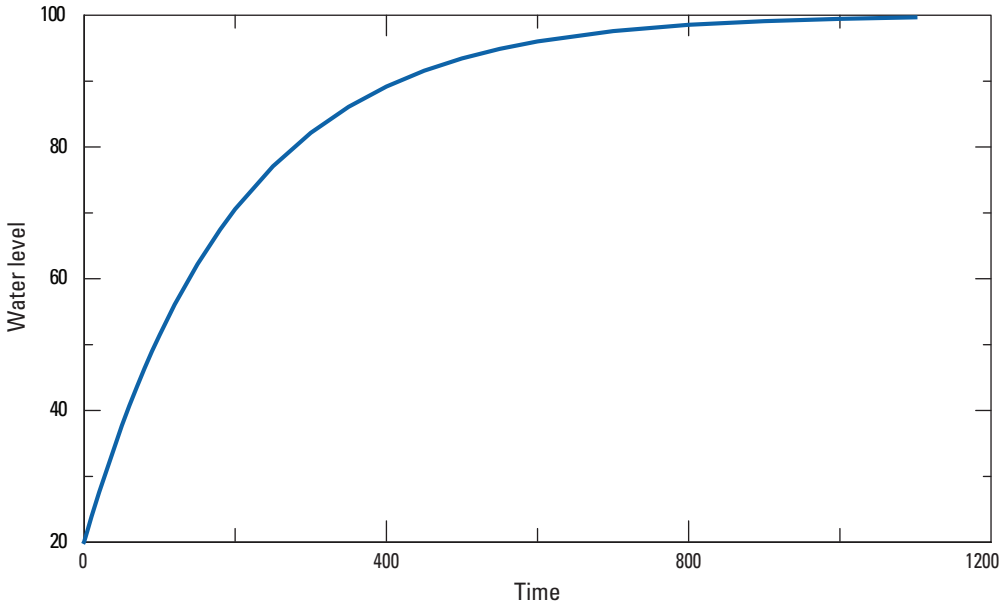

Figure 6–2. Water level in the storage depicted in figure 6–1; see text for details.

A Computer Program to Simulate the Storage

Operation of the storage depicted in figure 6–1 can be simulated by using a simple computer program based on equation 3. Such a program, written in Fortran, is listed in figure 6–3. See also file *storage.for* in the working directory for this session.

```
program storage

implicit none

real recharge, conductance, storage, inithead, time,
     +coeff, factor

open(unit=10,file='input.dat',status='old')
open(unit=20,file='output.dat')
read(10,*) recharge, conductance, storage
read(10,*) inithead
read(10,*)

coeff=(recharge/conductance-inithead)
factor=conductance/storage
write(20,10)
10    format('   Time           Water_Level')
do
read(10,*,end=100) time
write(20,20) time,inithead+coeff*(1.0-exp(-factor*time))
20      format(1x,1pg14.7,2x,1pg14.7)
end do
100   end
```

Figure 6–3. A program to compute storage water levels using equation 3.

Program STORAGE reads an input file named *input.dat*. The first line of this file should contain, in order, the recharge, outlet conductance, and storage coefficient of the storage. The next line should contain the initial storage water level. A blank line should follow that, followed by a listing of the times at which water-level computation is required. A typical input file for the STORAGE program is illustrated in figure 6–4. STORAGE writes an output file called *output.dat*. Figure 6–5 shows the *output.dat* file corresponding to the *input.dat* file depicted in figure 6–4.

```
1.0e-1  1.0e-3  0.2   / recharge conductance storage_coefficient
0.0                   / initial head

0.1                   / elapsed times
0.2
0.5
1.0
2.0
5.0
10.0
20.0
50.0
100.0
200.0
500.0
1000.0
2000.0
5000.0
10000.0
```

Figure 6–4. An input file for program STORAGE.

```
    Time           Water_Level
0.1000000       4.9987502E-02
0.2000000       9.9950016E-02
  0.5000000       0.2496878
   1.000000       0.4987521
   2.000000       0.9950166
   5.000000        2.469009
   10.00000        4.877058
   20.00000        9.516258
   50.00000        22.11992
   100.0000        39.34694
   200.0000        63.21206
   500.0000        91.79150
   1000.000        99.32620
   2000.000        99.99546
   5000.000        100.0000
   10000.00        100.0000
```

Figure 6–5. An output file written by program STORAGE.

PEST++ Model Run

The example problem can be run with PEST++ by typing the command

Pest++ storage5

The output written to the screen by PEST++ and the contents of the *storage5.rec*output file are presented below.

PEST++ ver. 1.0 Output Written to the Screen

```
C:\Users\dwelter\examples\stor>..\pest++ storage5
PEST++ Version 1.0.0

using control file: “storage5”

initializing serial run manager


OPTIMISATION ITERATION NUMBER: 1
  Iteration type: base parameter solution
calculatingjacobian... (3/3 runs complete)
testing upgrade vectors... (7/7 runs complete)
  Starting phi = 594.589;  ending phi = 22.7623  (3.82825% starting phi)

OPTIMISATION ITERATION NUMBER: 2
  Iteration type: base parameter solution
calculatingjacobian... (2/2 runs complete)
testing upgrade vectors... (7/7 runs complete)
  Starting phi = 22.7623;  ending phi = 14.9341  (65.6089% starting phi)

OPTIMISATION ITERATION NUMBER: 3
  Iteration type: base parameter solution
calculatingjacobian... (2/2 runs complete)
testing upgrade vectors... (7/7 runs complete)
  Starting phi = 14.9341;  ending phi = 0.749557  (5.01909% starting phi)

OPTIMISATION ITERATION NUMBER: 4
Iteration type: base parameter solution
calculatingjacobian... (2/2 runs complete)
testing upgrade vectors... (7/7 runs complete)
  Starting phi = 0.749557;  ending phi = 0.438044  (58.4404% starting phi)

OPTIMISATION ITERATION NUMBER: 5
  Iteration type: base parameter solution
calculatingjacobian... (2/2 runs complete)
testing upgrade vectors... (7/7 runs complete)
Starting phi = 0.438044;  ending phi = 0.437872  (99.9609% starting phi)

OPTIMISATION ITERATION NUMBER: 6
  Iteration type: base parameter solution
calculatingjacobian... (2/2 runs complete)
testing upgrade vectors... (7/7 runs complete)
Starting phi = 0.437872;  ending phi = 0.437865  (99.9982% starting phi)


Simulation Complete - Press RETURN to close window
```

PEST++ storage5.rec Output File

```
PEST++ Version 1.0.0

Control file = storage5”

OPTIMISATION ITERATION NUMBER: 1
    Iteration type: base parameter solution
    Model calls so far : 0
    Starting phi for this iteration                    Total : 594.589
    Contribution to phi from observation group     “OBSGROUP” : 594.589

      SVD information:
number of singular values used: 2/2
upgrade vector magnitude (without limits or bounds) = 0.727904
angle to direction of greatest descent: 45.2942 deg

      Rotation Factor = 0.00 (0.00 deg);  phi = 213.625 (35.93% starting phi)
      Rotation Factor = 0.01 (0.41 deg);  phi = 208.455 (35.06% starting phi)
      Rotation Factor = 0.10 (4.19 deg);  phi = 162.589 (27.34% starting phi)
      Rotation Factor = 0.20 (8.59 deg);  phi = 114.48 (19.25% starting phi)
      Rotation Factor = 0.50 (22.65 deg);  phi = 22.7623 (3.83% starting phi)
      Rotation Factor = 0.70 (32.12 deg);  phi = 124.385 (20.92% starting phi)
      Rotation Factor = 1.00 (45.29 deg);  phi = 522.574 (87.89% starting phi)

    Parameter Upgrades (Control File Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND            0.00184708         0.005       2.70697      0.630584
    SCOEFF                0.15          0.05             3            -2
    RECHARGE               0.1           0.1             1             0

    Parameter Upgrades (Transformed Numeric Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND              -2.73351      -2.30103       1.18795     -0.187952
    SCOEFF           -0.823909      -1.30103       1.57909      0.366726

   Maximum changes in transformed numeric parameters:
     Maximum relative change = 0.366726   [SCOEFF]
     Maximum factor change = 1.57909   [SCOEFF]


OPTIMISATION ITERATION NUMBER: 2
    Iteration type: base parameter solution
    Model calls so far : 10
    Starting phi for this iteration                    Total : 22.7623
    Contribution to phi from observation group     “OBSGROUP” : 22.7623

      SVD information:
number of singular values used: 2/2
upgrade vector magnitude (without limits or bounds) = 0.233686
angle to direction of greatest descent: 24.3818 deg

      Rotation Factor = 0.00 (0.00 deg);  phi = 14.9341 (65.61% starting phi)
      Rotation Factor = 0.01 (0.24 deg);  phi = 15.4713 (67.97% starting phi)
      Rotation Factor = 0.10 (2.39 deg);  phi = 20.7701 (91.25% starting phi)
      Rotation Factor = 0.20 (4.81 deg);  phi = 27.5767 (121.15% starting phi)
      Rotation Factor = 0.50 (12.19 deg);  phi = 52.672 (231.40% starting phi)
      Rotation Factor = 0.70 (17.13 deg);  phi = 71.9359 (316.03% starting phi)
      Rotation Factor = 1.00 (24.38 deg);  phi = 101.828 (447.35% starting phi)
```

```
    Parameter Upgrades (Control File Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND            0.00118955    0.00184708       1.55276      0.355985
    SCOEFF            0.204449          0.15       1.36299     -0.362992
    RECHARGE               0.1           0.1             1             0

    Parameter Upgrades (Transformed Numeric Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND              -2.92462      -2.73351       1.06991    -0.0699115
    SCOEFF           -0.689416     -0.823909       1.19508      0.163238

   Maximum changes in transformed numeric parameters:
     Maximum relative change = 0.163238   [SCOEFF]
     Maximum factor change = 1.19508   [SCOEFF]


OPTIMISATION ITERATION NUMBER: 3
    Iteration type: base parameter solution
    Model calls so far : 19
    Starting phi for this iteration                    Total : 14.9341
    Contribution to phi from observation group     "OBSGROUP" : 14.9341

      SVD information:
number of singular values used: 2/2
upgrade vector magnitude (without limits or bounds) = 0.176581
angle to direction of greatest descent: 73.4767 deg

      Rotation Factor = 0.00 (0.00 deg);  phi = 0.937774 (6.28% starting phi)
      Rotation Factor = 0.01 (0.55 deg);  phi = 0.749557 (5.02% starting phi)
      Rotation Factor = 0.10 (5.90 deg);  phi = 1.5011 (10.05% starting phi)
      Rotation Factor = 0.20 (12.61 deg);  phi = 9.06701 (60.71% starting phi)
      Rotation Factor = 0.50 (36.74 deg);  phi = 84.7025 (567.17% starting phi)
      Rotation Factor = 0.70 (53.36 deg);  phi = 147.327 (986.51% starting phi)
      Rotation Factor = 1.00 (73.48 deg);  phi = 179.343 (1200.90% starting phi)

    Parameter Upgrades (Control File Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND           0.000792229    0.00118955       1.50152      0.334008
    SCOEFF             0.20644      0.204449       1.00974   -0.00973847
    RECHARGE               0.1           0.1             1             0

    Parameter Upgrades (Transformed Numeric Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND              -3.10115      -2.92462       1.06036    -0.0603604
    SCOEFF           -0.685207     -0.689416       1.00614    0.00610503

   Maximum changes in transformed numeric parameters:
     Maximum relative change = -0.0603604   [COND]
     Maximum factor change = 1.06036   [COND]


OPTIMISATION ITERATION NUMBER: 4
    Iteration type: base parameter solution
    Model calls so far : 28
    Starting phi for this iteration                    Total : 0.749557
    Contribution to phi from observation group     "OBSGROUP" : 0.749557

      SVD information:
```

```
number of singular values used: 2/2
upgrade vector magnitude (without limits or bounds) = 0.0343731
angle to direction of greatest descent: 79.2524 deg

      Rotation Factor = 0.00 (0.00 deg);  phi = 0.439216 (58.60% starting phi)
      Rotation Factor = 0.01 (0.57 deg);  phi = 0.438044 (58.44% starting phi)
      Rotation Factor = 0.10 (6.10 deg);  phi = 0.533744 (71.21% starting phi)
      Rotation Factor = 0.20 (13.21 deg);  phi = 0.927359 (123.72% starting phi)
      Rotation Factor = 0.50 (39.63 deg);  phi = 4.2616 (568.55% starting phi)
      Rotation Factor = 0.70 (57.95 deg);  phi = 6.85316 (914.29% starting phi)
      Rotation Factor = 1.00 (79.25 deg);  phi = 8.26179 (1102.22% starting phi)

    Parameter Upgrades (Control File Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND           0.000731946   0.000792229       1.08236     0.0760929
    SCOEFF            0.206576       0.20644       1.00066  -0.000660826
    RECHARGE               0.1           0.1             1             0

    Parameter Upgrades (Transformed Numeric Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND              -3.13552      -3.10115       1.01108    -0.0110835
    SCOEFF            -0.68492     -0.685207       1.00042   0.000418703

   Maximum changes in transformed numeric parameters:
     Maximum relative change = -0.0110835   [COND]
     Maximum factor change = 1.01108   [COND]


OPTIMISATION ITERATION NUMBER: 5
    Iteration type: base parameter solution
    Model calls so far : 37
    Starting phi for this iteration                    Total : 0.438044
    Contribution to phi from observation group     “OBSGROUP” : 0.438044

      SVD information:
number of singular values used: 2/2
upgrade vector magnitude (without limits or bounds) = 0.00227503
angle to direction of greatest descent: 71.7733 deg

      Rotation Factor = 0.00 (0.00 deg);  phi = 0.437873 (99.96% starting phi)
      Rotation Factor = 0.01 (0.55 deg);  phi = 0.437872 (99.96% starting phi)
      Rotation Factor = 0.10 (5.82 deg);  phi = 0.438302 (100.06% starting phi)
      Rotation Factor = 0.20 (12.42 deg);  phi = 0.439937 (100.43% starting phi)
      Rotation Factor = 0.50 (35.89 deg);  phi = 0.453779 (103.59% starting phi)
      Rotation Factor = 0.70 (52.03 deg);  phi = 0.466782 (106.56% starting phi)
      Rotation Factor = 1.00 (71.77 deg);  phi = 0.479919 (109.56% starting phi)

      Switching to central derivatives:

    Parameter Upgrades (Control File Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND           0.000728163   0.000731946        1.0052    0.00516841
    SCOEFF            0.206736      0.206576       1.00077  -0.000771491
    RECHARGE               0.1           0.1             1             0

    Parameter Upgrades (Transformed Numeric Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND              -3.13777      -3.13552       1.00072  -0.000717723
```

```
    SCOEFF           -0.684585      -0.68492       1.00049   0.000488999

   Maximum changes in transformed numeric parameters:
     Maximum relative change = -0.000717723   [COND]
     Maximum factor change = 1.00072   [COND]


OPTIMISATION ITERATION NUMBER: 6
    Iteration type: base parameter solution
    Model calls so far : 46
    Starting phi for this iteration                    Total : 0.437872
    Contribution to phi from observation group     “OBSGROUP” : 0.437872

      SVD information:
number of singular values used: 2/2
upgrade vector magnitude (without limits or bounds) = 0.000495033
angle to direction of greatest descent: 79.207 deg

      Rotation Factor = 0.00 (0.00 deg);  phi = 0.437867 (100.00% starting phi)
      Rotation Factor = 0.01 (0.57 deg);  phi = 0.437865 (100.00% starting phi)
      Rotation Factor = 0.10 (6.10 deg);  phi = 0.437889 (100.00% starting phi)
      Rotation Factor = 0.20 (13.20 deg);  phi = 0.437975 (100.02% starting phi)
      Rotation Factor = 0.50 (39.60 deg);  phi = 0.438744 (100.20% starting phi)
      Rotation Factor = 0.70 (57.92 deg);  phi = 0.439424 (100.35% starting phi)
      Rotation Factor = 1.00 (79.21 deg);  phi = 0.439956 (100.48% starting phi)

    Parameter Upgrades (Control File Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND           0.000727346   0.000728163       1.00112      0.001122
    SCOEFF            0.206777      0.206736        1.0002   -0.00019962
    RECHARGE               0.1           0.1             1             0

    Parameter Upgrades (Transformed Numeric Parameters)
     Parameter      Current       Previous       Factor       Relative
        Name         Value         Value         Change        Change
    ------------  ------------  ------------  ------------  ------------
    COND              -3.13826      -3.13777       1.00016  -0.000155382
    SCOEFF           -0.684498     -0.684585       1.00013   0.000126624

   Maximum changes in transformed numeric parameters:
     Maximum relative change = -0.000155382   [COND]
     Maximum factor change = 1.00016   [COND]
```

PEST Model Run

The storage example problem can be run using PEST by typing the command:

pest storage5

The output written to the screen by PEST and the contents of the storage5.rec output file are presented below.

PEST Output Written to the Screen

```
C:\Users\dwelter\Desktop\alpha_0.0.1\stor>pest storage5

PEST Version 12.0.1.Watermark Numerical Computing.

PEST is running in parameter estimation mode.

PEST run record: case storage5
(See file storage5.rec for full details.)

Model command line:
storage1

Running model .....

   Running model 1 time....
   Sum of squared weighted residuals (ie phi) =   594.59

OPTIMISATION ITERATION NO.        : 1
   Model calls so far             : 1
   Starting phi for this iteration:   594.59

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =   5.0000     ----->
running model .....
          Phi =   178.26      (  0.300 of starting phi)

   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:   178.26
```

```
   Maximum   factor change:  1.747     ["cond"]
   Maximum relative change: 0.7351     ["scoeff"]


OPTIMISATION ITERATION NO.        : 2
   Model calls so far             : 4
   Starting phi for this iteration:   178.26

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =   2.5000     ----->
running model .....
          Phi =   99.133      (  0.556 of starting phi)

       Lambda =   1.2500     ----->
running model .....
          Phi =   96.104      (  0.539 of starting phi)

       Lambda =  0.62500     ----->
running model .....
          Phi =   93.707      (  0.526 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   93.707
   Maximum   factor change:  2.241     ["scoeff"]
   Maximum relative change:  1.241     ["scoeff"]


OPTIMISATION ITERATION NO.        : 3
   Model calls so far             : 9
   Starting phi for this iteration:   93.707

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =  0.31250     ----->
running model .....
          Phi =   8.0233      (  0.086 of starting phi)

   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:   8.0233
   Maximum   factor change:  1.517     ["cond"]
   Maximum relative change: 0.3410     ["cond"]


OPTIMISATION ITERATION NO.        : 4
   Model calls so far             : 12
   Starting phi for this iteration:   8.0233

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =  0.15625     ----->
running model .....
          Phi =   4.3394      (  0.541 of starting phi)

       Lambda =  7.81250E-02 ----->
running model .....
          Phi =   4.2460      (  0.529 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   4.2460
   Maximum   factor change:  1.398     ["cond"]
   Maximum relative change: 0.2845     ["cond"]
```

```
OPTIMISATION ITERATION NO.        : 5
   Model calls so far             : 16
   Starting phi for this iteration:   4.2460

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =  3.90625E-02 ----->
running model .....
          Phi =  0.49781      (  0.117 of starting phi)

   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:  0.49781
   Maximum   factor change:  1.243     ["cond"]
   Maximum relative change: 0.1953     ["cond"]


OPTIMISATION ITERATION NO.        : 6
   Model calls so far             : 19
   Starting phi for this iteration:  0.49781

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =  1.95313E-02 ----->
running model .....
          Phi =  0.43883      (  0.882 of starting phi)

       Lambda =  9.76563E-03 ----->
running model .....
          Phi =  0.43913      (  0.882 of starting phi)

       Lambda =  3.90625E-02 ----->
running model .....
          Phi =  0.43856      (  0.881 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  0.43856
   Maximum   factor change:  1.088     ["cond"]
   Maximum relative change: 8.1179E-02 ["cond"]


OPTIMISATION ITERATION NO.        : 7
   Model calls so far             : 24
   Starting phi for this iteration:  0.43856

   Calculating Jacobian matrix: running model 2 times .....
   2 runs completed.

       Lambda =  3.90625E-02 ----->
running model .....
          Phi =  0.43788      (  0.998 of starting phi)

       Lambda =  1.95313E-02 ----->
running model .....
          Phi =  0.43787      (  0.998 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  0.43787
   Relative phi reduction between optimisation iterations less than 0.1000
   Switch to central derivatives calculation
   Maximum   factor change:  1.010     ["cond"]
   Maximum relative change: 9.4250E-03 ["cond"]
```

```
OPTIMISATION ITERATION NO.        : 8
   Model calls so far             : 28
   Starting phi for this iteration:  0.43787

   Calculating Jacobian matrix: running model 4 times .....
   4 runs completed.

       Lambda =  9.76563E-03 ----->
running model .....
          Phi =  0.43786      (  1.000 of starting phi)

       Lambda =  4.88281E-03 ----->
running model .....
          Phi =  0.43787      (  1.000 times starting phi)

       Lambda =  1.95313E-02 ----->
running model .....
          Phi =  0.43786      (  1.000 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  0.43786
   Maximum   factor change:  1.001     [“cond”]
   Maximum relative change: 5.7673E-04 [“cond”]

Optimisation complete: the  3 lowest phi’s are within a relative distance
ofeachother of 1.000E-02
   Total model calls:     35

Running model one last time with best parameters.....

Recording run statistics .....

See file storage5.rec for full run details.
See file storage5.sen for parameter sensitivities.
See file storage5.seo for observation sensitivities.
See file storage5.res for residuals.
```

PEST storage5.rec Output File

```
                    PEST RUN RECORD: CASE storage5


PEST run mode:-

   Parameter estimation mode


Case dimensions:-

   Number of parameters                          :     3
   Number of adjustable parameters               :     2
   Number of parameter groups                    :     3
   Number of observations                        :    16
   Number of prior estimates                     :     0


Model command line(s):-

   storage1


Jacobian command line:-

na


Model interface files:-
   Templates:
      input.tpl
for model input files:
      input.dat

   (Parameter values written using single precision protocol.)
   (Decimal point always included.)

   Instruction files:
      output.ins
for reading model output files:
      output.dat


PEST-to-model message file:-

na


Derivatives calculation:-

Param        Increment   IncrementIncrement   Forward or   Multiplier  Method
group        type                  low bound   central      (central)   (central)
recharge     relative    1.0000E-02   none         switch       2.000     parabolic
cond         relative    1.0000E-02   none         switch       2.000     parabolic
scoeff       relative    1.0000E-02   none         switch       2.000     parabolic


Parameter definitions:-

Name         Trans-            Change       Initial        Lower          Upper
formation         limit        value          bound          bound
recharge     fixed              na       0.100000          nana
cond         log               factor     5.000000E-03   1.000000E-10   1.000000E+10
scoeff       log               factor     5.000000E-02   1.000000E-10   1.000000E+10

Name         Group          Scale         Offset        Model command number
```

```
rechargerecharge      1.00000        0.00000            1
condcond          1.00000        0.00000            1
scoeffscoeff        1.00000        0.00000            1


Prior information:-

   No prior information supplied


Observations:-

Observation name        Observation       Weight       Group
head1                  4.998750E-02      1.000       obsgroup
head2                  9.995002E-02      1.000       obsgroup
 head3                  0.249688          1.000       obsgroup
 head4                  0.498752          1.000       obsgroup
 head5                  0.955017          1.000       obsgroup
 head6                   2.66901          1.000       obsgroup
 head7                   4.67706          1.000       obsgroup
 head8                   9.81626          1.000       obsgroup
 head9                   21.8199          1.000       obsgroup
 head10                  40.8469          1.000       obsgroup
 head11                  63.2121          0.000       obsgroup
 head12                  91.7915          0.000       obsgroup
 head13                  99.3262          0.000       obsgroup
 head14                  99.9955          0.000       obsgroup
 head15                  100.000          0.000       obsgroup
 head16                  100.000          0.000       obsgroup


Control settings:-

   Initial lambda                                          :  5.0000
   Lambda adjustment factor                                :  2.0000
   Sufficient new/old phi ratio per optimisation iteration      : 0.30000
   Limiting relative phi reduction between lambdas          : 3.00000E-02
   Maximum trial lambdas per iteration                      :  10
   Forgive model run failure during lamda testing           : no

   Perform Broyden's update of Jacobian matrix              : no

Maximum  factor  parameter change (factor-limited changes)   :  3.0000
   Maximum relative parameter change (relative-limited changes) :na
   Fraction of initial parameter values used in computing
change limit for near-zero parameters                     : 1.00000E-03
   Allow bending of parameter upgrade vector                : no
   Allow parameters to stick to their bounds                : no

   Relative phi reduction below which to begin use of
central derivatives                                       : 0.10000
   Iteration at which to first consider derivatives switch      :   1

   Relative phi reduction indicating convergence            : 0.10000E-01
   Number of phi values required within this range          :   3
   Maximum number of consecutive failures to lower phi      :   3
   Minimal relative parameter change indicating convergence     : 0.10000E-01
   Number of consecutive iterations with minimal param change   :   3
   Maximum number of optimisation iterations                :  30

   Attempt automatic user intervention                      : no

   Attempt reuse of parameter sensitivities                 : no


File saving options: -
```

```
   Save multiple JCO files                                  : no
   Save multiple REI files                                  : no


                           OPTIMISATION RECORD


INITIAL CONDITIONS:
   Sum of squared weighted residuals (ie phi) =   594.59

      Current parameter values
recharge        0.100000
cond            5.000000E-03
scoeff          5.000000E-02


OPTIMISATION ITERATION NO.        : 1
   Model calls so far             : 1
   Starting phi for this iteration:   594.59

       Lambda =   5.0000     ----->
          Phi =   178.26      (  0.300 of starting phi)

   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:   178.26

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            2.861794E-03            cond             5.000000E-03
scoeff          8.675662E-02            scoeff           5.000000E-02
   Maximum   factor change:  1.747     ["cond"]
   Maximum relative change: 0.7351     ["scoeff"]


OPTIMISATION ITERATION NO.        : 2
   Model calls so far             : 4
   Starting phi for this iteration:   178.26

       Lambda =   2.5000     ----->
          Phi =   99.133      (  0.556 of starting phi)

       Lambda =   1.2500     ----->
          Phi =   96.104      (  0.539 of starting phi)

       Lambda =  0.62500     ----->
          Phi =   93.707      (  0.526 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   93.707

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            2.108721E-03            cond             2.861794E-03
scoeff          0.194440                scoeff           8.675662E-02
   Maximum   factor change:  2.241     ["scoeff"]
   Maximum relative change:  1.241     ["scoeff"]


OPTIMISATION ITERATION NO.        : 3
   Model calls so far             : 9
   Starting phi for this iteration:   93.707

       Lambda =  0.31250     ----->
          Phi =   8.0233      (  0.086 of starting phi)
```

```
   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:   8.0233

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            1.389632E-03            cond             2.108721E-03
scoeff          0.169866                scoeff           0.194440
   Maximum   factor change:  1.517     ["cond"]
   Maximum relative change: 0.3410     ["cond"]


OPTIMISATION ITERATION NO.        : 4
   Model calls so far             : 12
   Starting phi for this iteration:   8.0233

       Lambda =  0.15625     ----->
          Phi =   4.3394      (  0.541 of starting phi)

       Lambda =  7.81250E-02 ----->
          Phi =   4.2460      (  0.529 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:   4.2460

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            9.943230E-04            cond             1.389632E-03
scoeff          0.203611                scoeff           0.169866
   Maximum   factor change:  1.398     ["cond"]
   Maximum relative change: 0.2845     ["cond"]


OPTIMISATION ITERATION NO.        : 5
   Model calls so far             : 16
   Starting phi for this iteration:   4.2460

       Lambda =  3.90625E-02 ----->
          Phi =  0.49781      (  0.117 of starting phi)

   No more lambdas: phi is less than 0.3000 of starting phi
   Lowest phi this iteration:  0.49781

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            8.001240E-04            cond             9.943230E-04
scoeff          0.203811                scoeff           0.203611
   Maximum   factor change:  1.243     ["cond"]
   Maximum relative change: 0.1953     ["cond"]


OPTIMISATION ITERATION NO.        : 6
   Model calls so far             : 19
   Starting phi for this iteration:  0.49781

       Lambda =  1.95313E-02 ----->
          Phi =  0.43883      (  0.882 of starting phi)

       Lambda =  9.76563E-03 ----->
          Phi =  0.43913      (  0.882 of starting phi)

       Lambda =  3.90625E-02 ----->
          Phi =  0.43856      (  0.881 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  0.43856
```

```
      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            7.351710E-04            cond             8.001240E-04
scoeff          0.206478                scoeff           0.203811
   Maximum   factor change:  1.088     ["cond"]
   Maximum relative change: 8.1179E-02 ["cond"]


OPTIMISATION ITERATION NO.        : 7
   Model calls so far             : 24
   Starting phi for this iteration:  0.43856

       Lambda =  3.90625E-02 ----->
          Phi =  0.43788      (  0.998 of starting phi)

       Lambda =  1.95313E-02 ----->
          Phi =  0.43787      (  0.998 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  0.43787
   Relative phi reduction between optimisation iterations less than 0.1000
   Switch to central derivatives calculation

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            7.282420E-04            cond             7.351710E-04
scoeff          0.206731                scoeff           0.206478
   Maximum   factor change:  1.010     ["cond"]
   Maximum relative change: 9.4250E-03 ["cond"]


OPTIMISATION ITERATION NO.        : 8
   Model calls so far             : 28
   Starting phi for this iteration:  0.43787

       Lambda =  9.76563E-03 ----->
          Phi =  0.43786      (  1.000 of starting phi)

       Lambda =  4.88281E-03 ----->
          Phi =  0.43787      (  1.000 times starting phi)

       Lambda =  1.95313E-02 ----->
          Phi =  0.43786      (  1.000 of starting phi)

   No more lambdas: relative phi reduction between lambdas less than 0.0300
   Lowest phi this iteration:  0.43786

      Current parameter values                Previous parameter values
recharge        0.100000                recharge         0.100000
cond            7.278220E-04            cond             7.282420E-04
scoeff          0.206756                scoeff           0.206731
   Maximum   factor change:  1.001     ["cond"]
   Maximum relative change: 5.7673E-04 ["cond"]

Optimisation complete: the  3 lowest phi's are within a relative distance
ofeachother of 1.000E-02
   Total model calls:     35

   The model has been run one final time using best parameters.
   Thus all model input files contain best parameter values, and model
output files contain model results based on these parameters.


                           OPTIMISATION RESULTS
```

```
Adjustable parameters ----->

Parameter        Estimated         95% percent confidence limits
value             lower limit       upper limit
cond           7.278220E-04       5.770578E-04      9.179754E-04
scoeff         0.206756           0.198685          0.215154

Note: confidence limits provide only an indication of parameter uncertainty.
      They rely on a linearity assumption which  may not extend as far in
parameter space as the confidence limits themselves - see PEST manual.


Fixed parameters ----->

Parameter      Fixed value
recharge       0.100000

See file storage5.sen for parameter sensitivities.


Observations ----->

Observation            Measured       Calculated     Residual       Weight     Group
valuevalue
head1                 4.998750E-02   4.835774E-02   1.629764E-03    1.000      obsgroup
head2                 9.995002E-02   9.669846E-02   3.251564E-03    1.000      obsgroup
head3                 0.249688       0.241619       8.069500E-03    1.000      obsgroup
head4                 0.498752       0.482812       1.593980E-02    1.000      obsgroup
head5                 0.955017       0.963928      -8.910700E-03    1.000      obsgroup
head6                  2.66901        2.39715       0.271856        1.000      obsgroup
head7                  4.67706        4.75249      -7.542500E-02    1.000      obsgroup
head8                  9.81626        9.34058       0.475676        1.000      obsgroup
head9                  21.8199        22.1744      -0.354540        1.000      obsgroup
head10                 40.8469        40.7702       7.675000E-02    1.000      obsgroup
head11                 63.2121        69.4424       -6.23030        0.000      obsgroup
head12                 91.7915        113.760       -21.9688        0.000      obsgroup
head13                 99.3262        133.330       -34.0040        0.000      obsgroup
head14                 99.9955        137.276       -37.2804        0.000      obsgroup
head15                 100.000        137.396       -37.3962        0.000      obsgroup
head16                 100.000        137.396       -37.3962        0.000      obsgroup

See file storage5.res for more details of residuals in graph-ready format.

See file storage5.seo for composite observation sensitivities.


Objective function ----->

  Sum of squared weighted residuals (ie phi)               =  0.4379
```

```
Correlation Coefficient ----->

  Correlation coefficient                                =  0.9999


Analysis of residuals ----->

  All residuals:-
     Number of residuals with non-zero weight                      =    10
     Mean value of non-zero weighted residuals                     =  4.1430E-02
     Maximum weighted residual [observation "head8"]               =  0.4757
     Minimum weighted residual [observation "head9"]               = -0.3545
     Standard variance of weighted residuals                       =  5.4733E-02
     Standard error of weighted residuals                          =  0.2340

     Note: the above variance was obtained by dividing the objective
function by the number of system degrees of freedom (ie. number of
observations with non-zero weight plus number of prior information
articles with non-zero weight minus the number of adjustable parameters.)
     If the degrees of freedom is negative the divisor becomes
the number of observations with non-zero weight plus the number of
prior information items with non-zero weight.


K-L information statistics ----->


  AIC   =  -25.28434
AICC  =  -21.28434
  BIC   =  -24.37658
  KIC   =  -16.25440

Parameter covariance matrix ----->

condscoeff
cond           1.9110E-03  -3.0938E-04
scoeff        -3.0938E-04   5.6236E-05


Parameter correlation coefficient matrix ----->

condscoeff
cond            1.000      -0.9437
scoeff        -0.9437        1.000


Normalized eigenvectors of parameter covariance matrix ----->

              Vector_1     Vector_2
cond           0.1603      -0.9871
scoeff         0.9871       0.1603


Eigenvalues ----->

               5.9913E-06   1.9612E-03
```

Publishing support provided by
Columbus and Lafayette Publishing Service Centers