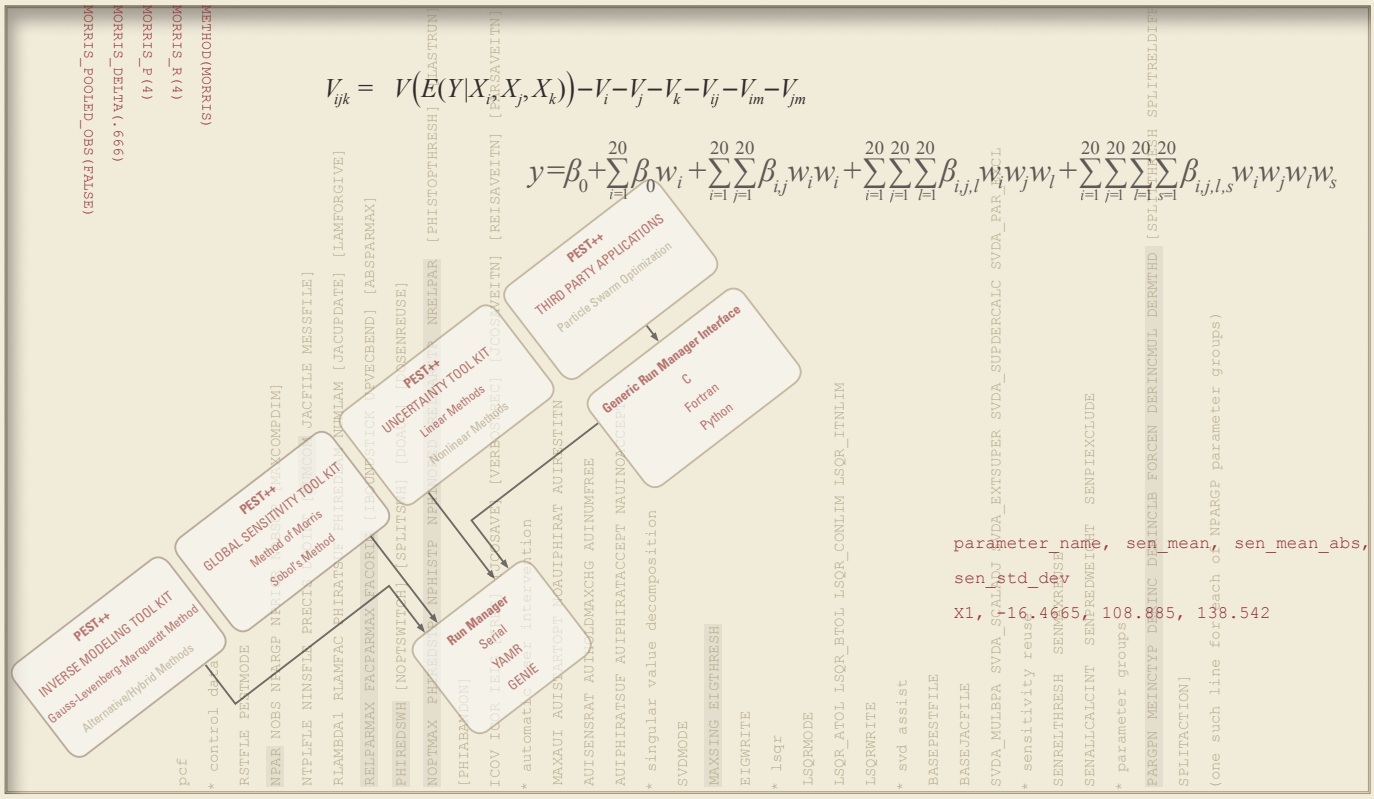
USGS
science for a changing world

Groundwater Resources Program
Prepared in cooperation with U.S. Environmental Protection Agency,
Great Lakes Restoration Initiative

Approaches in Highly Parameterized Inversion: PEST++ Version 3, A Parameter ESTimation and Uncertainty Analysis Software Suite Optimized for Large Environmental Models



Techniques and Methods 7–C12

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

Approaches in Highly Parameterized Inversion: PEST++ Version 3, A Parameter ESTimation and Uncertainty Analysis Software Suite Optimized for Large Environmental Models

By David E. Welter, Jeremy T. White, Randall J. Hunt, and John E. Doherty

Groundwater Resources Program
Prepared in cooperation with U.S. Environmental Protection Agency,
Great Lakes Restoration Initiative

Techniques and Methods 7–C12

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

U.S. Department of the Interior
SALLY JEWELL, Secretary

U.S. Geological Survey
Suzette M. Kimball, Acting Director

U.S. Geological Survey, Reston, Virginia: 2015

For more information on the USGS—the Federal source for science about the Earth, its natural and living resources, natural hazards, and the environment—visit http://www.usgs.gov or call 1–888–ASK–USGS.

For an overview of USGS information products, including maps, imagery, and publications, visit http://www.usgs.gov/pubprod/.

Suggested citation:
Welter, D.E., White, J.T., Hunt, R.J., and Doherty, J.E. , 2015, Approaches in highly parameterized inversion—PEST++ Version 3, a Parameter ESTimation and uncertainty analysis software suite optimized for large environmental models: U.S. Geological Survey Techniques and Methods, book 7, chap. C12, 54 p., http://dx.doi.org/10.3133/tm7C12.

ISSN 2328-7055 (online)

Contents

Figures

Figures—Continued

Tables

Approaches in Highly Parameterized Inversion: PEST++ Version 3, a Parameter ESTimation and Uncertainty Analysis Software Suite Optimized for Large Environmental Models

By David E. Welter,[1] Jeremy T. White,[2] Randall J. Hunt,[2] and John E. Doherty[3]

Abstract

The PEST++ Version 1 object-oriented parameter estimation code is here extended to Version 3 to incorporate additional algorithms and tools to further improve support for large and complex environmental modeling problems. PEST++ Version 3 includes the Gauss-Marquardt-Levenberg (GML) algorithm for nonlinear parameter estimation, Tikhonov regularization, integrated linear-based uncertainty quantification, options of integrated TCP/IP based parallel run management or external independent run management by use of a Version 2 update of the GENIE Version 1 software code, and utilities for global sensitivity analyses. The Version 3 code design is consistent with PEST++ Version 1 and continues to be designed to lower the barriers of entry for users as well as developers while providing efficient and optimized algorithms capable of accommodating large, highly parameterized inverse problems. As such, this effort continues the original focus of (1) implementing the most popular and powerful features of the PEST software suite in a fashion that is easy for novice or experienced modelers to use and (2) developing a software framework that is easy to extend.

The PEST++ Version 3 software suite can be compiled for Microsoft Windows®[4] and Linux®[5] operating systems; the source code is available in a Microsoft Visual Studio®[6] 2013 solution; Linux Makefiles are also provided. PEST++ Version 3 continues to build a foundation for an open-source framework capable of producing robust and efficient parameter estimation tools for large environmental models.

[1]Computational Water Resource Engineering.

[2]U.S. Geological Survey.

[3]Watermark Numerical Computing.

[4]"Windows" is a registered trademark of Microsoft Corporation in the United States and other countries

[5]"Linux" is the registered trademark of Linus Torvalds in the United States and other countries.

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

Introduction

Calibration of environmental models is an inherently non-unique (underdetermined) inverse problem, where infinitely many parameter sets can be found to provide suitable history matching of observations. Such inverse problems are considered to be ill posed because of the very large parameter space associated with a complex reality and information deficits in the accompanying observation dataset. Hence, quantifying the reliability of forecasts made with environmental models is becoming an important component of societal decision making. Oreskes and others (1994), Saltelli and others (2004), Pilkey and Pilkey-Jarvis (2007), Beven (2009), Doherty (2011), and White and others (2014) discuss some underlying modeling and uncertainty issues in detail and put forth concepts about the appropriate roles and uses of models in the process of environmental planning and decision making. As a result of these and many other works, parameter estimation and uncertainty analyses together are now considered a standard component of defensible environmental modeling (Anderson and others, 2015).

PEST++ Version 1 was built upon the theory documented in the PEST software suite (Doherty, 2010a,b), a widely used parameter estimation code in the environmental modeling community, with specific emphasis on handling highly parameterized inverse problems (Doherty and Hunt, 2010). The term "highly parameterized" refers to the use of many parameters in the inverse problem so that the optimal amount of information can be extracted from the calibration dataset. Highly parameterized problems are not only inherently non-unique but also characterized by calibration/uncertainty analyses that are computationally expensive. The use of regularized inversion techniques (where expert knowledge is explicitly employed to govern parameter plausibility and to stabilize the inverse problem; Hunt and others, 2007) on computationally expensive, highly parameterized models has been made more tractable by the proliferation of relatively inexpensive multicore processors available in modern desktop computers and the advent of cloud computing (Hunt and others, 2010).

When undertaking regularized inversion, modelers should not artificially limit the number of adjustable parameters in the analysis that could potentially influence the model forecast; in other words, "A regularized-inversion philosophy to parameterization, then, can be summarized as 'if in doubt, include it'" (Doherty and Hunt, 2010). This approach helps improve the reliability of the forecasts by minimizing biases introduced by the parameterization (Moore and Doherty, 2005; White and others, 2014). As noted by Welter and others (2012), the corresponding increase in modeling problem size has pushed many existing groundwater software tools to their limits, especially as advancements to the science increase both the number and sophistication of the software tools.

PEST++ Version 1 augmented available software by making tools accessible and more intuitive for new users but also more robust and efficient for those working on increasingly complex and more highly parameterized problems. However, Version 1 was a first step and did not include some capabilities important for environmental modeling. Notably, it did not include an implementation of the Gauss-Marquardt-Levenberg (GML) algorithm, lacked an integrated TCP/IP parallel run manager similar the one developed for PEST by Schreüder (2009) (BeoPEST), and lacked the ability to automatically scale observation weights during Tikhonov regularization (Tikhonov and Arsenin, 1977). Such capabilities are often required for large environmental modeling projects, such as those used in the Great Lakes Restoration Initiative, whereby Great Lake watershed-scale models are subjected to sophisticated, regularized inversion and uncertainty analyses.

Purpose and Scope

This report describes an expansion of the algorithms and tools included in PEST++ Version 1 documented by Welter and others (2012). PEST++ Version 3 supplants Version 1,[7] but the overriding design concepts and approaches are consistent with those previously documented in Welter and others (2012). Therefore, this report focuses on documentation of new capabilities added to PEST++ Version 3. The input instructions and examples are sufficiently documented, however, so a user can run PEST++ Version 3 without referring to the Version 1 documentation. Source code and executable of PEST++ Version 3 as documented in this report are available for download at http://wi.water.usgs.gov/models/pestplusplus/. More recent releases of PEST++, including any enhancements made after publication of this report, will be available at http://www.inversemodeler.org/. The most current development version of the source code is maintained in an online open-source version-control repository at https://github.com/dwelter/pestpp.

[7]Version 2 of PEST++ existed only as a code-development milestone, hence the designation of Version 3 for the current release to the user community.

PEST++ Version 1 did not attempt to reproduce all the functionality of PEST but instead focused on implementing the most used features for highly parameterized inversion while also focusing on streamlining proper use of these features (Welter and others, 2012). However, it was recognized at the time that the list of ported PEST capability would expand as PEST++ was applied to more real-world problems in the future (Welter and others, 2012, p. 7). Thus, this report focuses on the enhancements added to PEST++ Version 3 to incorporate additional features from the PEST software suite of Doherty (2010a,b) and elsewhere. In particular, PEST++ Version 3 includes the following additional capabilities: (1) efficient calculation of parameter upgrades by using a formulation of the Gauss-Marquardt-Levenberg (GML) algorithm, (2) integrated internal TCP/IP parallel run management and external independent TCP/IP parallel run management (GENIE Version 1 of Muffels and others 2012), (3) enhanced Tikhonov regularization capabilities, (4) integrated algorithms for performing linear parameter and predictive uncertainty analyses, and (5) methods for global sensitivity analysis.

The main text of this report briefly introduces the theory supporting the new capabilities of PEST++ Version 3 and provides references to supporting documents. The appendixes document the implementation, input instructions, and examples needed to apply these new capabilities. All related terminology, concepts, file extensions, and so forth, follow the conventions and derivations presented and cited by Doherty (2010a, 2015), Doherty and Hunt (2010), Doherty and others (2010), and Welter and others (2012) and are omitted here for brevity.

Major Enhancements to PEST++ Version 3

As described by Welter and others (2012), PEST++ Version 1 provided enhancements and changes to the original PEST software suite of Doherty (2010a,b). These included capabilities such as the following:

1. The ability to automatically switch between native parameters and superparameters (Tonkin and Doherty, 2005) without user intervention.

2. A PROPACK-based truncated singular value decomposition algorithm for large and sparse matrices (Larsen, 1998, 2001) to increase computational efficiency for high-dimensional inverse problems.

3. Automated normalization of parameter sensitivity based on parameter ranges.

Although these enhancements are important, Version 1 implemented only a subset of the most used PEST features. In PEST++ Version 3, the Version 1 capabilities have been retained and several new capabilities have been added and streamlined from existing PEST methods. These enhancements follow the Version 1 design goal of being object oriented and suitable for future development. In the next sections, the enhancements and changes now available in PEST++ Version 3 are described in detail. Consistent with Version 1 documentation, the input instructions for existing and new capabilities are included in appendix 1.

Gauss-Marquardt-Levenberg (GML) Capabilities

PEST provides the Gauss-Marquardt-Levenberg method with truncated SVD (Aster and Thurber, 2013) as numerical solution techniques for the least-squares problem, but PEST++ Version 1 implemented only truncated SVD. In PEST++ Version 3, a robust implementation of the Gauss-Marquardt-Levenberg (GML) algorithm has been added to the existing truncated SVD solution scheme. Briefly, the Marquardt lambda is a component of the Gauss-Marquardt-Levenberg method. When the Marquardt lambda is zero, the upgrade vector is in the direction of the Gauss-Newton solution, a solution direct that includes "curvature" information from the quadratically approximated Hessian matrix of sensitivities (Oliver and others, 2008). Increasing the magnitude of lambda has the effect of rotating the upgrade vector in the direction of the gradient descent solution, which can be more robust if the solution surface is not well approximated as a quadratic (Oliver and others, 2008). 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 form of regularization. The Marquardt lambda's 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 because it forms a trust region between the two solution endpoints. As a result, testing different values of the Marquardt lambda allows PEST++ to explore a larger portion of parameter space, helping to prevent premature termination at a local minimum on the objective function surface. The reader is referred to Hill and Tiedeman (2007), and Oliver and others (2008), Doherty (2010a, 2015), and Aster and Thurber (2013) for additional description of how GML and truncated SVD approaches are implemented in least-squares nonlinear parameter estimation framework.

It should be noted that in PEST++ Version 1, an SVD rotation factor approach was used instead of a GML approach (Welter and others, 2012) to define a trust region for solution exploration. Subsequent testing by the authors and Dahlstrom and Carter (2013) did not identify cases where this approach was demonstrably superior to the more widely used GML approach. Therefore, to streamline future maintenance of the PEST++ Version 3 code base, the SVD rotation factor capabilities have been omitted in Version 3.

TCP/IP Run Manager Integration (YAMR - Yet Another run ManageR)

Parallel processing of the large numbers of runs required by parameter estimation is a challenge to highly parameterized problems (for example, Hunt and others, 2010). PEST++ Version 1 relied on a separate computer code, GENIE (Muffels and others, 2012) to perform parallel run management across a network by using the TCP/IP protocol. PEST++ Version 3 retains and updates the external independent GENIE run manager to GENIE Version 2 (appendix 2). PEST++ Version 3 also now includes an integrated TCP/IP parallel run manager within the PEST++ Version 3 executable. The advantage of an integrated run manager is that it allows a user to perform parallel run management without requiring running a separate stand-alone code. The new, integrated parallel run manager is called YAMR (Yet Another run ManageR) to distinguish between the new capability and the external GENIE compatibility. YAMR is invoked similarly to BeoPEST (Schrüeder, 2009); instructions for use are provided in appendix 1. Within the YAMR run manager framework, worker nodes are multi-thread as compared to the BeoPEST single-thread worker nodes. The multi-threaded capability allows multiple lines of communication between the master and workers, which in turn facilitates more sophisticated and robust run management than with single-thread nodes. Using multi-threading, the master instance can communicate periodically check the workers' status while a forward run is being executed and, if nessecary, allows the master to interrupt and supplant runs on the workers. This allows the master instance to more easily and efficiently detect run failures and start competitions for overdue runs. PEST++ Version 3 can easily support any number of run managers because it implements an abstract run manager interface, which is a software approach that encapsulates the actual operations of a run manager away from the rest of the code. Additionally, Fortran and Python interfaces to YAMR are included in the PEST++ Version 3 software so that users can easily develop interfaces to YAMR for their own codes that require distributed run management, such as Monte Carlo analyses.

Including Expert Knowledge With Tikhonov Regularization

As described in detail in Doherty and Hunt (2010) and Anderson and others (2015), Tikhonov regularization (Tikhonov and Arsenin, 1977) provides a mechanism for formally incorporating soft knowledge about adjustable parameters into the calibration process. This is performed by augmenting the measurement objective function, which is the weighted sum-of-squared residuals, with a second separate regularization objective function that penalizes deviations from the user-specified preferred parameter states, which may include preferred parameter values and (or) relations between parameters (see Doherty, 2003, p. 171–173). The best-fit

model is then identified as the minimum of the combined measurement-regularization objective function. The formal minimization of the regularization objective function is one approach for determining a unique solution to the inverse problem, one that balances model-to-measurement misfit with adherence to the modeler's knowledge of the system. Mathematically, the regularization objective function supplements the calibration dataset through a suite of special pseudo-observations, each pertaining to a preferred condition for one or more adjustable parameters. In this way, Tikhonov regularization forms a fallback value for parameters, or for relations between parameters, when little or no information is contained for them in the observations used for calibration. Where the information content of a calibration dataset is insufficient for unique estimation of certain parameters or combinations of parameters, the fallback value prevails, resulting in a reasonable parameter value. Additionally, for spatially distributed parameterizations, Tikhonov regularization governs the extent to which heterogeneity is expressed in the estimated parameter field. When properly specified (and weighted), Tikhonov constraints enforce coherent departures from preferred parameter conditions, whether those preferred conditions are expressed as specific parameter values or relationships among parameters. Without such constraints, parameter estimates that result in a good fit with the calibration dataset may nonetheless be considered suboptimal because of implausible or unlikely parameter values (Doherty and Hunt, 2010).

PEST++ Version 1 did not fully support the use of Tikhonov regularization; Version 3 has added the recommended settings as described by Doherty and Hunt (2010). Similar to the setup in PEST, Tikhonov regularization in PEST++ Version 3 is controlled by the variable PHIMLIM, which reflects the "target measurement objective function" specified by the modeler. The target measurement objective function is usually set unrealistically low in an initial parameter estimation run to obtain the best fit to observations (that is, with no soft-knowledge penalty considered). After the first run, PHIMLIM is set to some value above this best fit, often 5–10 percent higher. The PHIMLIM variable gives the user a way to control overfitting—the case when a modeler considers model-to-measurement fit too good given the level of noise associated with the calibration dataset. Thus, the PHIMLIM variable limits how well the estimated parameter values reproduce the observation dataset. PEST++ dynamically adjusts the regularization weight with which Tikhonov constraints are applied by solving a constrained optimization problem. Relaxing Tikhonov constraints achieves a tighter fit with respect to observations, whereas strengthening these constraints results in a poorer fit. One important topic is how the regularization is enforced among different parameter groups (the IREGADJ variable; Doherty, 2010a). PEST++ Version 3 internally uses the recommended setting of IREGADJ = 1 as specified by Doherty and Hunt (2010), which adjust regularization weights by parameter group to ensure that each group is seen in the regularization objective function. The implementation of Tikhonov regularization in PEST is covered in depth by Vogel (2002), Doherty (2003), Oliver and others (2008), Doherty and Hunt (2010), Aster and Thurber (2013), and Doherty (2015). A PEST++ Version 3 example problem which uses dynamically weighted regularization is provided in appendix 3.

Linear Uncertainty Analysis Capabilities

Doherty and others (2010) outline the variety of uncertainty analysis tools available in the PEST software suite. They, as well as Hunt (2012) and Anderson and others (2015), suggest that linear methods may be well suited for efficiently estimating uncertainty around model forecasts for many environmental problems. PEST++ Version 3 includes the ability to estimate the posterior (that is, after calibration) uncertainty by calculating the parameter covariance matrix through the use of linear-based conditional uncertainty propagation. Also included within PEST++ Version 3 is the ability to estimate the prior (before calibration) and posterior (after calibration) forecast uncertainty for any observations listed in the PEST control file. Details regarding the implementation of linear uncertainty methods within PEST++ Version 3 are given in appendix 4, with an example problem provided in appendix 5].

The linear uncertainty analyses implemented in PEST++ Version 3 are based on the use of Schur's complement (Golub and Van Loan, 1996) for conditional uncertainty propagation:

$$\overline{\boldsymbol{\Sigma}}_{\theta} = \boldsymbol{\Sigma}_{\theta} - \boldsymbol{\Sigma}_{\theta}\boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{\Sigma}_{\theta}\boldsymbol{J}^{T} + \boldsymbol{\Sigma}_{\varepsilon})^{-1}\boldsymbol{J}\boldsymbol{\Sigma}_{\theta} \quad (1)$$

where $\boldsymbol{\Sigma}_{\theta}$ is the prior parameter covariance matrix, $\boldsymbol{J}$ is the Jacobian matrix of partial first derivatives of observations with respect to parameters, and $\boldsymbol{\Sigma}_{\varepsilon}$ the covariance matrix of measurement noise. The matrix $\overline{\boldsymbol{\Sigma}}_{\theta}$ of equation 1 is the posterior parameter covariance matrix, which can be seen as the prior parameter covariance matrix less the conditioning provided by the observations, which are embodied in the linear mapping provided by the Jacobian matrix. As such, equation 1 assumes a linear relation between adjustable parameters and model-simulated observation equivalents and that parameter and measurement noise uncertainty can be described by a multivariate Gaussian (or log-Gaussian) distribution. See Fienen and others (2010) for a complete derivation of equation 1 and Doherty (2015) for additional information regarding theoretical underpinnings implied in equation 1. See Dausman and others (2010) for an evaluation of the linearity assumption implied by equation 1. Note that the diagonals of the $\boldsymbol{\Sigma}_{\theta}$ and $\overline{\boldsymbol{\Sigma}}_{\theta}$ matrices are the prior and posterior variances of the adjustable parameters.

If one or more forecast sensitivity vectors are available, then the prior and posterior forecast uncertainty can be calculated by premultiplying and postmultiplying the associated parameter covariance matrices:

$$\sigma_s^2 = \boldsymbol{y}^{T}\boldsymbol{\Sigma}_{\theta}\boldsymbol{y}; \overline{\sigma}_s^2 = \boldsymbol{y}^{T}\overline{\boldsymbol{\Sigma}}_{\theta}\boldsymbol{y} \quad (2)$$

where $\boldsymbol{y}$ is the sensitivity vector of forecast s with respect to the adjustable parameters and σ_s^2 and $\bar{\sigma}_s^2$ are the prior and posterior variances of forecast s, respectively. PEST++ Version 3 facilitates application of equation 2 by extracting one or more observations from the Jacobian to serve as forecast sensitivity vectors. This in turn allows the user to obtain the prior and posterior uncertainty of any forecasts that are included as observations in the PEST++ analysis. See appendixes 4 and 5 for more details. Note that within the PEST++ input and appendixes, the terms "prediction" and "forecast" are used interchangeably.

Global Sensitivity Analysis

GSA++ is a stand-alone program distributed with the PEST++ Version 3 suite of tools that perform global sensitivity analysis (GSA). It leverages the PEST++ code base, which includes the parallel run manger, YAMR. GSA++ is fully compatible with PEST/PEST++ file formats, including template and instruction files. This design facilitates efficient application of global sensitivity analyses to projects set up for PEST++ and (or) PEST. GSA++ is notable because it provides capability currently not available in the PEST software suite. The implementation and use details of GSA++ are provided in appendix 6. Appendixes 7 and 8 demonstrate the use of GSA++ for two example problems.

Global sensitivity analysis is a class of statistical analyses that strives to characterize how model parameters affect model outputs over a wide range of acceptable parameter values. Although there are many different GSA methods, all GSA methods strive to be more robust than traditional, derivative-based local sensitivity analysis, which computes the local sensitivities at a single point in parameter space and is not always adequate for analyzing nonlinear problems where the sensitivities can change depending on where they are computed. Some GSA methods provide general information about the variability of the sensitivities and have relatively low computational requirements, whereas others provide detailed information on nonlinear behavior and interactions between parameters at the expense of larger computational requirements. For a complete introduction to GSA theory and methods, see Saltelli and others (2004, 2008).

The program GSA++ currently supports two GSA methods: (1) the Method of Morris (Morris, 1991), with extensions proposed by Campolongo and others (2005) and Sin and Gernaey (2009), and (2) the Method of Sobol (Sobol, 2001). The Method of Morris is in a class of GSA methods referred to as "one-at-a-time" methods (Saltelli and others, 2004) and is computationally more efficient than other GSA approaches. However, the Method of Morris only provides estimates of the first two moments (mean and variance) of the sensitivity distribution for each parameter. Because of the lack of complete description of the parameter nonlinearity and interactions between parameters, the Method of Morris is typically used as a screening-level tool to identify the most important parameters for the observations tested. This screening is typically followed by application of a more comprehensive tool, such as the Method of Sobol, to the most important subset of parameters to further characterize the effects of parameter nonlinearity and interactions. Because the Method of Sobol is based on the decomposition of variance (Saltelli and others, 2004), it can provide detailed information on how parameter nonlinearity and interaction affect the sensitivity, but at a high computational cost.

The Method of Morris, also known as the Elementary Effects Method (Saltelli and others, 2004), is referred to as a "one-at-a-time" method because each parameter is perturbed sequentially to compute sensitivities. The method, originally proposed by Morris (1991), samples the sensitivity of a given parameter at several locations in parameter space and then provides two measures of parameter sensitivity: the mean (μ) and the standard deviation (σ) of the resulting sensitivity distribution. The mean, μ, captures the overall effect of a parameter on the model output of interest; the standard deviation, σ, measures the variable of a parameter's sensitivity across the range of acceptable parameter values, this being an indicator of how nonlinear a given parameter is and (or) how the parameter interacts with other parameters. It is important to note that the Method of Morris cannot distinguish between parameter nonlinearity and parameter interactions because only the standard deviation of parameter sensitivity is available. However, this method is less computationally demanding than other GSA methods and is easily parallelizable.

The Method of Morris is based on the calculation of elementary effects, which are sensitivities of parameters that have been scaled to a closed interval [0,1] to facilitate direct comparison of the results. Morris (1991), defines an elementary effect as

$$d_i(\mathbf{x}) = \frac{[y(x_1, x_2, \ldots, x_{i-1}, x_i + \Delta, x_{i+1}, \ldots, x_k) - y(x)]}{\Delta} \quad (3)$$

where $d_i(\mathbf{x})$ is an elementary effect, k is the number of parameters, and the x_i variables are the components of a k-dimensional vector containing a set of scaled parameter values. Each x_i is drawn from the closed set $\{0, 1/(p-1), 2/(p-1), \ldots, 1\}$ where p is a variable that defines the number of intervals used for each parameter and Δ is the size of the perturbation applied to scaled parameters to calculate the elementary effects. This sampling strategy partitions the specified parameter range for each parameter into nearly equal parts for sensitivity sampling. Morris (1991) recommends choosing an even number for p and setting $\Delta = p/[2(p-1)]$.

Campolongo and others (2005) extended the Method of Morris by adding the mean of the absolute values (μ^*) as an additional metric to provide a more robust estimate of the parameter sensitivity that is not subject to canceling of positive and negative values that may occur if the sensitivity metric (for example, objective function) is not monotonic.

After a parameter is perturbed and the corresponding forward run is completed, the Method of Morris uses equation 3 to compute an elementary effect for the perturbed parameter. The parameters are each perturbed one at a time to define a trajectory through parameter space which provides a single estimate of the sensitivity of the objective function with respect to each of the scaled parameters. The sample size, *r*, is then defined to be the number of trajectories used in the analysis and represents the number of sensitivities that are computed for each parameter. It is important is remember that, for nonlinear problems, the sensitivity of a parameter can depend on the value of the parameter as well as the values of all the other parameters. Thus, the sample size controls how much of the parameter space is sampled, with a large value providing a more comprehensive sample. Table 1 summarizes the important variables in the Method of Morris.

The standard Method of Morris implementation only computes the sensitivity of a single model output with respect each parameter and is not designed to compute sensitivities with respect to multiple outputs. In contrast, the GSA++ implementation of the Method of Morris computes elementary effects of scaled parameters for PEST++ objective function. Unfortunately, this is not compatible with the previously published work of Morris (1990), which is typically based on a single observation rather than the sum of squares of the residuals of multiple observations. This difference in approach becomes especially problematic when the user wants to perform the Method of Morris on a single observation. The PEST++ objective function would provide perform the analysis on observation squared, which is very different than the observation itself. To overcome this limitation, GSA++ provides an option to compute the Method of Morris sensitivities for each observation in the PEST++ control file in addition to the objective function. Because the additional analysis does not require additional model runs and can provide insight even for those users primarily interested in working with the PEST++ objective function, this option is turned off when a very large number of observations is used and the additional computations would become burdensome.

Table 1. Summary of variables in the Method of Morris.

Variable	Input/output	Description
p	input	Number of levels or the number of points each parameter is sampled at.
Δ	input	Size of the sampling step. This must be a multiple of $p/[2(p-1)]$ and represent the size of the interval that will be used to calculate the perturbation sensitivities.
r	input	Sample size. The number of times the sensitivity will be computed for each parameter.
μ	output	Mean sensitivity of the output with respect to each parameter.
μ^*	output	Mean sensitivity of the absolute value of the output with respect to each parameter.
σ	output	Standard deviation of the sensitivity of the output with respect to each parameter.

Although the Method of Morris can provide valuable information, analyzing the sensitivity of only a single model output is a limitation. Because GSA++ can access the individual observations used to construct the composite objective function, it is desirable to look for ways to extend the Method of Morris to analyze the sensitivities of individual observations or subsets of observations. Sin and Gernaey (2009) proposed the Standardized Elementary Effects (SEE) to extend the Method of Morris to account for the elementary effects of multiple model outputs by using the standard deviation of the elementary effects of parameters as well as the standard deviation of the desired model outputs:

$$SEE_{ij} = \frac{\partial y_j}{\partial x_i} \frac{\sigma_{xi}}{\sigma_{yj}} \tag{4}$$

where SEE_{ij} is the standardized elementary effect of parameter x_i on model output y_j, $\frac{\partial y_j}{\partial x_i}$ is the sensitivity of model output y_j with respect to parameter x_i, σ_{xi} is the standard deviation of elementary effects of scaled parameter x_i, and σ_{yj} is the standard deviation of model output y_j. Equation 4 scales the sensitivities with respect to the individual model outputs. However, this approach may be suboptimal for many water-resource problems because the scaling process obscures the differences in sensitivity between observations that are of a similar type. For example, if a model predicts heads and flows, it is desirable to scale the associated sensitivities of all the heads and the flows by factors that preserve variation insensitivities within the group but scale the sensitivities across the groups. To accomplish this, GSA++ provides an option to use pooled standard deviations in lieu of the individual observation sensitivities in equation 4. The pooled standard deviation is an estimate of the model output standard deviation that contains several groups which have different means but share a common standard deviation. The equation used to compute the pooled variance is

$$Sp = \sqrt{\frac{\sum_{i=1}^{n}(n_i-1)s_i^2}{\sum_{i=1}^{n}(n_i-1)}} \tag{5}$$

where s_p is the pooled standard deviation, n is the number of groups, n_i is the number of samples in group i, and s_i^2 is the standard deviation of group i. It is generally recommended that initial GSA++ runs use the pooled standard deviation option.

The Method of Sobol is based on the decomposition of variance so that the variance of a model output over the parameter space is decomposed to separate the effects arising from different parameters and combinations of parameters. Sobol (2001) and Saltelli and others (2004, 2008) provide comprehensive derivations and explanations of the method, which is summarized herein. The total model output variance, $V_T(Y)$, from a model with k parameters can be decomposed as

$$V_T = V(Y) = \sum_i V_i + \sum_i \sum_{j>i} V_{ij} + \ldots V_{12\ldots k} \quad (6)$$

where

$$V_i = V\left(E(Y|X_i)\right) \quad (7)$$

$$V_{ij} = V\left(E(Y|X_i, X_j)\right) - V_i - V_j \quad (8)$$

$$V_{ijk} = V\left(E(Y|X_i, X_j, X_k)\right) - V_i - V_j - V_k - V_{ij} - V_{im} - V_{jm} \quad (9)$$

When the variance is decomposed in the Method of Sobol, V_i is the variance of the model output attributed to varying i'th parameter while holding all the other parameters fixed, and V_{ij} is the variance in the model output attributed to varying the i'th and j'th parameters simultaneously while holding all the other parameters fixed, minus the variances associated with varying the i'th and j'th parameters individually. The sensitivity indices, S, are defined to be the ratios of the decomposed components of the variance with respect to the total variance and form the basis for the Method of Sobol. These and can be written as

$$S_i = \frac{V_i}{V_T} \quad (10)$$

and

$$S_{ij} = \frac{V_{ij}}{V_T} \quad (11)$$

In a similar way, the total effects term, S_{Ti}, is defined as

$$\left(S_{Ti} = \frac{V_T - V_{-i}}{V_T}\right) \quad (12)$$

where V_{-i} is variance associated with allowing all the parameters but the i'th parameter to vary and thus S_{Ti} represents the total effect of the i'th parameter, including parameter interactions of the i'th parameter with all other adjustable parameters in the model. Because variance is a measure of variability, the parameters which contribute most to the variance of the model output will be the parameters the model is most sensitive to. Inspection of equation 6 reveals how computationally demanding the Method of Sobol can be, especially if the number of parameters is large. This is why the Method of Morris is typically employed to screen all adjustable parameters and identify a (small) subset of parameters that will subsequently be used in the Method of Sobol. See appendix 6 for details regarding the implementation of the Method of Morris and the Method of Sobol in GSA++.

Other Enhancements to PEST++ Version 3

In addition to the major enhancements to PEST++ Version 3 described above, numerous minor enhancements also have been made. A few of these are mentioned below.

Restart Capability

PEST++ Version 1 required the user to start all parameter estimation runs from initial values. PEST++ Version 3 has the ability to restart a run that was terminated prematurely, a feature that can significantly reduce computational demands for highly parameterized problems. Restart is invoked by adding /r flag to the end of the PEST++ command line.

Excluding Lambda Search Using Last-Iteration Base Parameter Jacobian

When PEST++ Version 1 recalculated the base parameter sensitivities, these new sensitivities were used for a lambda search/parameter upgrade evaluation prior to formulation of the internal superparameter solution. However, in some cases, it may be advantageous to forego this base-parameter lambda search and proceed to the superparameter solution. In PEST++ Version 3, the user has the option to omit these lambda search runs performed on the basis of the new base parameter Jacobian matrix by invoking N_ITER_BASE = −1 (appendix 1). Under this condition, PEST++ will omit the lambda search using the base parameter Jacobian matrix and instead define a new superparameter Jacobian matrix on the basis of the newly calculated base parameter sensitivities. This operation is then immediately followed by lambda-based parameter upgrade calculation and testing performed by using only the superparameter Jacobian. This option is equivalent to sequential manual SVDA runs in PEST (Doherty, 2010a) that might be generated by using multiple runs of the SVDAPREP utility at various stages within the PEST-based parameter estimation process (Doherty and Hunt, 2010, p. 22).

Maximum Number of Allowed Forward-Run Failures

When a YAMR worker node encounters a failed forward model run, it is not immediately apparent to the master whether the failure is a result of a network communication error (or failure) or instability in the underlying forward model being run with the parameters estimated by PEST++. To test the underlying cause of failure, PEST++ allows the user to specify the maximum number of tries the PEST++ master will attempt using the parameter set of the failed forward run before marking the run as a failure. If a failed run occurs during the Jacobian matrix calculation, and if the DERFORGIVE algorithmic option is invoked in the PEST control file, then the parameter

associated with the failed perturbation run will be treated as frozen in the subsequent upgrade calculations. If the failed run is associated with parameter upgrade testing, then the upgrade vector associated with the failed run is ignored and the objective function is reported as "NA." Note that the efficiency of this capability is greatly enhanced by the multi-threaded worker nodes of YAMR because the penalty for starting concurrent attempts for a single model run is negligible.

Iteration Summary Suitable for Postprocessing

PEST++ Version 3 provides a new reporting capability that allows the user to also obtain comma-separated value (CSV) files of iteration-based PEST++ results, which are more amenable to postprocessing and visualization than were the output-file options in Version 1. Iteration-specific CSV summary files are provided for parameters (*.ipar*), base parameter sensitivities (*.isen*), and objective function components (*.iobj*).

PEST++ Performance Log File

PEST++ Version 3 writes a log file that details the process execution time for the most computationally demanding aspects of the algorithm. The file is written with a *.pfm* extension. Users are encouraged to monitor this file if PEST++ appears unresponsive.

PEST++ Version 3 User Interface

Similar to PEST++ Version 1, Version 3 maintains full backward compatibility with PEST input files; changes to the file formats and user interfaces have been minimized. This consistency facilitates switching between the enhancements available in PEST++ and the full capabilities of PEST. This compatibility design criterion embodies the concept that although PEST++ is appropriate for the majority of parameter estimation problems, there are expected to be a subset of inverse problems that will require additional, less-used capabilities available in PEST that are not yet available in PEST++. This design criterion facilitates PEST++ compatibility with the large set of utilities included in the PEST suite of tools.

In a similar manner, PEST++ Version 3 continues to retain 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 available beyond the linear uncertainty methods included in PEST++ Version 3 (for example, see Doherty 2010a,b; and Doherty and others, 2010). PEST++ also retains support for the legacy PEST control file (*.pst*), although it is anticipated that future versions of PEST++ will support more flexible control file formats. Similar to Version 1, PEST++ Version 3 offers additional functionality that can be accessed through optional lines beginning with "++" at the end of the PEST control file (see appendix 1). Because the truncated SVD solution scheme (Aster and Thurber, 2013) is optional in PEST but not in PEST++, PEST++ programmatically provides default values for the algorithmic parameters in the "* singular value decomposition" section of the PEST control file if these values are not specified by the user.

Development Environment

All the source code required to build PEST++ Version 3 has been consolidated in a Microsoft Visual Studio® 2013 solution. Source and Makefiles are also provided for Linux compilation. PEST++ Version 3 is compiled as a statically linked PEST++ executable without any external run-time dependencies. This setup simplifies and facilitates sharing the source code and distributing the executable. Note that PEST++ requires a C++11-compliant compiler as well as a modern Fortran compiler. The PEST++ Visual Studio solution, Makefiles, as well as source code and executable documented in this report are available for download at the Web page associated with this report: http://dx.doi.org/10.3133/tm7–C12. More recent releases of PEST++, including any enhancements made since the release of this report, are available at http://www.inversemodeler.org/. The most current version of the source code is maintained an online open-source version-control repository at https://github.com/dwelter/pestpp.

Limitations of Version 3

Similar to PEST++ Version 1, some features of the PEST suite are not available in Version 3. Most notable are the following:

- Use of a full observation covariance weights matrix is not supported.
- Similar to a SVD-Assist run in PEST, PEST++ does not perform a final run with the best parameters. A strategy needs to be developed to implement a final run when using parallel runs via YAMR or GENIE. However, PEST++ does provide the best-fit parameter values, as well as the model output corresponding to observations and residuals associated with the best model run, so that a user simply needs to update the PEST control file with the best parameters and run PEST++ with NOPTMAX set to 0.
- The Jacobian (*.jco)* file for superparameter iterations is written in terms of the superparameters, whereas PEST++ writes the base parameter Jacobian to a *.jcb* file. Both of these files are compatible with the PEST *.jco* file specification. The use of a *.jcb* is necessary to support restart capabilities in concert with the internalization of the superparameter solution process.

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

Capabilities of Version 1 of the object-oriented parameter estimation code PEST++ have been augmented and extended in Version 3. Similar to the design concepts of Version 1, PEST++ Version 3 does not reproduce all the functionality of PEST but has increased the subset of features important for parameter estimation of large, highly parameterized problems. These enhancements follow the Version 1 design goal of being object orientated and suitable for future development. The primary enhancements are the following:

- The widely used Gauss-Marquardt-Levenberg method was added as a robust numerical solution technique for the least-squares problem. This is in contrast to PEST++ Version 1, which used a trust region based on the SVD rotation factor.
- A multi-threaded TCP/IP parallel run manager has been integrated into PEST++ Version 3 and is invoked similarly to the parallel run manager in BeoPEST (Schreüder, 2009). The new integrated run manager is called YAMR ("Yet Another run ManageR") to distinguish its integrated capability from the independent or external parallel run manager capability provided by GENIE (Muffels and others, 2012) in PEST++ Version 1. The GENIE parallel run manager is retained in PEST++ Version 3 and is updated to GENIE Version 2 (appendix 2).
- The ability to dynamically adjust Tikhonov regularization weight has been included in Version 3. The theoretical and implementation aspects are consistent with the most widely used settings available in the PEST software suite.
- Version 3 incorporates the most widely used subset of the of uncertainty analysis tools available in the PEST software suite. The integrated analyses are currently restricted to linear uncertainty methods, because these methods often constitute an acceptable tradeoff between theoretical rigor and computational burden for highly parameterized environmental models.
- The program GSA++ for global sensitivity analysis is now included in the PEST++ software suite. This program implements two widely used approaches for estimating global sensitivity measures with a parallel run manager. GSA++ is fully compatible with the PEST/PEST++ file formats so that it can be easily and efficiently applied to existing projects. GSA++ is notable because it provides capability currently not available in the PEST software suite.
- Additional capabilities added to Version 3 include other user-friendly options such as the ability to restart a terminated run and enhanced output better suited for postprocessing.

Similar to Version 1, all code necessary to produce a statically linked PEST++ executable has been consolidated into the Microsoft Visual Studio 2013 solution and Makefiles for the Windows and Linux environments, respectively.

References

Anderson, M.P., Woessner, W.W., and Hunt, R.J., 2015, Applied groundwater modeling—Simulation of flow and advective transport (2d ed.): Amsterdam, The Netherlands, Elsevier, 610 p.

Aster, R.C., and Thurber, C.H., 2013, Parameter estimation and inverse problems (2d ed.): Waltham, Mass., Academic Press, 360 p.

Beven, Keith, 2009, Environmental modelling—An uncertain future?: CRC Press, 328 p.

Campolongo, F., Cariboni, J., Saltelli, A. and Schoutens, W., 2005, Enhancing the Morris Method, *in* Hanson, K.M., and Hemez, F.M, eds., Sensitivity analysis of model output—Proceedings of the 4th International Conference on Sensitivity Analysis of Model Output (SAMO 2004) Santa Fe, New Mexico, March 8–11, 2004: Los Alamos National Laboratory Research Library, p. 369–379.

Dahlstrom, D.J., and Carter, J.T.V., 2013, Inverse modeling with PEST++ and GENIE: Groundwater, v. 51, no. 2, p. 162–167, doi:10.1111/gwat.12021.

Dausman, A.M.; Doherty, John; Langevin, C.D.; and Sukop, M.C., 2010, quantifying data worth toward reducing predictive uncertainty: Ground Water, v. 48, no. 5, p. 729–740.

Doherty, John, 2003, Ground water model calibration using pilot points and regularization: Ground Water, v. 41, no. 2, p. 170–177.

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.E., 2015, Calibration and uncertainty analysis for complex environmental models (PEST—Complete theory and what it means for modelling the real world): Brisbane, Australia, Watermark Numerical Computing.

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., http://pubs.usgs.gov/sir/2010/5169/.

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., http://pubs.usgs.gov/sir/2010/5211/.

Golub, G.E., and Van Loan, C.F., 1996, Matrix computations (3d ed.): Baltimore, Md., Johns Hopkins University Press, 694 p.

Fienen, M.N., Doherty, J.E., Hunt, R.J., and Reeves, H.W., 2010, Using prediction uncertainty analysis to design hydrologic monitoring networks—Example applications from the Great Lakes Water Availability Pilot Project: U.S. Geological Survey Scientific Investigations Report 2010–5159, 44 p., http://pubs.usgs.gov/sir/2010/5159/.

Hill, M.C., and Tiedeman, C.R., 2007, Effective groundwater model calibration—With analysis of data, sensitivities, predictions, and uncertainty: Hoboken, N.J., Wiley-Interscience, 455 p.

Hunt, R.J., 2012, Uncertainty, *in* Australian groundwater modelling guidelines: Canberra, Australia, National Water Commission, Waterlines Report Series No. 82, p. 92–105.

Hunt, R.J.; Doherty, John; 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, Joseph; Schreüder, W.A.; Rumbaugh, J.O.; Doherty, John; 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, Catherine, and Doherty, John, 2005, The role of the calibration process in reducing model predictive error: Water Resources Research, v. 41, no. 5, W05020, 14 p., doi:10.1029/2004WR003501.

Morris, M.D., 1991, Factorial sampling plans for preliminary computational experiments: Technometrics, v. 33, no. 2, p. 161–174.

Muffels, C.T., Schreüder, W.A., Doherty, J.E., 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, chap. C6, 26 p., http://pubs.usgs.gov/tm/tm7c6/.

Oliver, D.S., Reynolds A.C., and Liu, Ning, 2008, Inverse theory for petroleum reservoir characterization and history matching (1st ed.): Cambridge, UK, Cambridge University Press, 380 p., http://dx.doi.org/10.1017/CBO9780511535642.

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.

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., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D. Saisana, M., and Tarantola, S., 2008, Global sensitivity analysis—The primer: Chichester, England, John Wiley & Sons Ltd., 292 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.

Schreüder, W.A., 2009, Running BeoPEST, *in* Tonkin, M.J., ed., Proceedings, PEST Conference 2009, Potomac, Md., November 1–3, 2009: Bethesda, Md., S.S. Papadopulos and Associates, p. 228–240.

Sin, G., and Gernaey, K., 2009, Improving the Morris method for sensitivity analysis by scaling the elementary effects: Computer Aided Chemical Engineering, v. 26, p. 925–930.

Sobol, I.M., 2001, Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates: Mathematics and Computers in Simulation, v. 55, p. 271–280.

Tikhonov, A.N., and Arsenin, V.Y., 1977, Solutions of ill-posed problems: New York, Halstead Press-Wiley, 258 p.

Tonkin, M.J., and Doherty, John, 2005, A hybrid regularized inversion methodology for highly parameterized models: Water Resources Research, v. 41, no. 10, W10412, 16 p., doi:10.1029/2005WR003995.

Vogel, C.R., 2002, Computational methods for inverse problems: Philadelphia, Pa., Society for Industrial and Applied Mathematics, 183 p.

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, chap. C5, 47 p., http://pubs.usgs.gov/tm/tm7c5/.

White, J.T., Doherty, J.E. and Hughes, J.D., 2014, Quantifying the predictive consequences of model error with linear subspace analysis: Water Resources Research, v. 50, no. 2, p. 1152–1173, doi:10.1002/2013WR014767.

Appendixes

Appendix 1. PEST++ Version 3 Input Instructions

The PEST++ Version 3 Visual Studio solution, as well as source code and executable as documented in this report, are available for download at https://www.usgs.gov/software/pest-parameter-estimation-code-optimized-large-environmental-models-version-3. More recent releases of PEST++, including any enhancements made since the publication of this report, will be available at http://www.pestpp.org/. The most current development version of the source code is maintained an online open-source version-control repository at https://github.com/dwelter/pestpp/.

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, 2010). The intent is to make PEST++ input and output compatible with the large number of existing PEST utilities (for example, Doherty, 2011a,b).

The PEST++ Command Line

PEST++ supports various command line options that control run manager invocation as well as restart options. PEST++ Version 3 supports three run mangers to complete the forward model runs: (1) Yet Another Run ManageR (YAMR), (2) GENIE, and (3) a serial run manager. YAMR and GENIE are sophisticated parallel run managers capable of performing parallel runs on a single machine or over a TCP/IP-enabled network. YAMR is integrated into PEST++ and is invoked similarly to BeoPEST (Schreüder, 2009). Although PEST++ provides an interface to the GENIE run manager, this interface relies on the external GMAN and GSLAVE programs (Muffles and others, 2012) to manage and perform the actual model runs. The serial run manager provides a simple alternative that mimics the functionality currently in PEST.

In addition to run manager specification, the command line also controls the restart functionality of PEST++.

The various options related to run manger and restart control are summarized in table 1–1, where /j and /r are optional commands; /j invokes Jacobian reuse for the first iteration, and /r invokes restart. When PEST++ is run with the serial run manager or as the master node with a parallel run manager, it now supports the /j option to reuse an existing binary Jacobian file rather than computing the Jacobian for the first iteration. Note that PEST++ can be restarted by using a Jacobian computed by PEST as long as the PEST++ "autonorm" option is not invoked in the control file.

Table 1–1. Summary of PEST++ command line options.

Run Manger / Mode	Command
Serial Run Manager / Master	`pest++.exe <casename>.pst [/j] [/r]`
YAMR / Master	`pest++.exe <casename>.pst /H :<port> [/j] [/r]`
YAMR / Worker Node	`pest++.exe <casename>.pst /H <hostname>:<port>`
GENIE / Master	`pest++.exe <casename>.pst /G <GENIE Master hostname>:<port> [/j] [/r]`
GENIE / Master	`genie.exe /port <port>`
GENIE / Worker Node	`genie.exe /ip <GENIE Master IP address> /port <port>`

The Pest Control File

For ease of reference, variables within the PEST control file are listed on the next three pages, and the variables used by PEST++ are shaded. PEST++ relies on the structure of the PEST control file (Doherty, 2010) to read the necessary algorithmic parameters and reads only those algorithmic parameters that are needed. For example, there is no need to read the NOBS variable because each line in the "observation data" section of the PEST 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
PARGPN MEINCTYP 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 PARGPSCALE 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
++ MAX_N_SUPER(20)  SUPER_EIGTHRES(1.0E-8)
++ N_ITER_BASE(1)  N_ITER_SUPER(3)
++ SVD_PACK(PROPACK)   AUTO_NORM(4)
++  LAMBDAS(0.1,1,10,100,1000)
++ MAX_SUPER_FRZ_ITER(5)
++ MAX_REG_ITER(20)
++ MAT_INV(inv_type)
++ SUPER_RELPARMAX(sup_relpar_max)
++ MAX_RUN_FAIL(3)
++ ITERATION_SUMMARY(TRUE)
++ DER_FORGIVE(TRUE)
++ UNCERTAINTY(TRUE)
++ FORECASTS(pred_1,pred_2,pred_3)
++ PARAMETER_COVARIANCE(prior_parameter.cov)
++ OVERDUE_RESCHED_FAC(2.0)
++ OVERDUE_GIVEUP_FAC(10.0)
```

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. PEST++ V3 only supports "parabolic."

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 WFMAX	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 $\mathbf{x^t qx}$ (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 file specific to PEST++ is marked by lines starting with "++". Although the examples provided in this report place all 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 file. Lines starting with "++#" are considered comments and are ignored by PEST and PEST++.

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. Although one can use parentheses to more clearly delineate the values assigned to the variable (for example, ++N_ITER_BASE(1) specifies N_ITER_BASE=1), these could just as well be replaced by white spaces (for example, ++N_ITER_BASE 1 also specifies N_ITER_BASE=1). Table 1–2 includes a listing and explanation of the permissible PEST++ keywords.

Table 1–2. PEST++ optional arguments.—Continued

Variable	Type	Values	Description
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 MAX_N_SUPER. Because PEST++uses SVD on the superparameter problem, a low value for this SUPER_EIGTHRES will not adversely impact the stability of the solution.
MAX_N_SUPER	Integer	integer between 1 and the minimum either of maximum number of parameters or the maximum number of observations	Maximum number of superparameters to use in the superparameter iterations.
MAX_REG_ITER	Integer	integer greater than 1; default is 20	Provides a limit on the maximum the number of iterations used to compute dynamic regularization weights when PEST++ is run in regularization mode. Setting this value too large can result in appreciable slowdown, especially in early iterations.
MAX_SUPER_FRZ_ITER	Integer	1 or greater; default value is 5	Maximum number of times a superparameter iteration will try to freeze any parameters that go out of bounds and try to recompute a Jacobian. If the Jacobian cannot be computed in MAX_SUPER_FRZ_ITER iterations, PEST++ will switch to a base parameter iteration.
AUTO_NORM(4)	Integer	1 or greater; default is no scaling	Automatically normalizes the sensitivities by assuming there are X standard deviations between the upper and lower parameter bounds, where X is the value passed with the AUTO_NORM variable (4 is shown).
SVD_PACK(PROPACK)	String	"JACOBI" or "PROPACK; default is "JACOBI"	Flag to use PROPACK to compute SVD factorizations. "JACOBI" is the SVD provided by the EIGEN library; "PROPACK" is the iterative SVD factorization suitable for large problems.
MAT_INV	String	"Q1/2J" or "JTQJ"; default is "JTQJ"	Flag to specify the formulation of the normal equation. This option is forced to "Q1/2J" when PROPACK is used.
SUPER_RELPARMAX	Real	greater than 0; default is 0.1	Parameter relative change limit for superparameters.

Table 1–2. PEST++ optional arguments.—Continued

Variable	Type	Values	Description
MAX_RUN_FAIL	Integer	greater than 0; default is 3	Maximum times the run manager will try to rerun a failed run.
LAMBDAS	Comma-separated list of reals	greater than 0; default is (0.01,1,10,100,1000)	Specify the standard values of lambda to be used each iteration.
ITERATION_SUMMARY	Boolean	"TRUE" or "FALSE"; default is "TRUE"	Setting this to "TRUE" will save a summary of each iteration to a series of comma-separated files for easy plotting.
DER_FORGIVE	Boolean	"TRUE" or "FALSE"; default is "TRUE"	Setting this to "FALSE" will turn off derivative forgive and cause PEST++ to terminate if a run fails while computing the Jacobian.
UNCERTAINTY	Boolean	"TRUE" or "FALSE"; default is "TRUE"	A flag to disable uncertainty analyses.
FORECASTS	Comma separated list of text	Observation names in the control file; default is none	The names of observations to treat as forecasts in the uncertainty analyses.
PARAMETER_COVARIANCE	Text	Filename; default is none	The name of a PEST-compatible ASCII matrix or uncertainty file to use as the prior parameter covariance matrix.
OVERDUE_RESCHED_FAC	Real	greater than 1.0; default is 1.15	YAMR specific command. If a model run takes longer than (OVERDUE_RESCHED_FAC * the average runtime) it will rescheduled on another node if one is available.
OVERDUE_GIVEUP_FAC	Real	greater than 1.0; default is 100.0	YAMR specific command. If a model run has been running longer than (OVERDUE_GIVEUP_FAC * the average runtime) it will canceled.

References

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

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

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

Muffels, C.T., Schreüder, W.A., Doherty, J.E., 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, chap. C6, 26 p., http://pubs.usgs.gov/tm/tm7c6/.

Schreüder, W.A., 2009, Running BeoPEST, *in* Tonkin, M.J., ed. Proceedings, PEST Conference 2009, Potomac, Md., November 1–3, 2009: Bethesda, Md., S.S. Papadopulos and Associates, p. 228–240.

Appendix 2. GENIE Version 2, A General Model-Independent TCP/IP Run Manager

By Christopher T. Muffels,[1] Douglas A. Hayes,[1] Matthew J. Tonkin,[1] and Randall J. Hunt[2]

Preface

GENIE Version 1 was documented in—

Muffels, C.T., Schreüder, W.A., Doherty, J.E., 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, chap. C6, 26 p., http://pubs.usgs.gov/tm/tm7c6/.

Capabilities of GENIE Version 1 have been expanded, called Version 2. This appendix documents the Version 2 enhancements. The suggested citation for Version 2 is—

Muffels, C.T., Hayes, D.A., Tonkin, M.J., and Hunt, R.J., 2015, GENIE Version 2, A general model-independent TCP/IP run manager, *appendix 2 of* Welter, D.E., White, J.T., Hunt, R.J., and Doherty, J.E., 2015, PEST++ Version 3—A Parameter ESTimation and uncertainty analysis software suite optimized for large environmental models: U.S. Geological Survey Techniques and Methods, book 7, chap. C12, 54 p.

Introduction

GENIE is a model-independent suite of programs external to PEST++ that can be used to generally distribute, manage, and execute multiple model runs via the TCP/IP network infrastructure. The suite consists of a Run Manager, a Run Executer, and a set of routines that can be compiled as part of a program and used to exchange model runs with the Run Manager. Because communication is via a standard protocol (TCP/IP), any computer connected to the Internet can serve in any of the capacities offered by this suite. Model independence is consistent with the existing template and instruction file protocols of the widely used PEST and PEST++ parameter estimation programs. GENIE was originally documented by Muffels and others (2012).

GENIE is a single program; the Run Manager and Run Executer are distinguished at runtime by using a command line switch. The program is intended to be stand-alone and does not require modification by users. Users need only modify their programs to call the provided integration routines that are a conduit/entry point to GENIE, which in turn handles all exchanges regarding the model runs. The Run Manager receives runs and distributes them to the different client computers, where they are executed (fig. 2–1).

Acknowledgments

Support for GENIE is also provided by the California Department of Water Resources (CDWR).

Limitations

- Unlike BeoPEST of Schreüder (2009), which includes MPI and TCP/IP communication, GENIE supports only TCP/IP communication.
- Support for IP v6 is available but is not tested in Version 2.
- 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.

[1] S.S. Papadopulos and Associates, Inc.
[2] U.S. Geological Survey.

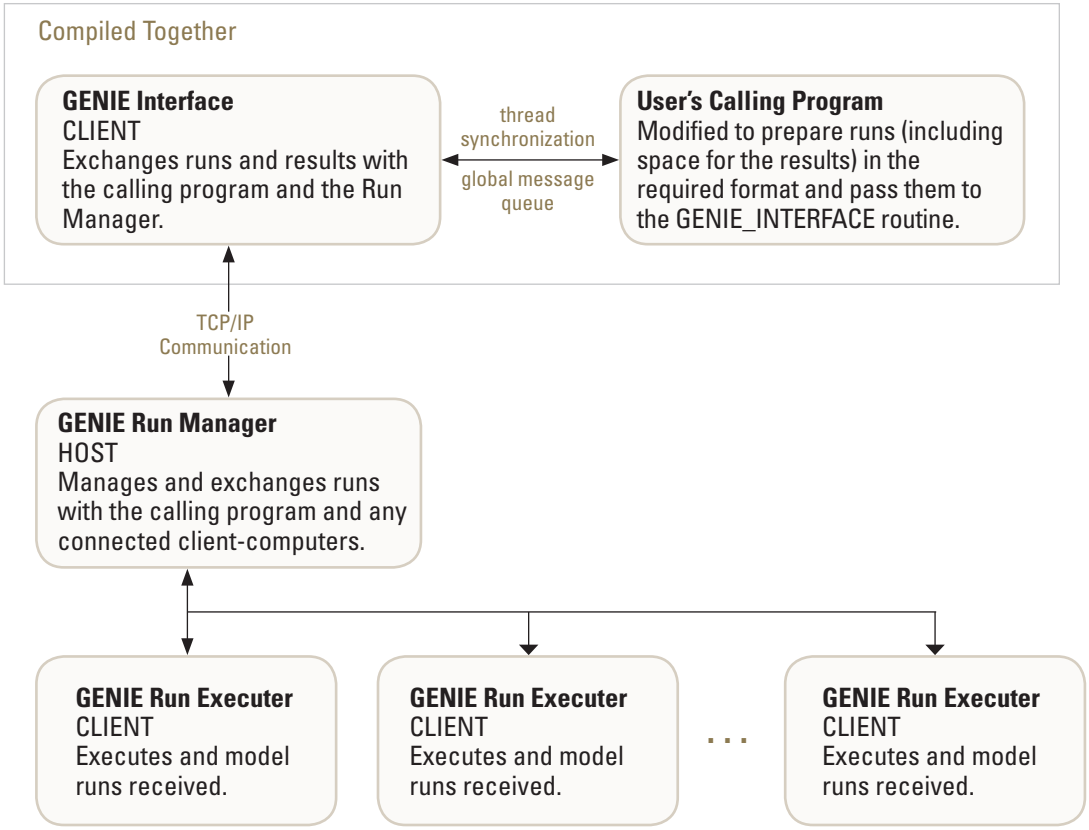


Figure 2–1. Flow of communication between the different components of the GENIE suite.

Updates to the Run Manager

Command Line Interaction

An interactive feature is provided that allows users to query different aspects of the Run Manager state or give it commands to perform. For example, it can be queried for a list of the connected clients or a list of the outstanding runs or can be instructed to terminate a specific client or model run. A complete list of the available queries and commands is presented in the table below. The user can enter "genie ?" at the command prompt for a list of available queries and actions.

Command	Arguments	Description
quit		Terminate Run Manager and any connected clients.
list	clients disconnects runs runs_inprogress runs_failed	Lists to the screen any members belonging to the requested argument.
end_client	all client_name	Terminates connection with all the connected clients, or just the supplied client.
end_run	all client_name	Terminates the run currently being executed by all clients, or just on the supplied client.

Termination

The Run Manager remains active unless instructed to terminate. The termination command can be provided from the calling program via the Interface or at the GENIE command prompt. The termination command from the Interface is only acknowledged if the "quitwhendone" option is specified at the command line. The default is to keep GENIE alive (and any connected clients) so that a calling program can be started and restarted as needed without having to restart the Run Manager or any clients.

Input Instructions

The Run Manager is started by executing the *genie.exe* file. This file can be copied to the desktop (or a shortcut to another folder that contains it) and simply double-clicked to start actions most easily accessed when running models over a local network only. In the event slave computers are to be used across the Internet, then optional command line switches may be required:

Switch	Description
/ip	Used to specify an IP address to use for communication. Only IP v4 has been tested to date. Example: /ip 192.168.0.1
/port	Used to specify a PORT to use for communication. This switch is most likely to be required when communicating with slave computers over the Internet. In most cases a specific PORT must be opened in the firewall to allow Internet communication. This switch is used to provide that PORT. Viable ports are between 1024 and 65535. Example: /port 4040
/runtime	The expected time (in minutes) of a model run. Used by the load balancer: if the execution time of a model run exceeds RUNTIME on a client, it will be started again on an idle client. If this switch is not supplied, the default is 10 minutes. Example: /runtime 0.5
/nfail	The number of times a failed model run is retried before the Run Manager returns the run as failed to the calling program. If this switch is not supplied, the default is 3. Example: /nfail
/quitwhendone	If specified, GENIE will terminate itself and any connected clients when finished.

Updates to the Run Executer (Worker Node)

Input Instructions

The Run Executer is started by executing the *genie.exe* file at the command line with the following, case-insensitive, required switches:

Switch	Description
/host	Used to specify the socket the Run Manager is listening on. Example: /host 192.168.0.1:4040
/name	Used to uniquely identify the slave computer to the Run Manager. Example: /name S1

The switches can be specified in any order, for example:

```
genie /interval 1.0 /console on /name S-1 /host 192.168.0.1:4040
```

The following table lists optional switches that can be used to define a specific socket for the Run Executer to communicate on.

Switch	Description
/ip	Used to specify an IP address to use for communication. Only IP v4 is supported in version 1.0. Example: /ip 192.168.0.1
/port	Used to specify a PORT to use for communication. This switch is most likely to be required when communicating with slave computers over the Internet. In most cases a specific PORT must be opened in the firewall to allow Internet communication for the Run Manager. This switch is used to provide that PORT to the Run Manager. Viable ports are between 1024 and 65535. Example: /port 4040
/interval	The number of seconds the Run Executer is idle before checking on a run. For long model run times, this number can be higher. If this number is unnecessarily low, the Run Executer will consume more CPU resources than it should. For short run times it is important to make this number small (0.1). The default is 10 seconds. Example: /interval 0.1
/console	Indicates whether the RUN should be executed in a visible console window or not. This switch can be either ON, OFF, or PIPE. If it is ON, then the RUN console is visible. If PIPE is specified, the STDERR and STDOUT of the run process are piped to the GENIE console window. The default is OFF. Example: /console ON

Updates to the Interface Routines

The Interface routine required by the calling program (that is, PPEST or PEST++) to exchange information with the Run Manager is a threaded function operating in parallel with the outside program. Thus, the calling program never loses focus, and it and the Interface operate (somewhat) independently of each other, giving control to the outside program developer. The calling program is able to make decisions as results come in and communicate new directives to the Interface as needed. Effort was made to make the implementation of the Interface in the calling program straightforward. To this end, self-evident routine names were used, and the threading and TCP/IP communication programming is "hidden" from uninterested users.

The outside program needs to prepare the model run information in the format required by GENIE and then, using the available function calls, exchange runs with the Interface, which in turn forwards the requests onto the GENIE Run Manager. The Interface routines, listed in the table below, are available in the *genie_interface.lib* library file that can be compiled against statically from the calling program. The routines are written in C++ but contain the necessary external functions for compiling with Fortran.

Function	Description
genie_interface_v2(char *ip_port)	Initializes the Interface, which makes first contact with the GENIE Run Manager. IP_PORT is the socket required to connect to the GENIE Run Manager.
genie_interface_v2_terminate()	Terminates both the Interface and Receiver threads, and disconnects from the GENIE Run Manager and cleans up any shared global memory.
initialize_synchronization_variables()	Initializes the different synchronization variables utilized by the Interface and calling program.
destroy_synchronization_variables()	Properly disposes of the synchronization variables.
signout_response_mutex()	Signs out the mutex for the shared message Response queue. This mutex is used to guard the queue to ensure it is manipulated by only one thread at a time.
signin_response_mutex()	Signs in the mutex for the shared message Response queue.
signout_cp_mutex()	Signs out the mutex controlling access to the condition variable the calling program is blocking on.
signin_cp_mutex	Signs in the mutex controlling access to the calling program condition variable.
wait_for_signal_from_receiver()	Blocks (indefinitely) until a message is returned by the Receiver thread.
timedwait_for_signal_from_receiver()	Blocks, for a limited time or until a message is returned by the Receiver thread.
signal_interface()	Signals the Interface that there is a message to be sent to the GENIE Run Manager.
add_message_to_request_queue(cls_message *msg)	Provides a (single) run to the Interface after it has been converted to a message. Used if the calling program is C++ based. Note: a run is easily converted to a message by using the appropriate constructor.
give_fortran_run_to_interface(int *runid, int *nexec, char *_execnams, int *npar, int *nobs, char *_apar, char *_aobs, double *pval, double *oval, int *ntpl, int *nins, char *_tplfle, char *_infle, char *_insfle, char *_oufle)	Used to provide a (single) run to the Interface when called from a Fortran-based calling program. See table 2–2 for a description of the variables passed to this function.
get_number_of_responses()	Returns the number of responses (received by Receiver thread) in the global queue.
fortran_get_result_from_response_queue(int *runid, int *npar, int *nobs, double *pval, double *oval)	Gets a single response from the global queue.
reset_write_counter() report_runid_to_console(int *pid) report_runid_to_log(int *pid)	These routines can be used to write a particle ID to either the console or the GENIE_Interface_Misc log file. Particle IDs are written 20 per line, reset_write_counter resets the counter used to track how many IDs were written.

Figure 2–2 conceptualizes the use and interaction of a calling program with the GENIE Interface. Because the Interface is started as a thread from within the calling program, thread synchronization (mutexes and condition variables) and shared global queues are used to exchange messages between the two.

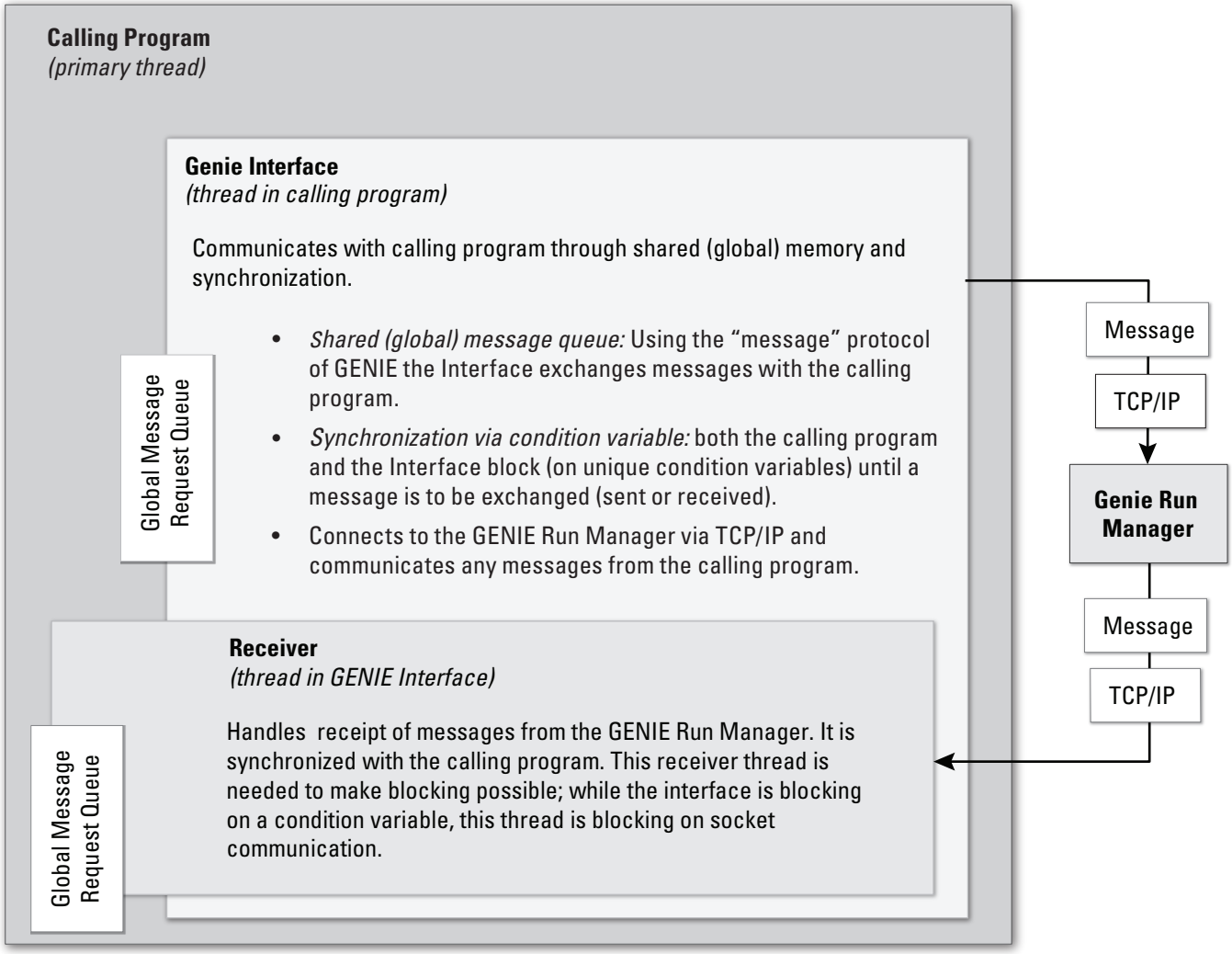


Figure 2–2. Conceptualization of the use and interaction of a calling program and the GENIE Interface.

Table 2–1. Definition of Parameters Required by Routine "give_fortran_run_to_interface."

Parameter	Type	Description
NRUN	Integer	The number of model runs to be managed and executed.
NEXEC	Integer	The number of executable files to be processed as part of a run; currently must be 1 and multiple executables listed in a batch file.
_EXECNAMS	String (NEXEC)	Array, of size NEXEC, listing each executable file.
NPAR	Integer	Number of parameters.
NOBS	Integer	Number of observations.
_APAR	String (NPAR)	Array, of size NPAR, listing each parameter name.
_AOBS	String (NOBS)	Array, of size NOBS, listing each observation name.
PVAL	Double C/C++: (NRUN,NPAR) Fortran: (NPAR,NRUN)	Array, of size NRUN * NPAR, listing each parameter value for each run. Values must be listed in the same order as for _APAR; that is, PVAL(1,1) is the value corresponding to the parameter named _APAR(1,1). Can be 2D array (NRUN,NPAR) or an equivalent 1D array. Because Fortran is column-major, the equivalent Fortran array must be dimensioned and filled as (NPAR,NRUN).
OVAL	Double C/C++: (NRUN,NPAR) Fortran: (NPAR,NRUN)	Array, of size NRUN * NOBS, listing each observation value for each run. Values must be listed in the same order as for _AOBS; that is, OVAL(1,1) is the value corresponding to the observation named _AOBS(1,1). Can be 2D array (NRUN,NOBS) or an equivalent 1D array. Because Fortran is column-major, the equivalent Fortran array must be dimensioned and filled as (NOBS,NRUN).
NTPL	Integer	Number of template files.
NINS	Integer	Number of instruction files.
_TPLFLE	String (NTPL)	Array, of size NTPL, listing the template file names.
_INFLE	String (NTPL)	Array, of size NTPL, listing the model input file names. These file names must be listed in the same order as for _TPLFLE; that is, _INFLE(1) is the model input file whose template is provided by file _TPLFLE(1).
_INSFLE	String (NINS)	Array, of size NINS, listing the instruction file names.
_OUFLE	String (NINS)	Array, of size NINS, listing the model output file names. These file names must be listed in the same order as for _INSFLE; that is, the instructions to read _OUFLE(1) are listed in file _INSFLE(1).
ID	String	A string identifying the calling program to the Run Manager – for example "ppest" or "pest++"

The following figures, 2–3*A* and 2–3*B*, outline the basic logic and Interface routine calls required by the calling program to exchange runs with the Run Manager.

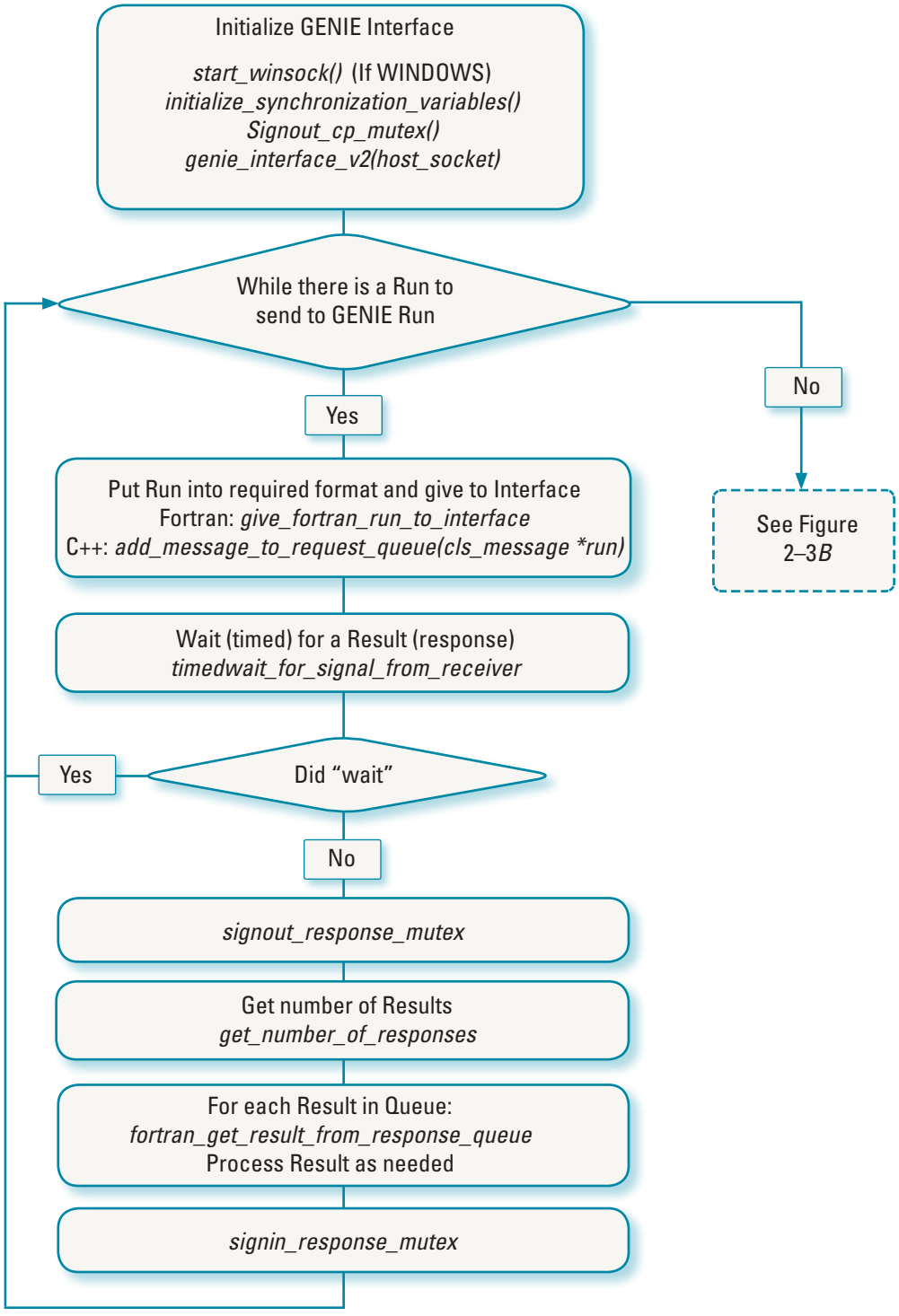


Figure 2–3*A*. Flow of logic in the calling program to communicate model runs to the Interface.

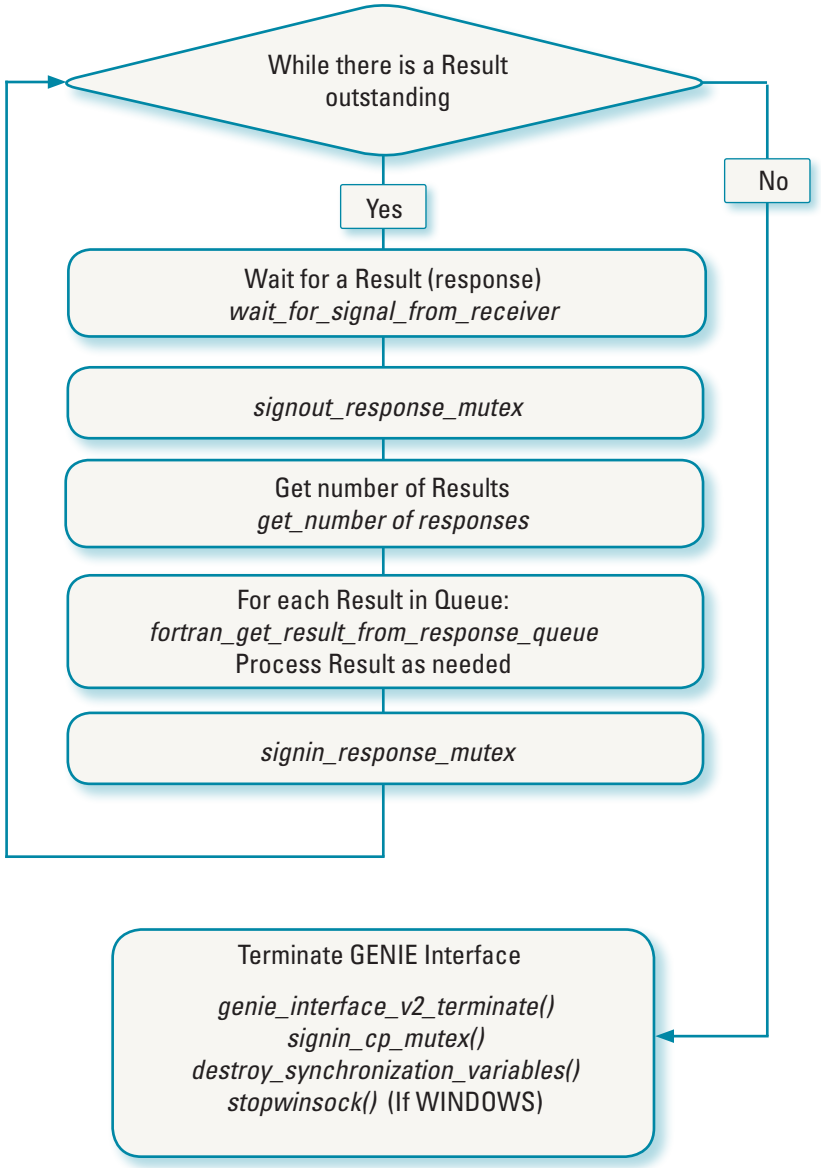


Figure 2–3*B*. Flow of logic to get results returned by the Interface (receiver thread).

Output

The GENIE Interface writes limited information to a series of files:

- *genie_interface_v2.log* – Interface thread log,
- *receiver.log* – Receiver thread log, and
- *GenieInterfaceSTDERR.log* – STDERR is written to this file and not the console.

Example Application—Interfacing GENIE with PPEST

This section describes in detail the steps taken to link Parallel PEST (PPEST; Doherty, 2010) to GENIE. The link primarily consists of exchanging model runs required and associated results with the Run Manager using the Interface. There are four instances in which PPEST may execute a model run or a collection of runs:

- the initial model run to get the current objective function,
- during sensitivity matrix calculation,
- during parameter update calculation if using the Levenberg-Marquardt technique, and
- the final model run using best parameter values.

The PPEST parallel Run Manager is encapsulated in a subroutine called DORUNS in source file *parpest.f*. This routine serves a role similar to the GENIE Run Manager—it is responsible for distributing and collecting model results as they become available. It is called during each of the four instances listed above. PPEST stores the parameter and observation values for different model runs in direct-access binary files. The runs distributed by the DORUNS routine are constructed from these files. The most significant change to the PPEST program was development of an additional source file called *genie.f* that contains the routines that prepare the PEST data for use with the GENIE Interface and then call this routine. This file is of most interest to developers wishing to link with the GENIE suite and is available in the GPEST source code section of the GENIE software download.

Most important in the *genie.f* file is a DORUNS-like routine called DORUNS_GENIE. DORUNS_GENIE gathers, one at a time, runs from the various binary files used by PEST to store parameter and observation information, puts them in the necessary arrays required by the give_fortran_run_to_interface routine, calls the routine to give the run to the Interface and, as results are returned, processes them in a manner similar to DORUNS. All of the modules and subroutines contained within *genie.f* are listed below.

Subroutine	Description
GENIE_DATA	Module. Contains the host socket information. These values are set within *pest.f* from the command line.
checkhost	Routine to verify the host provided is valid. Currently this check is limited to IP:PORT division.
DORUNS_GENIE	Routine equivalent to PPEST routine DORUNS for use with GENIE.

All of the changes made to the existing PPEST source code are contained within GENIE (#ifdef GENIE) preprocessor definitions. The following lists the subroutines that were modified, including the source file they belong to.

Subroutine	File	Description	Change
slavdat1	*parpest.f*	Opens the Run Manager file and reads the number of slaves.	Initializes the variables set by this routine because the details are not required.
slavedat2	*parpest.f*	Reads part of the Run Manager file and tests part of the information in it.	Skips details and writes host socket and other details to PPEST run management record (RMR) file.
writslv2	*parpest.f*	Summarizes slave properties to the RMR file.	Added property statements to reflect use of GENIE.
run_pest	*runpest.f*	Main PEST subroutine. Executes the functionality of PEST.	Added calls to DORUNS_GENIE.
parse_command_line	*pest.f*	Parses the PEST command line.	Modified to get host socket.
pest	*pest.f*	Main program—initialization and call to run_pest.	Modified to indicate use of GENIE.

Updates to the GENIE Programming

This section details some of the classes developed for the GENIE suite that will be of use to developers interested in linking existing programs with GENIE. The following is a list of the generic classes developed for the GENIE suite of programs. Programming-specific details of these classes are provided in the subsections that follow.

Class (or Object)	Used by	Description
	Network programming classes	
CLS_NODE (socketstuff.h)	Run Manager, Run Executer, and the Interface	Contains the bulk of the functionality required to start a node (client or host) using TCP/IP for communication.
	Message-passing classes	
CLS_HEADER (message.h)	Run Manager, Run Executer, and the Interface	The header portion of a message.
CLS_BUFFER (message.h)	Run Manager, Run Executer, and the Interface	A generic buffer class typically used to store the data portion of a message.
CLS_MESSAGE (message.h)	Run Manager, Run Executer, and the Interface	The low-level message-passing infrastructure developed for GENIE.
	Multi-threaded classes	
CLS_CONNECTOR (connector.h)	Run Manager	A thread to monitor and accept incoming connection requests.
CLS_BALANCER (balancer.h)	Run Manager	A thread to handle the load balancing features of the Run Manager.
CLS_MONITOR (monitor.h)	Run Manager, and Run Executer	A thread spawned for every client instance that monitors communication between it and the host.
CLS_PIPER (piper.h)	Run Executer	A thread to handle the piping of an application's console output to the GENIE console when the "/console pipe" option is used.
CLS_RUNNER (runner.h)	Run Manager and Run Executer	A thread to either distribute a run to a client (Run Manager) or execute a run (Run Executer)
	"Run" classes	
CLS_EXECUTABLE (run.h)	Run Manager, Run Executer, and the Interface	The definition of an executable; i.e. name, path, and arguments.
CLS_RUN (run.h)	Run Manager, Run Executer, the Interface	The definition of a complete model run.
CLS_RESULT (run.h)	Run Manager, Run Executer, and the Interface	The result portion of a model run, including functionality to send and receive this aspect of a run.

The following subsections detail the specific methods and properties of classes that will be of most use to developers; namely, those related to TCP/IP communication, the message-passing infrastructure, and multi-threading. These classes are very general and do not necessarily have to be used together to write a run-management-type program.

Network Programming

CLS_NODE

The CLS_NODE class contains all the functionality of a host (Run Manager) and a client (Run Executer, Interface).

Attribute	Description
ip (public)	(property – string) The IP of the NODE
port (protected)	(property – unsigned int) The PORT of the NODE
sockets (protected)	(property – set<int>) The set of sockets the NODE communicates on.
cls_node(int port)	(constructor method – void) HOST constructor instance
cls_node(string hostip)	(constructor method – void) CLIENT constructor instance

Message Passing

CLS_HEADER

The CLS_HEADER class is a set of properties comprising the header of a message. The header is a fixed size (16 bytes currently) that dictates how to read the rest of a message.

Attribute	Description
type	(property – integer) The type of the message. Can be one of the following defined in definitions.h: COMMAND, STATUS_UPDATE, CONNECTION_TYPE, CONNECTION_NAME, CONNECTION_SPEED, or RUN
bytesize	(property – integer) The number of bytes constituting a single element of the buffer portion of a message
nbytes	(property – integer) The number of elements in the buffer portion of a message.
compression	(property – integer) Currently not used.

CLS_BUFFER

The CLS_BUFFER class is used to store the data portion of a message.

Attribute	Description
size	(property – integer) The size of the buffer (supplied by CLS_HEADER).
buf	(property – character pointer) The buffer.
copyfrom	(method – void) Copy source memory block into buf. Uses C memcpy function.
copyto	(method – void) Copy buf to a user-specified memory block. Uses C memcpy function.

CLS_MESSAGE

The CLS_MESSAGE class is used to prepare and send a message

Attribute	Description
type	(property – integer) The number of bytes in the data component of the message.
hdr	(property – header) The header. See header class.
bdy	(property – buffer) The data portion of the message. See buffer class.
cls_message(int type)	(constructor method – void) An empty message (buffer size is zero) containing only a TYPE.
cls_message(int type, int data)	(constructor method – void) A message with an INTEGER buffer type.
cls_message(int type, size_t data)	(constructor method – void) A message with a SIZE_T buffer type.
cls_message(int type, std::string data)	(constructor method – void) A message with a STRING buffer.
cls_message(int type, double data)	(constructor method – integer) A message with a DOUBLE buffer type.
cls_message(cls_run *run)	(constructor method – void) Used to construct a message from a RUN object.
cls_message(cls_result *result)	(constructor method – double) Used to construct a message from a RESULT object.
cls_message(int socket, int *ifail)	(constructor method – integer) Receives and constructs a message from a TCP/IP communication.
receivedata	(method – integer) Routine to receive the data portion of a message from a CLIENT according to the header.
sendme	(method – integer) Sends a message to a CLIENT

Model Run

CLS_RUN

The CLS_RUN class contains all the properties and methods required to define a model

Attribute	Description
id	(property, pointer – integer) Assigned by GENIE_INTERFACE, ID is a unique identifier for a run within a run collection.
npar	(property, pointer – integer) The number of parameters.
nobs	(property, pointer – integer) The number of observations.
ntpl	(property, pointer – integer) The number of template files.
nins	(property, pointer – integer) The number of instruction files.
nexec	(property, pointer – integer) The number of executables (currently assumed to be 1).
tplfiles	(property, pointer – string) A pointer to an ntpl-element array that contains the name of each template file.
insfiles	(property, pointer – string) A pointer to an nins-element array that contains the name of each instruction file.
infiles	(property, pointer – string) A pointer to an ntpl-element array that contains the name of each model input file.
outfiles	(property, pointer – string) A pointer to an nins-element array that contains the name of each model output file.
parnams	(property, pointer – string) A pointer to an npar-element array that contains the name of each parameter.
obsnams	(property, pointer – string) A pointer to an nobs-element array that contains the name of each observation
parvals	(property, pointer – double) A pointer to an npar-element array that contains the value of each parameter.
obsvals	(property, pointer – double) A pointer to an nobs-element array that contains the value of each observation.
executables	(property, pointer – EXECUTABLE) A pointer to an instance of the CLS_EXECUTABLE class.
cls_run(cls_message *message)	(constructor-method – void) Creates a RUN object from a CLS_MESSAGE message.
cls_run(int *runid, int *nexec, char *_execnams, int *npar, int *nobs, char *_apar, char *_aobs, double *pval, double *oval, int *ntpl, int *nins, char *_tplfle, char *_infle, char *_insfle, char *_oufle)	(constructor-method – void) Creates a RUN object from arrays corresponding to the different properties of a RUN. This constructor is required by a calling program to create a CLS_MESSAGE object from arrays. This constructor is used in the *give_fortran_run_to_interface* of the GENIE Interface.

CLS_RESULT

CLS_RESULT is similar to the CLS_RUN class, but it contains only a property referencing a base CLS_RUN object and constructor methods to convert a CLS_RUN or CLS_MESSAGE object into the necessary format—the observation values and the adjusted parameter values given the precision they could be written to the model input file.

References

Doherty, John, 2010, PEST, Model independent parameter estimation—User Manual (5th ed., with slight additions): Brisbane Australia, Watermark Numerical Computing, 336 p.

Muffels, C.T., Schreüder, W.A., Doherty, J.E., 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, chap. C6, 26 p., http://pubs.usgs.gov/tm/tm7c6/.

Schreüder, W.A., 2009, Running BeoPEST, *in* Tonkin, M.J., ed. Proceedings, PEST Conference 2009, Potomac, Md., November 1–3, 2009: Bethesda, Md., S.S. Papadopulos and Associates, p. 228–240.

Appendix 3. Example Problem Using GML and Tikhonov Regularization

Introduction

A simple synthetic groundwater model is used to demonstrate the combined use of the GML algorithm with Tikhonov regularization. The inversion was completed with both PEST++ Version 3 and PEST (Doherty, 2010) so that users can compare the regularized solution and algorithmic behavior of both codes.

Model Description

The synthetic model is a cross-section MODFLOW-2005 (Harbaugh, 2005) model (X–Z domain) with 1 layer, 1 row, and 10 columns. The top and bottom of the model domain are bounded by no-flow boundaries (Neumann-type boundary condition where flux is specified as zero); the right side of the model domain is a specified flow boundary representing inflow from upgradient; the left boundary is a specified head (Dirichlet-type) boundary (fig. 3–1). Conceptually, groundwater enters the domain on the right and exists on the left. The model has two steady-state stress periods corresponding to a calibration period and a forecast period of reduced flux through the right boundary.

The inverse problem has 10 parameters, which are the horizontal hydraulic conductivity of each model cell, and 2 observations, collected from model cells 4 and 6 at the end of the calibration stress period. Both observations were assigned a weight of 10.0, which corresponds to a measurement noise standard deviation of 0.1 meter. Because observation weights were specified according to assumed measurement noise, the PHIMLIM variable should equal the number of non-zero weight observations, which in this case is 2.0. Tikhonov regularization was used enforce a condition of preferred parameter values. To demonstrate the computational savings and ease of use of the automatic superparameter solution scheme in PEST++, the PEST++ optional variables N_ITER_BASE and N_ITER_SUPER were set to 1 and 4, respectively.

Results

The model was inverted with both PEST (Doherty, 2010) and PEST++ so that the functionality of the combined GML algorithm and Tikhonov regularization as implemented in PEST++ Version 3 can be compared to the complementary functionality implemented in PEST. The inverted parameters found with PEST++ and PEST are very similar, although the PEST++ solution is slightly more regularized (smoother) than the PEST solution (fig. 3–2). PEST++ terminated after five iterations of the GML algorithm, whereas PEST continued until the maximum 20 iterations were complete. However, the minimum measurement objective function found by PEST++ is 2.07, which compares well to the PEST minimum of 2.09. For this simple synthetic case, PEST++ found a slightly more regularized solution with a slightly lower minimum objective function in five iterations than was found by PEST in 20 iterations. However, examination of the iteration-specific algorithmic behavior of PEST++ and PEST shows that the two different codes function very similarly (fig. 3–3).

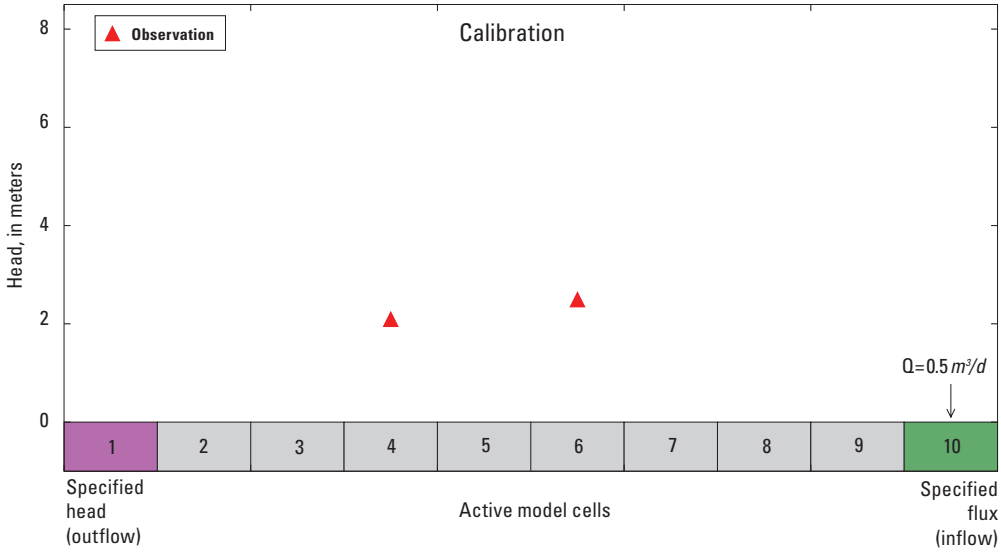


Figure 3–1. The synthetic problem test domain.

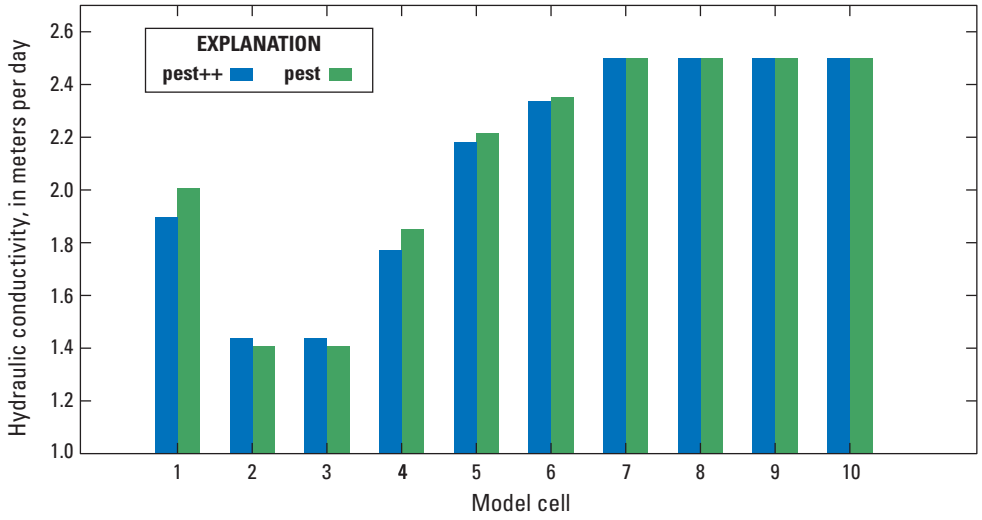


Figure 3–2. Comparison of the final inverted parameter values. The PEST++ solution is slightly more regularized than the PEST solution.

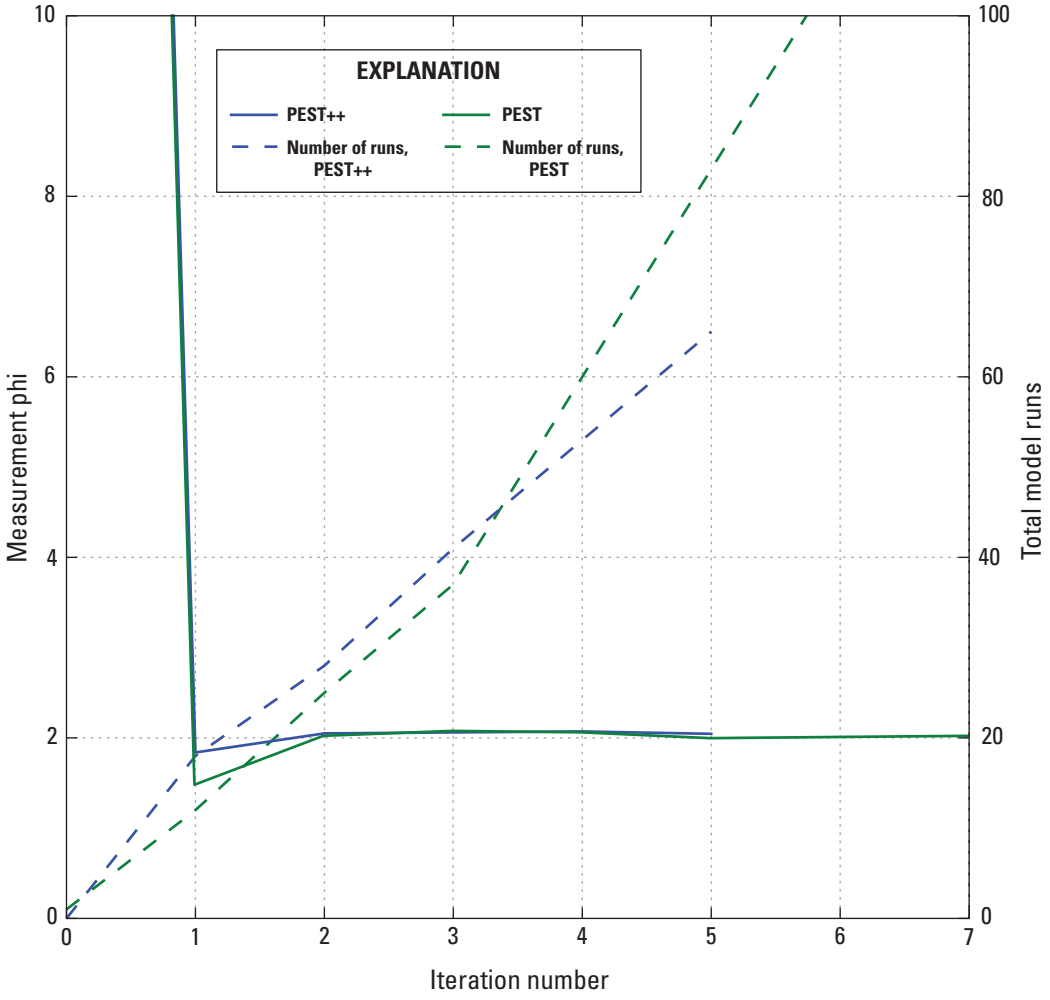


Figure 3–3. Comparison of the algorithmic behavior of PEST++ Version 3 and PEST version 13.3. The target minimum measurement phi (the PHIMLIM variable) is 2.0. PEST++ terminates after five iterations; PEST terminates after 20 iterations. PEST++ has less "overshoot" of the target phi after iteration 1.

References

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

Harbaugh, A.W., 2005, MODFLOW-2005, The U.S. Geological Survey modular ground-water model—The Ground-Water Flow Process: U.S. Geological Survey Techniques and Methods, book 6, chap. A16 [variously paged].

Appendix 4. Linear Uncertainty Methods Included in Version 3

Linear-based uncertainty analyses have been implemented in PEST++ Version 3 so that, at the completion of an inversion analysis, users can easily obtain estimates of parameter (and optionally forecast) uncertainty. The integrated analyses replicate the behavior of PREDUNC1 and PREDUNC 7 (Doherty, 2010). Similar to PEST++, the code that implements the linear uncertainty analysis follows object-oriented design principles and is built into the PEST++ code base. The implementation includes several classes for handling generic and structured (covariance) matrices and uses compressed (sparse) matrix storage and operations. The integration of these uncertainty analyses has been structured so that minimal user input is needed, in contrast to existing software for linear analysis such as the PREDUNC suite of tools (Doherty, 2010). However, less involvement from the user also means that many choices related to the implementation of the uncertainty analyses are programmatically assigned and are opaque to the user, including the following:

- If the optional "forecasts" PEST++ argument is specified, the names associated with this argument are assumed to be observations in the PEST control file. If one or more of these forecast observations is not found in the observation data section of the PEST control file, PEST++ will raise an exception. Once the inversion is complete, the rows of the last base parameter Jacobian matrix associated with these observations are extracted and treated as forecast sensitivity vectors (see equation 2). If the "forecasts" argument is not found, only parameter uncertainty estimates are calculated.

- All prior information (PEST control file observations and prior information groups that begin with "regul") are excluded from $\boldsymbol{J}$ and $\boldsymbol{\Sigma}_{\varepsilon}$ of equation 1 during uncertainty estimation.

- The matrix $\boldsymbol{\Sigma}_{\varepsilon}$ of equation 1 is formed from the observation weights listed in the PEST control file and is scaled to account for the contribution of each observation group to the final composite objective function so that $\boldsymbol{\Sigma}_{\varepsilon}$ reflects the level of measurement noise implied by the final composite objective function. Note that observations listed in the PEST control file with zero weight are assigned an artificial weight of 1.0E−30.

- If a prior parameter covariance matrix ($\boldsymbol{\Sigma}_{\theta}$ of equation 1) is not passed as a PEST++ option (the PARAMETER_COVARIANCE argument, see below), then this matrix if formed from the parameter bounds listed in the PEST control file. The variance of a non-log-transformed parameter p, σ_p, is calculated as

$$\sigma_p = \left(\frac{UL_p - LL_p}{4} \right)^2 \tag{1}$$

  where UL_p and LLp are the upper and lower bounds of parameter p listed in the PEST control file. This formulation assumes that the prior distribution of parameters is Gaussian and that the upper and lower bounds correspond to the upper and lower 95-percent confidence interval (±2 standard deviations). Similarly, for log-transformed parameters, the prior variance is calculated from parameter bounds as

$$\sigma_p = \left(\frac{\left(\log_{10}(UL_p) - \log_{10}(LL_p)\right)}{4} \right)^2 \tag{2}$$

Input Instructions

Several new PEST++ options must be added to control the use of integrated linear uncertainty analysis. These are listed in appendix 1 and are discussed in detail below:

- UNCERTAINTY(True).— A Boolean flag to enable or disable parameter and predictive uncertainty analyses. It is internally set to true by default. If UNCERTAINTY(False) is found in the PEST++ options, no uncertainty calculations will be performed, regardless of what other uncertainty-related input options are specified.

- FORECASTS(*obs_1,obs_2,obs_3*).— Identify PEST control file observations obs_1, obs_2, and obs_3 as predictions. As many observations as desired can be identified, as long as at least one observation remains a nonforecast. Equations 1 and 2 will be used to estimate the prior and posterior uncertainty for each forecast. If this argument is not found in the PEST control file, only parameter uncertainty analyses will be completed. A Pest Error is raised if one or more forecasts are not found in the observation names. A warning is issued if a forecast is assigned a non-zero observation weight.

- PARAMETER_COVARIANCE(*prior_parameter.cov*).—Identifies an existing covariance matrix (for example, *prior_parameter.cov*) to use as the prior parameter covariance matrix ($\boldsymbol{\Sigma}_{\theta}$ of equation 1). The format of the external file must be either a PEST-compatible ASCII matrix file or a PEST-compatible uncertainty file (see Doherty, 2010, for a description of these simple file types: p. 43 for ASCII matrices and p. 194 for uncertainty files). If this argument is not passed, if the external file cannot be loaded properly, or if the file is not compatible with the current inverse problem, a warning is issued and the prior parameter covariance matrix is constructed from parameter bounds as discussed previously. The user must ensure that an external parameter covariance file respects the log-transform status of the parameters as listed in the PEST control file (and as represented in the corresponding Jacobian matrix).

  Alternatively, the parameter bounds specified in the PEST control file can be used to construct a diagonal prior parameter covariance matrix. This is the default behavior and requires that the parameter bounds correspond to 95-percent confidence intervals.

Output

Several types of output are generated as part of the linear uncertainty analysis calculations. The posterior parameter covariance matrix ($\boldsymbol{\Sigma}_{\theta}$ of equation 1) is written to a PEST-compatible ASCII matrix file named <*case*>.*post.cov* where <*case*> is the base name of the PEST control file for the current PEST++ analysis. Additionally, a parameter uncertainty summary is written the record file (<*case*>.*rec*) of the current PEST++ analysis and to a comma-separated value (CSV) file named <*case*>.*par.usum.csv*. This uncertainty summary lists the prior and posterior mean, variance, and upper and lower bound for all adjustable parameters. If forecasts were identified, then a similar forecast uncertainty summary is written to the record file, as well as to a CSV file named <*case*>.*pred.usum.csv*.

For inverse problems with observation numbering greater than 50,000, the linear uncertainty calculations may take more than 30 minutes of clock time. During this time, the user can monitor the progress of the calculations in the PEST++ performance log file (*case.pfm*), which lists the starting, ending, and elapsed time for the most computationally demanding aspects of the analysis.

Additional Considerations

A robust estimate of parameter and predictive uncertainty requires the contribution from all uncertain model inputs treated as adjustable parameters. An artificially low, nonconservative estimate of uncertainty can be found by using a subset of these uncertain inputs as adjustable parameters. Additionally, not adjusting all uncertain inputs as parameters in the inverse problem can produce substantial model error that may bias parameter and forecast uncertainty estimates in ways that are not readily apparent. Users are therefore encouraged to include all adjustable parameters in the uncertainty analysis process even if not included for the initial parameter estimation. See White and others (2014) for more detailed discussion of this topic.

References

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

White, J.T., Doherty, J.E., and Hughes, J.D., 2014, Quantifying the predictive consequences of model error with linear subspace analysis: Water Resources Research, v. 50, no. 2, p. 1152–1173, doi:10.1002/2013WR014767.

Appendix 5. Example Problems Using PEST++ Version 3 Linear Uncertainty Capabilities

Introduction

This example demonstrates the use of the PEST++ integrated linear-based uncertainty analyses with a density-dependent SEAWAT model (Langevin and others, 2007) based on the analysis described in Herckenrath and others (2011) which is patterned after the Henry saltwater intrusion problem (Henry, 1964) . The outputs of both PEST++ and the complementary PEST utilities are presented so that users can review and verify the results.

Model Description

The synthetic model is a 2-dimensional SEAWAT model (X–Z domain) with 1 row, 120 columns, and 20 layers (fig. 5–1). The left boundary is a specified flux of freshwater; the right boundary is a specified head and concentration representing an infinite saltwater boundary. Conceptually, freshwater enters the domain on the left and a classic saltwater-freshwater interface "wedge" is formed where to lower-density freshwater overrides the saltwater and discharges in the upper right. The model has two stress periods: an initial steady state (calibration) period, then a transient period with less flux, which causes the saltwater-freshwater interface to move to the left. The second stress period serves as the forecast period.

The inverse problem has 601 parameters (600 hydraulic conductivity pilot points and 1 specified flux rate) and 36 observations (21 heads and 15 concentrations) measured at the end of the steady-state calibration period. The forecast of interest is the distance from the left boundary to the toe of the saltwater wedge (model layer 20) at the end of the forecast period. For this model, the toe of the wedge is characterized by the 1-, 10- and 50-percent seawater concentrations in model layer 20, which correspond to observations named pred_one,pred_ten, and pred_half .

To verify the PEST++ integrated linear uncertainty analysis results, the same analyses were performed by using the legacy PEST utilities. To accomplish the PEST++ integrated linear analyses, as well as the preprocessing and postprocessing (such as observation weight scaling, removing prior information, etc.), several PEST utilities and a text editor are needed. The utilities include PWTADJ2, PREDUNC7, PREDUNC1, and JCO2JCO (Doherty, 2010). The results of both PEST++ integrated uncertainty analysis, as well as a complementary uncertainty analysis using PEST utilities, are presented below for comparison. Note the analyses using the PEST utilities were completed with the base parameter Jacobian matrix filled by PEST++ with the NOPTMAX variable set to −1.

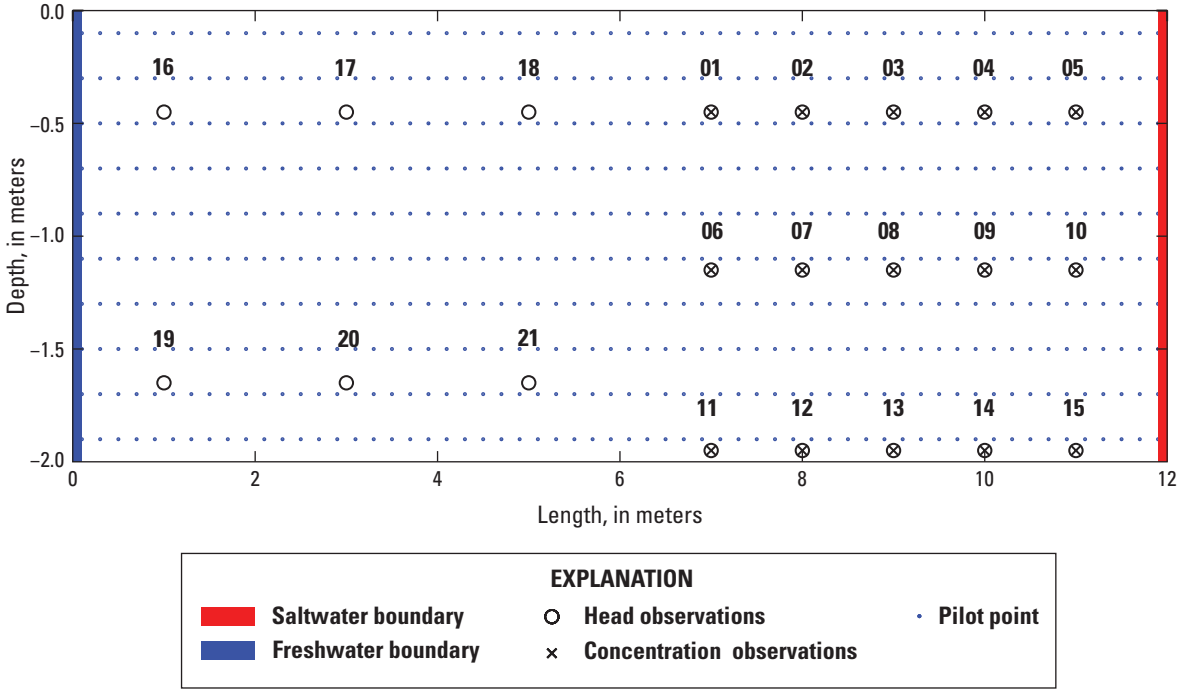


Figure 5–1. SEAWAT model domain.

Results

For the synthetic SEAWAT model, the PEST++ integrated linear analysis results compare well with the same analysis completed by using PEST utilities. An excerpt from the PEST++ record file (fig. 5–2) displays the prior and posterior variance of select parameters and predictions. The output of the PREDUNC1 legacy software that was used to verify the integrated linear analyses, when applied to forecast pred_one, is shown in figure 5–3.

The cumulative element-wise sum of the differences between the PEST++ and PREDUNC7 posterior parameter matrices is 1.6E−07. Table 5–1 compares the posterior variances of selected parameters. The prior and posterior forecast uncertainty estimates agree well for forecasts pred_one, pred_ten, and pred_half (table 5–2). Note that PREDUNC1 reports standard deviation as the measure of uncertainty, whereas the PEST++ linear analyses report variance.

```
---------------------------------------
---- parameter uncertainty summary ----
---------------------------------------
    name  prior_mean  prior_variance   post_mean  post_variance
   MULT1     1.00000      0.00307605     1.00000     0.00175068
 KR01C01     200.000        0.250000     200.000       0.250000
 KR10C35     200.000        0.250000     200.000       0.249919
 KR10C38     200.000        0.250000     200.000       0.249774
 KR10C44     200.000        0.250000     200.000       0.249654
---------------------------------------
---- prediction uncertainty summary ----
---------------------------------------
     name  prior_mean  prior_variance  post_mean  post_variance
  PD_HALF     9.39664       0.0800481    9.39664      0.0600698
   PD_ONE     8.77066       0.0449995    8.77066
```

Figure 5–2. Excerpt from PEST++ record file.

```
PREDUNC1 Version 13.3. Watermark Numerical Computing.

 Enter name of PEST control file:  Enter observation reference variance:
 Enter name of parameter uncertainty file:  Enter name of predictive sen-
sitivity matrix file:
 Use which version of linear predictive uncertainty equation:-
     if version optimized for small number of parameters   - enter 1
     if version optimized for small number of observations - enter 2
 Enter your choice:
 - reading PEST control file pest_ord.pst....
 - file pest_ord.pst read ok.

 - reading Jacobian matrix file pest_ord.jco....
 - file pest_ord.jco read ok.

 - reading predictive sensitivity matrix file pd_one.vec....
 - file pd_one.vec read ok.

 - reading parameter uncertainty file pest.unc....
 - reading covariance matrix file cov.mat...
 - covariance matrix file cov.mat read ok.
 - parameter uncertainty file pest.unc read ok.

 - computing pre-calibration predictive uncertainty....
 - forming XtC-1(e)X matrix....
 - inverting C(p) matrix....
 - inverting [XtC-1(e)X + C-1(p)] matrix....
 - calculating post-calibration predictive uncertainty....
```

Figure 5–3. Output of PREDUNC1 utility applied to prediction pred_one. Note that PREDUNC1 reports standard deviation as the measure of uncertainty, whereas the PEST++ linear analyses report variance.

```
************************************************************
*                                                          *
*                                                          *
*    Pre-cal predictive uncertainty  =  0.2121308          *
*    Post-cal predictive uncertainty =  0.1777507          *
*                                                          *
*                                                          *
************************************************************
```

Figure 5–3.—Continued Output of PREDUNC1 utility applied to prediction pred_one. Note that PREDUNC1 reports standard deviation as the measure of uncertainty, whereas the PEST++ linear analyses report variance.

Table 5–1. Comparison of posterior parameter variances.

Parameter	PEST++	PEST
mult1	0.001751	0.001751
kr01c35	0.249982	0.249982
kr01c38	0.249894	0.249894
kr01c44	0.249822	0.249822

Table 5–2. Comparison of prior and posterior forecast standard deviations.

Forecast	PEST++		PEST	
	Prior	Posterior	Prior	Posterior
pd_half	0.282928	0.245091	0.282928	0.245091
pd_ten	0.289607	0.24578	0.289607	0.24578
pd_one	0.212131	0.177751	0.212131	0.177751

References

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

Henry, H.R., 1964, Effects of dispersion on salt encroachment in coastal aquifers: U.S. Geological Survey Water-Supply Paper, 1613–C, p. C71–C84.

Herckenrath, Daan; Langevin, C.D.; and Doherty, John, 2011, Predictive uncertainty analysis of a saltwater intrusion model using null-space Monte Carlo: Water Resources Research, v. 47, no. 5, W05504, 16 p.

Langevin, C.D., Thorne, D.T., Jr., Dausman, A.M., Sukop, M.C., and Guo, W., 2008, SEAWAT Version 4—A computer program for simulation of multi-species solute and heat transport: U.S. Geological Survey Techniques and Methods, book 6, chap. A22, 39 p.

Appendix 6. GSA++ Implementation and Use

GSA++ shares a common command line with PEST++ as well as the input control, template files, and instruction file. Algorithmic variables that control the behavior of GSA++ are stored in a text file with a *.gsa* suffix. For example, control variables specific to the Method of Morris must be specified in a file that has the same base name the PEST control file, but with a *.gsa* extension. The variables in this file are shown in figure 6–1.

```
METHOD(MORRIS)

MORRIS_R(4)

MORRIS_P(4)

MORRIS_DELTA(.666)

MORRIS_POOLED_OBS(FALSE)
```

Figure 6–1. Example GSA++ input file for Method of Morris analysis.

General GSA++ Options

Variable	Type	Values	Description
METHOD	Text	"MORRIS" or "SOBOL"	Specifies type of analysis to be performed.
RAND_SEED	Unsigned integer		Seed for the random number generator.

GSA++ Options Specific to Method of Morris

Variable	Type	Values	Description
MORRIS_R	Integer	positive integer	Sample size. The number of times the sensitivity will be computed for each parameter.
MORRIS_P	Integer	positive integer	Number of levels or the number of points at which each parameter is sampled.
MORRIS_DELTA	Real	multiple of $\frac{p}{[2(p-1)]}$ where p=MORRIS_P	Size of the sampling step. This must be a multiple of $p/[2(p-1)]$ and represent the size of the interval that will be used to calculate the sensitivities.
MORRIS_POOLED_OBS	Text	"TRUE" or "FALSE"; default is "FALSE"	
MORRIS_OBS_SEN	Text	TRUE" or "FALSE"; default is "TRUE"	A value of "TRUE" instructs GSA++ to perform the Method of Morris sensitivity for each observation.

GSA++ Options Specific to the Method of Sobol

Variable	Type	Values	Description
SOBOL_SAMPLES	Long integer	positive integer	Size of the samples to be used in Sobol's method when computing sample variances. This is "n" in the equation $s^2=\Sigma\ (x_i-x)^2/(n-1)$.
SOBOL_PAR_DIST	String	"NORM" "UNIF"	Specifies whether the parameter samples should be drawn from a uniform or normal distribution. If the parameters are assumed to be uniformly distributed use "UNIF"; otherwise, if the parameters are normally distributed, use "NORM".

GSA++ Output Files

The GSA++ implementation produces two output files summarizing the global sensitivity analysis. The Morris sensitivity file (*.msn*) is the primary output file which contains the metric associated with the Method of Morris analysis. The file contains a header line describing the information stored in the file, which consists of parameter_name, sen_mean(μ), sen_mean_abs(μ*), and sen_std_dev(σ). Each subsequent line contains the metrics for one of the adjustable parameters.

In addition to the *.msn* file, a raw sensitivity file (*.raw*) is also written which summarizes the raw model output that was used to compute the information stored in the *.msn* file. Each line stores a single sensitivity computed from a pair of model runs where phi_0, phi_1 are the values of the objective function used to compute the sensitivity; par_0, par_1 are the values of the adjustable parameter used to compute the sensitivity; and sen is the sensitivity.

```
parameter_name, sen_mean, sen_mean_abs, sen_std_dev
X1, -16.4665, 108.885, 138.542
X2, 53.5115, 72.4633, 98.2834
```

Figure 6–2. Example Morris sensitivity (*.msn*) file.

```
parameter_name, phi_0, phi_1, par_0, par_1, sen
X1,  128.437,  84.7042,  0.999999,  0.333333, 65.5993
X2,  114.144,  128.437,  0.666666,  0, -21.4395
```

Figure 6–3. Example raw sensitivity (*.raw*) file.

Appendix 7. Example Problem Using GSA++ and the Method of Morris

This appendix demonstrates the application GSA++ to perform the Method of Morris analysis using a benchmark analytical equation with known results. The equation was originally published by Morris (1991) and was also used by Saltelli and others (2004). The equation has 20 input parameters (x_1,……, x_{20}), each of which were treated as adjustable parameters in the GSA++ analysis. The equation is of the form:

$$y=\beta_0+\sum_{i=1}^{20}\beta_0 w_i+\sum_{j=1}^{20}\sum_{i=1}^{20}\beta_{i,j}w_i w_i+\sum_{l=1}^{20}\sum_{j=1}^{20}\sum_{i=1}^{20}\beta_{i,j,l}w_i w_j w_l+\sum_{s=1}^{20}\sum_{l=1}^{20}\sum_{j=1}^{20}\sum_{i=1}^{20}\beta_{i,j,l,s}w_i w_j w_l w_s \quad (1)$$

where

$$\begin{aligned}
w_i &= 2\left(x_i-\frac{1}{2}\right) && for && i\neq 3,5,7\\
w_i &= \frac{1.1x_i}{x_i+0.1} && for && i=3,5,7\\
\beta_0 &= 20 && for && 1\leq i\leq 10\\
\beta_{i,j} &= -15 && for && 1\leq i\leq 6,\ 1\leq j\leq 6\\
\beta_{i,j,l} &= -10 && for && 1\leq i\leq 5,\ 1\leq j\leq 5,\ 1\leq l\leq 5\\
\beta_{i,j,l,s} &= 5 && for && 1\leq i\leq 4,\ 1\leq j\leq 4,\ 1\leq l\leq 4
\end{aligned}$$

The remaining β_0 and $\beta_{i,j}$ terms are generated from a normal distribution with zero mean and unit standard deviation, and the remaining $\beta_{i,j,l}$ and $\beta_{i,j,l,s}$ term are set to zero. This simple example tests the ability of the Method of Morris to distinguish between the parameters that have negligible effects, linear (or additive) effects, and nonlinear effects. The published results of applying the Method of Morris to equation 1 above indicate (1) the first 10 parameters are important, (2) 7 of these have significant effects that involve interactions and (or) curvatures and, (3) and 3 of these are important primarily because of first-order effects (Saltelli and others, 2004).

The GSA++ *.pst* and *.gsa* control files are shown in figures 7–1 and 7–2. The output from the Method of Morris implementation available in GSA++ to the benchmark semi-analytical test problem is presented in figure 7–3 and plotted in figure 7–4. To remain consistent with the published results, the sensitivities of observation "Y" are used in the analysis rather than those of the objective function, which contain the sensitivities of Y2. Because the Method of Morris is typically used to make qualitative inferences about the relative importance of parameters rather than quantitative measurements, Saltelli and others (2004) did not publish numbers for the elementary effects but instead published the results in a graph, which corresponds well to the GSA++ result plotted in figure 7–4.

References

Morris, M.D., 1991, Factorial sampling plans for preliminary computational experiments: Technometrics, v. 33, no. 2, p. 161–174.

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.

```
pcf
* control data
restart  estimation
    20    1     1     0     1
    1     1 single point 1 0 0
  0.0   2.0   0.3  0.03    1
  3.0   3.0 0.001
  0.1
    0 0.01     3     3  0.01     3
    1     1     1
* parameter groups
sen relative 0.01  0.0  ALWAYS_3  2.0 parabolic
* parameter data
x1 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x2 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x3 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x4 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x5 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x6 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x7 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x8 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x9 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x10 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x11 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x12 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x13 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x14 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x15 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x16 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x17 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x18 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x19 none factor   .5     0.0   1.0    sen  1.0   0.0  1
x20 none factor   .5     0.0   1.0    sen  1.0   0.0  1
* observation groups
obsgroup
* observation data
y     0  1  obsgroup1
* model command line
exe\morris_1991.exe
* model input/output
misc\input.tpl   morris_1991.inp
misc\output.ins  morris_1991.out
```

Figure 7–1. GSA++ *.pst* control file.

```
METHOD(MORRIS)
MORRIS_R(4)
MORRIS_P(4)
MORRIS_DELTA(.666666)
MORRIS_POOLED_OBS(FALSE)
MORRIS_OBS_SEN(TRUE)
```

Figure 7–2. GSA++ *.gsa* control file.

```
.
.
.
observation "y"
parameter_name, sen_mean, sen_mean_abs, sen_std_dev
X1, -3.0754, 56.3122, 70.2853
X2, -4.44507, 45.5431, 58.0327
X3, 0.890216, 33.8028, 41.5672
X4, -6.67899, 39.4061, 47.5703
X5, 21.2355, 40.5433, 60.3545
X6, -16.9367, 40.0154, 59.769
X7, 9.64928, 29.699, 36.5273
X8, -0.917067, 36.9443, 39.841
X9, 15.5057, 37.1315, 35.8605
X10, 25.72, 41.7879, 35.5455
X11, -0.475847, 6.03301, 7.87455
X12, 1.95161, 5.05447, 6.26432
X13, -2.33225, 5.10258, 6.11983
X14, -0.867056, 2.82701, 3.42373
X15, 0.553902, 3.52677, 4.25734
X16, 4.08245, 6.55249, 7.10021
X17, 1.38838, 6.80866, 9.08623
X18, -2.30373, 9.77486, 11.3164
X19, -1.00296, 5.02641, 5.73822
X20, 1.83217, 5.13305, 6.06983
```

Figure 7–3. GSA++ *.msn* output file.

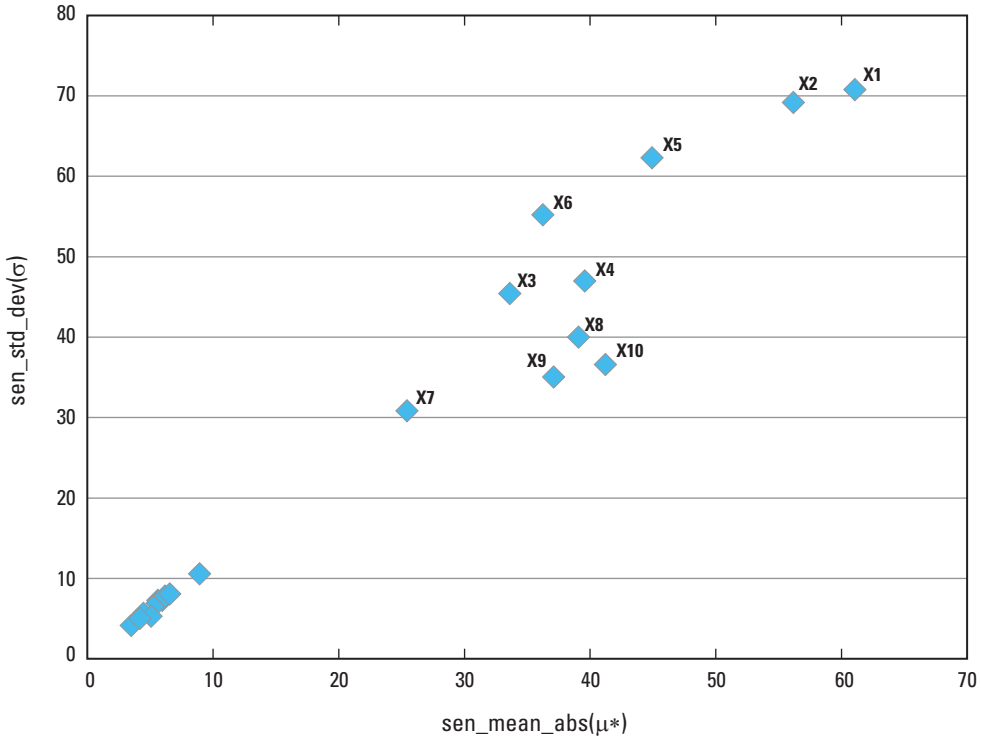


Figure 7–4. Plot of the Method of Morris results.

Appendix 8. Example Problem Using GSA++ and the Method of Sobol

This appendix summarizes application of the Method of Sobol in GSA++ to a benchmark analytical solution, which is based on the Ishigami function (Ishigami and Homma, 1990):

$$Y=g(X_1, X_2, X_3)=\sin X_1+a\sin^2 X_2+bX_3^4\sin X_1 \quad (1)$$

where X_1, X_2, X_3 are independent random variables drawn from the uniform distribution [$-\pi$, π]. This function provides verification of the GSA++ implementation of the Method of Sobol. Because this function is strongly nonlinear and nonmonotonic and has an analytical solution for the decomposed variances, it is widely used as an example for uncertainty and sensitivity analysis methods (Baudin and Martinez, 2013). Unfortunately, these examples do not contain descriptions of actual physical problems that can be solved by using this equation, so the user is left to imagine a generic process "Y" that is a function of three variables "X1", "X2", and "X3". The GSA++ *.pst* and *.gsa* files used for this example are presented in figures 8–1 and 8–2. The associated *.sbl* output file generated by the GSA++ contains the Sobol sensitivities for the weighted composite objective function, phi, followed by the Sobol sensitivities for each parameter defined in the *.pst* control file (fig. 8–3). The GSA++ computed results compare well to published results (table 8–1).

```
pcf
* control data
restart  estimation
    3    1     1     0     1
    1     1 single point 1 0 0
  0.0   2.0   0.3  0.03    1
  3.0   3.0 0.001
  0.1
    0 0.01     3     3  0.01     3
    1     1     1
* parameter groups
sen relative 0.01  0.0  ALWAYS_3  2.0 parabolic
* parameter data
x1 none factor   .5     -3.14159265359   3.14159265359    sen  1.0
0.0  1
x2 none factor   .5     -3.14159265359   3.14159265359    sen  1.0
0.0  1
x3 none factor   .5     -3.14159265359   3.14159265359    sen  1.0
0.0  1
* observation groups
obsgroup
* observation data
y     0  1  obsgroup1
* model command line
exe\ishigami.exe
* model input/output
misc\input.tpl   input.dat
misc\output.ins  output.dat
```

Figure 8–1. PEST++ *.pst* control file.

```
METHOD(SOBOL)
SOBOL_SAMPLES(50000)
SOBOL_PAR_DIST(UNIF)
RAND_SEED(1)
```

Figure 8–2. GSA++ *.gsa* control file.

```
Sobol Sensitivity for PHI
E(Y) = 26.1461;  Var(Y) = 1130.54 (for S_i calculations)
E(Y) = 26.0038;  Var(Y) = 1136.29 (for S_ti calculations)
parameter_name, s_i, st_i, n_runs
X1, 0.208987, 0.580258, 50000
X2, 0.281347, 0.420693, 50000
X3, 0.140049, 0.425424, 50000


Sobol Sensitivity for observation "Y"
E(Y) = 3.50619;  Var(Y) = 13.711 (for S_i calculations)
E(Y) = 3.49479;  Var(Y) = 13.7508 (for S_ti calculations)
parameter_name, s_i, st_i, n_runs
X1, 0.313813, 0.559639, 50000
X2, 0.443826, 0.439341, 50000
X3, -0.00343356, 0.243083, 50000
```

Figure 8–3. GSA++ *.sbl* output file for Method of Sobol.

Table 8–1. Comparision of GSA++ and analytical results.

Sobol sensitivity	GSA++ computed value	Analytical value
S_1	0.313813	0.313905
S_2	0.443826	0.442411
S_3	−0.00343356	0.000000
S_{T1}	0.559639	0.557589
S_{T2}	0.439341	0.442411
S_{T3}	0.243083	0.243684

References

Baudin, Michael, and Martinez, J.-M., 2013, Introduction to sensitivity analysis with NISP, Version 0.4, *in* Scilab users manual: 73 p., available at http://www.scilab.org.

Ishigami, T., and Homma, T., 1990, An importance quantification technique in uncertainty analysis for computer models, *in* Uncertainty modeling and analysis—Proceedings of the First International Symposium on Uncertainty Modeling and Analysis, College Park, Md., December 3–5, 1990: IEEE, p. 398–403.

ISSN 2328-7055 (online)
http://dx.doi.org/10.3133/tm7C12