Environmental Modelling & Software 85 (2016) 217–228

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A python framework for environmental model uncertainty analysis

Jeremy T. White [a, *], Michael N. Fienen [b], John E. Doherty [c]

[a] U.S. Geological Survey, Texas Water Science Center, 1505 Ferguson Lane, Austin, TX, USA
[b] U.S. Geological Survey, Wisconsin Water Science Center, 8505 Research Way, Middleton, WI, USA
[c] Watermark Numerical Computing, Brisbane, Australia

ARTICLE INFO

Article history:
Received 25 March 2016
Received in revised form
19 August 2016
Accepted 25 August 2016
Available online 3 September 2016

Keywords:
Uncertainty analysis
FOSM
Python
Model independent
Highly-parameterized
Data worth analysis

ABSTRACT

We have developed pyEMU, a python framework for Environmental Modeling Uncertainty analyses, open-source tool that is non-intrusive, easy-to-use, computationally efficient, and scalable to highly-parameterized inverse problems. The framework implements several types of linear (first-order, second-moment (FOSM)) and non-linear uncertainty analyses. The FOSM-based analyses can also be completed prior to parameter estimation to help inform important modeling decisions, such as parameterization and objective function formulation. Complete workflows for several types of FOSM-based and non-linear analyses are documented in example notebooks implemented using Jupyter that are available in the online pyEMU repository. Example workflows include basic parameter and forecast analyses, data worth analyses, and error-variance analyses, as well as usage of parameter ensemble generation and management capabilities. These workflows document the necessary steps and provides insights into the results, with the goal of educating users not only in how to apply pyEMU, but also in the underlying theory of applied uncertainty quantification.

Published by Elsevier Ltd.

1. Introduction

Stakeholders and other consumers of environmental analysis are increasingly advocating for the quantification of uncertainty in the analysis provided to them (e.g. Hester and Coleman, 2014; Uusitalo et al., 2015). For example, environmental models are increasingly being used to inform the decisions related to resource management, which in this context, requires that forecasts made with such models be subjected to uncertainty analysis. Recent administrations have explicitly called for uncertainty analysis in forecasts. They specifically stated *"A good analysis provides … the results of formal sensitivity and other uncertainty analyses."* (Office of Management and Budget, 2003, p. 3). Further, *"If … uncertainty is reduced and accurately described, then decisions will be made that tend to make for better use of the resource and increase public benefits and/or reduce risk."*(National Science and Technology Council, 2007, p. 12). Recent environmental modeling analyses that include some form uncertainty treatment include Zhang et al. (2016); Zheng and Han (2016); Clough et al. (2016); Cho et al. (2016) among others. Many of these analyses employe models that are amenable to rigorous uncertainty treatment: the forward execution time of model is relatively short (possibly due in part to access to high-performance or high-throughput computing) and/or the number of uncertain model inputs is relatively small. Indeed, to date, numerous uncertainty analysis frameworks have been proposed in the literature, see for example Wu and Liu (2012); Yen et al. (2014); Lu et al. (2014); Wang et al. (2016); most of these approaches employ sophisticated Bayesian analysis techniques, but, as a result of associated computational burden, are limited to models with short execution times and few uncertain model inputs to treat as parameters.

Unfortunately, application of uncertainty analysis techniques to complex, computationally-intensive environmental models (such as groundwater, surface water, and ecosystem models) is difficult because of the large numbers of model inputs that are uncertain, which can easily number in the thousands. This difficulty is compounded by the use of increasingly sophisticated and complex forward models that require increasingly long execution times. The result is large computational requirements, which, for many uncertainty analysis methods, yields an intractable problem. In this situation, computationally-tractable methods are needed (Hill et al., 2016). One class of techniques to evaluate model forecast uncertainty that scales efficiently to high dimensions is linear-based, first-order, second-moment (FOSM) uncertainty analyses,

* Corresponding author.
E-mail address: jwhite@usgs.gov (J.T. White).

http://dx.doi.org/10.1016/j.envsoft.2016.08.017
1364-8152/Published by Elsevier Ltd.

also known as Bayes linear theory (Goldstein and Wooff, 2007). While FOSM-based analyses have been proven successful in the environmental modeling literature (Glasgow et al., 2003; Moore and Doherty, 2005; Gallagher and Doherty, 2007; James et al., 2009; Dausman et al., 2010; Fienen et al., 2010; Lu et al., 2012; White et al., 2014; Leaf et al., 2015; Brakefield et al., 2015), existing software tools for practicing environmental modelers to easily apply these techniques are scant and the understanding required to use these tools can discourage potential users. Existing software tools that perform some form of FOSM-based analyses include PEST++ (Welter et al., 2015), the PREDUNC and PREDVAR software suites (Doherty, 2015), UCODE-2014 utility software (Poeter et al., 2014) and the software OPR-PPR (Tonkin et al., 2007), which was constructed using the building blocks within the joint universal parameter identification and evaluation of reliability application programming interface (JUPITER API) (Banta et al., 2006). Each of these tools provide FOSM as a statically-compiled executable, which is not favorable for exploratory analyses as the executable must be called repeatedly with different inputs and the outputs for each call must be stored/tracked by the user. Additionally, each of these existing tools implement at most two or three calculations, such as parameter uncertainty estimation, forecast uncertainty estimation, or a form of data worth analysis. This places an additional burden on users interested in completing a full suite of FOSM analyses.

Another useful feature of FOSM-based analyses is that they are considered model-independent because these techniques for uncertainty quantification only require the sensitivity of model parameters to model outputs that correspond to observations and forecasts. Therefore, FOSM-based analyses can be applied to any computer model for which these derivatives can be estimated or calculated.

In an effort to remedy the lack of available uncertainty analysis tools that can be easily and efficiently applied to highly-parameterized inverse problems, we have developed a python framework for environmental modeling uncertainty analyses (pyEMU). The pyEMU framework builds on and is compatible with the PEST suite of tools (Doherty, 2015) and has been designed to efficiently implement many forms of FOSM-based analysis in a single framework while also focusing on improving the user experience. pyEMU also implements sophisticated parameter ensemble generation and management capabilities, including the null-space Monte Carlo analysis of Tonkin and Doherty (2009), which uses FOSM theory to pre-condition parameter realization to reduce the computational demand of Monte Carlo analysis.

One of the most attractive uses of FOSM-based analyses is for estimating parameter and forecast uncertainty prior to a computationally-expensive inversion effort, which is possible since linear analyses do not require specific observation values or estimated parameter values. Employing linear analyses prior to inversion allows practitioners to estimate the value of the inversion process to reducing forecast uncertainty, as well as to investigate the sources of uncertainty and the worth of existing and potential new observations. The results of these pre-inversion analyses can be used to guide parameterization design, objective function formulation, and help focus the collection of new data, which are all important elements for any environmental modeling analysis.

pyEMU was created with the goal of increasing the number of environmental modeling analyses that include uncertainty estimates so that model-based resource management decisions can be better informed. This also provides an exploratory computing environment for users to build a better understanding of uncertainty analysis concepts. The software was developed in the context of groundwater modeling, but the techniques are general and can be used with any numerical modeling of environmental systems or others, provided that the model can be driven through text files and that model results can be read from files without manual user intervention.

2. Theory

pyEMU implements several types of linear parameter and predictive uncertainty analyses, including:

- Schur's complement for conditional uncertainty propagation (Koch, 1988; Golub and Van Loan, 1996; Doherty, 2015),
- error variance analysis (Moore and Doherty, 2005; White et al., 2014; Doherty, 2015), including the calculation of parameter identifiability (Doherty and Hunt, 2009; Hill, 2010; Doherty, 2010b), and
- Null Space Monte Carlo (Tonkin and Doherty, 2009).

A brief description of the theory supporting linear-based uncertainty analyses is presented. The interested reader is referred to Doherty (2015) as well as references cited herein for a more complete and rigorous treatment of these concepts.

2.1. Schur's complement

Schur's complement for linear uncertainty analyses can be viewed as a form of Bayes equation under the assumptions of a linear model and multivariate Gaussian distributions to describe stochastic character of parameters, forecasts, and observation noise (Tarantola, 2005; Fienen et al., 2010; Doherty, 2015). Briefly, the posterior parameter covariance matrix, $\overline{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}$ estimated with Schur's complement is:

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

where $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ is the prior parameter covariance matrix, $\boldsymbol{\Sigma}_{\varepsilon}$ is the epistemic observation noise covariance matrix, and $\mathbf{J}$ is the Jacobian matrix of partial first derivatives of observations with respect to parameters. This equation highlights the behavior of the inversion process. The first term represents the parameter uncertainty prior to inversion, and the second term encapsulates the inversion process, through the Jacobian matrix and both parameter and observation covariance, as mapping of information from observations to parameters. Depending on which data are collected and their level of certainty, the inversion process should result in a decrease in parameter uncertainty.

Note Eq. (1) expects the user to codify their understanding of the uncertainty in the model parameters – the uncertainty that exists in the parameter before the model is used – in the prior parameter covariance matrix ($\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$), which is a component of a Bayesian prior description of the parameters. The process of specifying a prior is a foundational part of Bayesian analysis. In this way, the Bayesian form of FOSM analysis is represented by Eq. (1). Note that the Bayesian formulation in pyEMU differs from the forms of FOSM analyses implemented in the OPR-PPR program (Tonkin et al., 2007), UCODE-2014 (Poeter et al., 2014) or the JUPITER API (Banta et al., 2006), which instead implement the inclusion of prior parameter information through the use of prior information equations, rather than through explicit specification of a prior parameter covariance matrix.

The Jacobian matrix is typically calculated using a finite-difference approximation

$$\mathbf{J}_{i,j} = \frac{\partial z_i}{\partial \theta_j} \approx \frac{m(\theta_j + \delta\theta) - m(\theta_j)}{\delta\theta} \quad (2)$$

where $\mathbf{J}_{i,j}$ is the Jacobian matrix entry for the i^{th} observation and the j^{th} parameter, $m(\cdot)$ is a model output—either collocated with an observation data point (z_i), or located at a potential observation point—θ is a parameter value, and $\delta\theta$ is a perturbation of the parameter value. Note that actual observation values (z_i) do not play a role in this calculation—only the modeled outputs. Similarly, optimal parameter values are not required.

Prior and posterior uncertainty estimates for forecast s—σ_s^2 and $\overline{\sigma}_s^2$—can be easily calculated by projecting the requisite parameter covariance matrix to the forecast output space using a forecast sensitivity vector:

$$\sigma_s^2 = \mathbf{y}^T \boldsymbol{\Sigma}_\theta \mathbf{y} \quad (3)$$

and

$$\overline{\sigma}_s^2 = \mathbf{y}^T \overline{\boldsymbol{\Sigma}}_\theta \mathbf{y} \quad (4)$$

where σ_s^2 and $\overline{\sigma}_s^2$ are the prior and posterior variances of forecast s and $\mathbf{y}$ is the vector of forecast sensitivity to each parameter, calculated as

$$\mathbf{y} = \frac{\partial s}{\partial \boldsymbol{\theta}} \quad (5)$$

As shown in Eqs. (1), (3) and (4), linear-based uncertainty analyses do not require the actual values for parameters or observations, only the sensitivity of observations with respect to parameters ($\mathbf{J}$ of Eq. (1)) and the sensitivity of forecasts with respect to parameters ($\mathbf{y}$ of equations (3) and (4)). In other words, the linear assumption underlying FOSM-based analyses implies that $\mathbf{J}$ of Eq. (1) and $\mathbf{y}$ of Eqs. (3) and (4) are approximately constant across parameter space. As a result of the linearity assumption, FOSM-based analyses can be applied before the costly inversion process to estimate the value of the inversion to reduce parameter and forecast uncertainty; FOSM-based analyses can also be used for efficient data worth analyses to evaluate which data are most important to reducing forecast uncertainty. For example, Fienen et al. (2010) used Schur's complement to design and optimize monitoring networks with respect to reducing forecast uncertainty, and Leaf et al. (2015) evaluated forecasts important to stakeholders to recommend where the most valuable new data could be collected.

2.2. Schur's complement for data worth analysis

The Bayesian calculations of posterior parameter and forecast uncertainty expressed in Eqs. (1) and (4) are conditional upon both the prior parameter uncertainty and the information contained in the observations used in the notional calibration implied by the $\mathbf{J}$ matrix and the second term of Eq. (1). Because of this conditional dependence, and because of the linearity assumption, FOSM-based uncertainty analyses can be employed as a computationally efficient means with which evaluate the value of information used in the calibration process. That information can take the form of obtaining increased prior knowledge about parameters, evaluating the worth of potential new observations, or evaluating the worth of existing observations. Fienen et al. (2010), Dausman et al. (2010), and Tonkin et al. (2007) among others discuss these approaches in more detail. We summarize the general workflow for these types of analyses in the remainder of this section. Instructions and examples of using pyEMU for these analyses are presented in the accompanying example notebooks implemented using Jupyter (Pérez and Granger, 2007; Jupyter, 2016).

Lowering the uncertainty of forecast-sensitive parameters in a model typically reduces the uncertainty of forecasts. Parameter uncertainty can be reduced by either directly collecting more information about parameters, or indirectly by collecting more/better observation information that informs parameter values as estimated by calibration. An advantage to FOSM-based analyses is that it approximates the expensive process of calibration and forecasting through closed-form analytical equations, allowing a variety of observation and parameter information strategies to be evaluated without the need to collect the information or make additional runs the computationally-expensive model.

2.2.1. Increased parameter knowledge

FOSM-based analyses can be used to quantitatively evaluate which parameters or groups of parameters are contributing most to prior and posterior forecast uncertainty. This assessment proceeds by systematically assigning each nominated parameter or group of parameters a variance of zero on the diagonal of the $\boldsymbol{\Sigma}_\theta$ matrix of Eqs. (1), (3) and (4). The remaining non-zero diagonal elements of $\boldsymbol{\Sigma}_\theta$ are subjected to conditioning through relevant off-diagonal covariates to form a new $\boldsymbol{\Sigma}_\theta$ matrix. The new $\boldsymbol{\Sigma}_\theta$ matrix is then used in equations (1), (3) and (4) to calculate prior and posterior forecast uncertainty resulting from perfect knowledge (zero variance) of the nominated parameter or group of parameters. The results of the parameter contribution analysis can be used to identify the dominant sources of forecast uncertainty and to guide future data collection or analyses.

2.2.2. Evaluating worth of potential new observations

As previously shown, the values of observations and parameters do not appear in Eq. (1), only the sensitivity of observations to parameters (equation (2)). This means that Eqs. (1) and (4) can be used to estimate the posterior forecast uncertainty resulting from potential observations that have not yet been collected. This powerful functionality of FOSM-based analyses can be used to guide observation collection strategies.

To apply this type of analysis with pyEMU, users should include additional observations when calculating the $\mathbf{J}$ matrix of Eq. (1); the additional observations should represent the model-simulated equivalent to potential new observations. Note that these additional observations should be assigned a large variance (practically implemented by assigning low observation weight) on the diagonal of $\boldsymbol{\Sigma}_\varepsilon$ matrix to represent their large uncertainty as the actual value of these observations is unknown. The worth of these additional observations can then be evaluated by systematically assigning the new observations or groups of observations more representative variances on the diagonal of $\boldsymbol{\Sigma}_\varepsilon$, then recalculating forecast posterior uncertainty with Eqs. (1) and (4). By lowering the associated variances, the influence of the yet-to-be-collected observations is included in the notional inversion implied by Eq. (1). This type of data worth analysis measures the value of potential observations as a decrease of the posterior forecast uncertainty as a result of gaining information.

Additionally, the approach to evaluating the worth of additional observations in pyEMU is also available as a method to sequentially evaluate the importance of added observations one-by-one. After a single calculation of the data worth for each potential new observation in a group is evaluated, the calculations can be repeated with the highest worth potential observation from the previous iteration incorporated into the observation dataset for the next iteration.

2.2.3. Evaluating worth of existing observations

Similar to the evaluation of potential new observations, FOSM-

based analysis can be used to evaluate the worth of existing observations. As an example, this type of data worth analysis can be used to optimize the cost associated with an existing monitoring network/schedule by identifying which observations are most important for reducing posterior forecast uncertainty. Observations that are shown to play a smaller role in reducing forecast uncertainty can be considered redundant and targeted for removal.

Using the $\boldsymbol{\Sigma}_\varepsilon$ matrix, observations or groups of observations can be systemically assigned large variances, then Eqs. (1) and (4) can be used to estimate the resulting posterior forecast uncertainty. This type of data worth analysis measures the value of the nominated observation or group of observations as a rise in posterior forecast uncertainty as a result of losing information.

2.3. Error-variance analysis

Error variance analysis is another form of uncertainty analysis that relies on a linear model assumption and a multivariate Gaussian assumption to describe prior and posterior parameter and forecast uncertainty. However, unlike Schur's complement, error variance analysis produces a potential forecast error function that is dependent on the number of parameter components used when singular value decomposition is used for the inversion (Moore and Doherty, 2006). Since the number of parameter components used for inversion controls the level of agreement between observations and model simulated equivalents, error variance analysis also provides the user with information regarding how the calibration process should be implemented to reduce forecast uncertainty to an optimal, or minimum error variance, level (Moore and Doherty, 2005).

The form of error variance analysis implemented in pyEMU includes the optional contribution from model error calculated through the use of "omitted" parameters (White et al., 2014). This form of error variance analysis can provide useful insights into the parameterization design process and the objective function formulation process in the presence of model error, and can be used to guide the inversion process to minimize the detrimental effects of model error.

Briefly, the error variance of a model forecast s, $\sigma^2_{s-\bar{s}}$ implemented in pyEMU is of the form

$$\sigma^2_{s-\bar{s}} = \mathbf{y}_i^T(\mathbf{I}-\mathbf{R})\boldsymbol{\Sigma}_{\theta_i}(\mathbf{I}-\mathbf{R})^T\mathbf{y}_i + \mathbf{y}_i^T\mathbf{G}\boldsymbol{\Sigma}_\varepsilon\mathbf{G}^T\mathbf{y}_i + \mathbf{p}\boldsymbol{\Sigma}_{\theta_o}\mathbf{p}^T \quad (6)$$

where $\mathbf{p}$ is

$$\mathbf{p} = \mathbf{y}_i^T\mathbf{V}_{i1}\mathbf{S}_{i1}^{-1}\mathbf{U}_{i1}^T\mathbf{J}_o - \mathbf{y}_o^T, \quad (7)$$

$\mathbf{y}_i$ and $\mathbf{y}_o$ are the forecast sensitivity vectors for the "included" and "omitted" parameters, $\mathbf{R}$ and $\mathbf{G}$ are the resolution matrix and parameter solution matrix of the "included" parameters, $\boldsymbol{\Sigma}_{\theta_i}$ and $\boldsymbol{\Sigma}_{\theta_o}$ are the "included" and "omitted" prior parameter covariance matrices, and $\mathbf{V}_{i1}$, $\mathbf{S}_{i1}$, and $\mathbf{U}_{i1}$ are the non-null singular components of the "included" normal matrix ($\mathbf{J}_i^T\mathbf{Q}\mathbf{J}_i$) and $\mathbf{J}_o$ is the "omitted" Jacobian matrix, $\mathbf{I}$ is the identity matrix and ^{T} denotes the transpose operator. In the subspace context of Eq. (6), $\mathbf{R} = \mathbf{V}_{i1}\mathbf{V}_{i1}^T$ and $\mathbf{G} = \mathbf{V}_{i1}\mathbf{S}_{i1}^{-1}\mathbf{U}_{i1}^T$.

It is important to note that Eqs. (6) and (7) are functions of the number of singular components used to form $\mathbf{R}$, $\mathbf{G}$, $\mathbf{V}_{i1}$,$\mathbf{S}_{i1}$, and $\mathbf{U}_{i1}$; therefore these equations are functions of how well the model outputs reproduce the observations since more parameter components is synonymous with improved fit. The first term on the right-hand side of Eq. (6) represents the potential forecast error arising from parameter uncertainty; the second represents the potential forecast error arising from measurement noise. As more parameter components are included in the notional calibration process implied by Eq. (6), the first term decreases monotonically, as more information in the observations is transferred to the parameters, while simultaneously, the second term rises monotonically as a result of the potential for "overfitting" increases as more parameter components are used.

The third term of Eq. (6) represents the potential forecast error arising from not adjusting all uncertain model inputs during calibration—that is, the forecast error penalty for fixing parameters. This penalty term includes the direct contribution from the uncertainty in the fixed parameters (second term on the right-hand side of Eq. (7)) as well as the contribution from adjustable parameters compensating for the fixed parameters (first term on the right-hand side of Eq. (7)). White et al. (2014) and Doherty (2015) provide more details on this topic.

The distinction between included and omitted parameters in Eqs. (6) and (7) is make by the user—both included and omitted parameters are model inputs. In this way, Eq. (6) allows users to explore how subjective parameterization choices made during the model construction process may effect the forecast reliability of the model.

2.4. Monte Carlo sampling techniques

An alternative to FOSM-based uncertainty analyses is Monte Carlo sampling in which model forecasts are evaluated for an ensemble of parameter values sampled stochastically (Tarantola (2005), chapter 2). To create this ensemble of forecast values, an ensemble of possible parameter values must be generated, typically by random sampling. The model is then run for each parameter set in the ensemble; for each run, the model simulated values of forecasts of interest are recorded, resulting in an ensemble of values for each forecast.

A constraint on the Monte Carlo process can be that model outputs generated by evaluating given parameter sets are generally consistent with an observation dataset. This constraint, usually enforced by a maximum acceptable calibration objective function value resulting from each parameter set, can require a large computational burden to identify a satisfactory number of parameter sets. The null-space Monte Carlo (NSMC) approach of (Tonkin and Doherty, 2009) uses FOSM theory to pre-condition the ensemble, yielding parameter sets that should be closer to an acceptable value of the calibration objective function. The key to NSMC is generating candidate parameter sets that explore parameters, or combinations of parameters that are uninformed by the observation dataset. In other words, NSMC explores the variability in the null space of the normal matrix. (Golub and Van Loan, 1996) provide additional background on the concept of a matrix null space; Doherty et al. (2011) and Doherty (2015) provide additional information on the theory of NSMC. For a comparison of NSMC with other, more rigorous techniques, see Keating et al. (2010).

pyEMU includes methods to draw ensembles of specified size and distribution and to pre-condition them using the NSMC methodology of Tonkin and Doherty (2009). pyEMU currently (2016) offers multivariate Gaussian and uniform statistical distributions to characterize parameter uncertainty. Sampling from a multivariate Gaussian distribution requires a pyEMU `Cov`-type matrix object; pyEMU offers several user-friendly means to instantiate (programmatically create a python variable of) a `Cov` object.

Several candidate parameter distributions are also available to draw random samples available with pyEMU and include the posterior Schur complement-derived covariance, a variogram-derived covariance, or others defined in various formats such as parameter bounds (such as those listed in a PEST control file), a PEST-compatible uncertainty file, or others. Ensembles can then be

written out to ASCII comma-separated-value files or plotted. Additionally, pyEMU ensembles can be written to PEST-specific files such as a series of sequentially numbered PEST control files or compatible parameter definition files. These can be then run with PEST or PEST++ and are suitable for distribution on parallel computing platforms using, for example, HTCondor or other tools (e.g. Fienen and Hunt, 2015).

To account for observation noise (e.g. measurement error) in the uncertainty analysis process, pyEMU also allows users to generate observation noise ensembles. When random parameter sampling is conducted on the $\overline{\Sigma}_\theta$ of Eq. (1) (the posterior distribution of parameters found using Schur's complement) and epistemic observation noise is sampled as well, the technique is similar to the "conditional realizations" suggested by Kitanidis (1995).

3. Implementation

The python scripting language (van Rossum, 1995) was selected as the basis for the pyEMU linear analysis framework because of its relatively simple and clean syntax, which facilitates a lower barrier for entry into computer programming (Bakker, 2014) and the full suite of object-oriented capabilities. pyEMU makes use of many object-oriented design principals, including inheritance, encapsulation, and operator overloading, all of which combine to make an extensible, efficient, and easy-to-maintain code base. Besides standard python, pyEMU has three dependencies, NumPy (Oliphant, 2006), SciPy (Jones et al., 2001), and Pandas (McKinney, 2012), all of which are included in common scientific python distributions.

pyEMU was designed to be fully compatible with the PEST (Doherty, 2010a, b) and PEST++ (Welter et al., 2015) software suites; pyEMU has been designed to seamlessly read and write PEST and PEST++ output and input files. This compatibility affords the numerous environmental modeling analyses that employ PEST and PEST++ for inversion easy access to the pyEMU functionality. Note, however, that pyEMU also supports instantiation with generic `NumPy ndarray` and/or `Pandas DataFrame` objects, so it can be used independent of the PEST framework as well.

Several base and derived classes constitute the pyEMU package. Appendix A summarizes the object structure of pyEMU. Also, the following Application section demonstrates the basic usage of pyEMU in the context of a simple, synthetic groundwater model. Users are also directed to the suite of Jupyter notebooks included with the code that demonstrate and describe the usage of pyEMU in a fully annotated fashion.

The object structure of pyEMU facilitates a range of functionality that is currently either not available in existing software, or is only available as sequential application of stand-alone command line executables, such as those in the PEST software suite (Doherty, 2010b). By unifying the functionality of separate, stand-alone software into a single framework, users can now more easily examine and explore the uncertainty estimates for a given environmental model using a suite of easily-accessed tools without the need to run separate codes, which can disrupt workflows and increase the potential for user error. Furthermore, when combined with interactive Jupyter notebooks (Pérez and Granger, 2007), pyEMU can provide an interactive environment for simultaneously analyzing and documenting exploratory parameter and forecast uncertainty analysis workflows, which can improve reproducibility and transparency of the uncertainty analysis.

The computational requirements of FOSM-based analyses depend directly on the number of parameters and the number of observations because these two factors control the size of the matrices and vectors used throughout the analyses. The time required to estimate parameter and forecast uncertainty with pyEMU for a problem with 1000 parameters and 1000 observations is less than a few seconds on a typical laptop computer. Since the data worth evaluations require repeatedly solving the FOSM forecast uncertainty estimation problem, the computational time scales linearly with the number of parameters and observations that are tested as part of the data worth process. In the case of the iterative process of finding the next best observation, if hundreds or thousands of potential observations are being tested, the FOSM forecast uncertainty estimate must be calculated several thousand times, resulting in an hour or more of time to complete the calculations. Note pyEMU includes extensive logging capabilities that can echo to the screen to help users follow the work flow as well as estimate the computational time for a given analysis.

4. Example application

The utility of pyEMU to perform FOSM-based analyses is demonstrated with a MODFLOW-2005 (Harbaugh, 2005) groundwater model based on the synthetic model of Freyberg (1988). Briefly, the model has 2 stress periods, a regular grid with 1 layer, 20 columns, and 40 rows and includes recharge, river boundaries, well boundaries, and specified-water-level boundaries (Fig. 1). The first stress period is a steady-state stress period that serves as the "calibration" period, while the second, transient stress period is the "forecast" period, in which recharge is decreased by 50% and pumping is increased (Table 1). This synthetic model captures many aspects of a typical groundwater modeling analysis: some knowledge of aquifer parameters and observations of water levels are available and a model is constructed to forecast how the aquifer system will respond to future climate and/or water use. In this sense, this synthetic model provides many of the key elements used in an actual application of pyEMU.

Many model inputs that are typically not known with certainty were specified as parameters, including (with the number of parameters in each class of parameter indicated in parentheses):

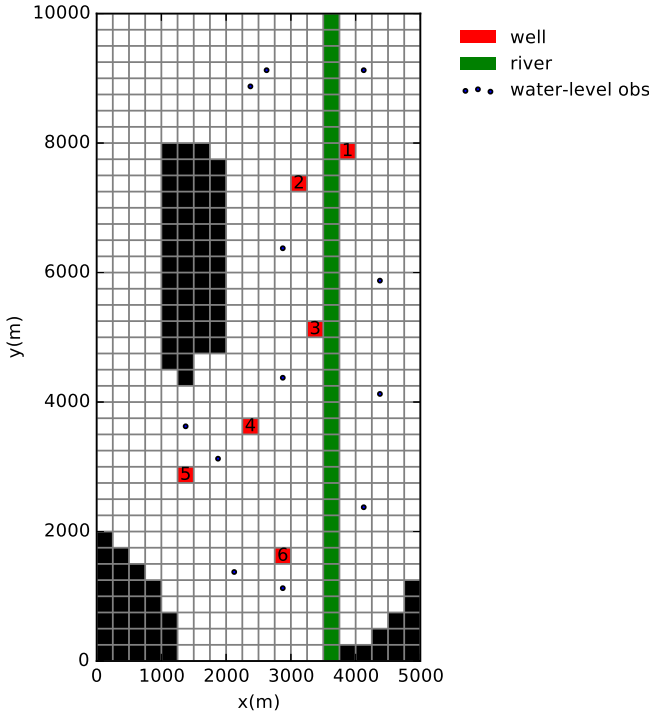

Fig. 1. Synthetic model domain after Freyberg (1988). Black areas are inactive flow areas within the model domain.

Table 1
Summary of prior parameter distribution.

Parameter	Transform	Mean	Standard deviation
hydraulic conductivity $\left(\frac{m}{d}\right)$	$\log_{10}$	0.7737	0.5
specific storage $\left(\frac{1}{d}\right)$	$\log_{10}$	−5	0.5
specific yield (dimensionless)	$\log_{10}$	−2	0.5
recharge multiplier-calibration $\left(\frac{m}{d}\right)$	$\log_{10}$	0.0	0.0215
recharge multiplier-forecast $\left(\frac{m}{d}\right)$	$\log_{10}$	0.0	0.0215
river cell conductance $\left(\frac{m^2}{d}\right)$	$\log_{10}$	3.69	0.5
pumping well 1 calibration $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.85	0.0215
pumping well 2 calibration $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.55	0.0215
pumping well 3 calibration $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.52	0.0215
pumping well 4 calibration $\left(\frac{m^3}{d}\right)$	$\log_{10}$	1.85	0.0215
pumping well 5 calibration $\left(\frac{m^3}{d}\right)$	$\log_{10}$	1.79	0.0215
pumping well 6 calibration $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.57	0.0215
pumping well 1 forecast $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.95	0.0215
pumping well 2 forecast $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.64	0.0215
pumping well 3 forecast $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.62	0.0215
pumping well 4 forecast $\left(\frac{m^3}{d}\right)$	$\log_{10}$	1.95	0.0215
pumping well 5 forecast $\left(\frac{m^3}{d}\right)$	$\log_{10}$	1.89	0.0215
pumping well 6 forecast $\left(\frac{m^3}{d}\right)$	$\log_{10}$	2.67	0.0215

- hydraulic conductivity of each active model cell (705)
- specific storage (1)
- specific yield (1)
- calibration period recharge multiplier (1)
- forecast period recharge multiplier (1)
- flux of each pumping well during the calibration period (6)
- flux of each pumping well during the forecast period (6)
- conductance of each river boundary cell (40)

For our example application, water level observations are recorded at 12 locations at the end of the first stress period (Fig. 1). Forecasts of interest include:

- surface water-groundwater exchange volume during the calibration period (`sw_gw_0`)
- surface water-groundwater exchange volume during the forecast period (`sw_gw_1`)
- water level at pumping well 5 at the end of the calibration period (`or28c05_0`)
- water level at pumping well 5 at the end of the forecast period (`or28c05_1`)

The name of the observation in the PEST++ dataset corresponding to each forecast is shown in parenthesis.

To apply the pyEMU framework to this example problem, the inverse problem was designed in the model-independent approach of PEST++. $\boldsymbol{\Sigma}_{\theta}$ and $\boldsymbol{\Sigma}_{\varepsilon}$ of Eqs. (1) and (6) were constructed "on the fly" by pyEMU from the parameter data and observation data sections of the PEST++ control file. To use this option, the upper and lower bounds of the parameters are assumed to represent values of $\theta+2\sigma$ and $\theta-2\sigma$ for the upper and lower bounds, respectively where σ is the standard deviation and θ is the base parameter value. Table 1 presents the prior distribution assigned to the parameters. Note that, if the parameters are subject to a log transformation, then the bounds are calculated as $10^{log_{10}\theta+2\sigma}$ and $10^{log_{10}\theta-2\sigma}$, respectively: The value of σ in such a case is the standard deviation in log-transformed space. An alternative formulation is to supply a parameter uncertainty file, following the PEST-compatible formats outlined by Doherty (2010a, b). For observations, the weights are assumed to be the inverse of the assumed standard deviation.

pyEMU was applied to the synthetic groundwater model before inversion to demonstrate the utility of pyEMU in a pre-inversion setting. Therefore, the forward model was only run 762 times to calculate perturbation sensitivities for each parameter, which were used to fill the requisite Jacobian matrix ($\mathbf{J}$ of Eqs. (1) and (6)). The parallel run manager of PEST++ (Welter et al., 2015) was used for this operation.

4.1. Schur's complement

Some basic input arguments for the use of Eq. (1) as implemented in pyEMU are now discussed. Note that the online code repository (see Code Availability section) for pyEMU includes Jupyter notebooks that provide more detailed descriptions for these arguments, as well as provide working examples of the analyses discussed herein.

Most modeling analyses are undertaken to make forecasts of unobserved system response. As such, forecast uncertainty is of

paramount importance. To that end, the following block of code demonstrates how to get FOSM-based forecast uncertainty estimates with pyEMU for this example

```
>>>import pyemu
>>>forecasts = ["sw_gw_0", "sw_gw_1", "or28c05_0", "or28c05_1"]
>>>la = pyemu.Schur(jco="freyberg.jco", forecasts=forecasts)
>>>forecast_sum = la.get_forecast_summary()

>>>forecast_sum.percent_reduction.plot(kind="bar")
```

This block of code, which is included in the Jupyter notebooks, was used to generate Fig. 2. Here, we will briefly describe the actions of each line:

1. `import pyemu` import the pyEMU module into the current namespace.
2. `forecasts = ["sw_gw_0", "sw_gw_1", "or28c05_0", "or28c05_1"]` define a python list that contains the names of observations in the PEST dataset that will be used as forecasts. These must be included in the PEST input files in the same way as calibration observations are included. In that sense, values must be assigned to them, but the values are arbitrary as only their sensitivities will be used in the calculations. Similarly, their weights are supplied to complete the PEST input but the values of the weights are not used.
3. `la = pyemu.Schur(jco="freyberg.jco", forecasts=forecasts)` instantiates a Schur-type object using an existing Jacobian matrix file and the previously named forecasts. pyEMU seeks and loads the accompanying PEST control file (freyberg.pst) and constructs $\mathbf{\Sigma}_{\theta}$ and $\mathbf{\Sigma}_{\varepsilon}$ of Eq. (1) from the parameter bounds and observation weights, respectively. pyEMU also loads the Jacobian matrix into memory and extracts the rows that correspond to the forecasts so that they can be used as y vectors in Eqs. (3) and (4).
4. `forecast_sum = la.get_forecast_summary()` gets a summary of the forecast prior and posterior uncertainty estimates. The method implicitly results in calculation of $\mathbf{\Sigma}_{\theta}$ of Eq. (1), which is subsequently used in Eq. (4). The method of the Schur object returns a Pandas DataFrame object with three columns: `prior, post, percent_reduction`, where `prior` and `post` are the prior and posterior variances of each forecast and `percent_reduction` is the reduction in uncertainty from prior to posterior.
5. `forecast_sum.percent_reduction.plot(kind="bar")` makes a bar chart of the percent_reduction column in the forecast summary dataframe, as illustrated in Fig. 2.

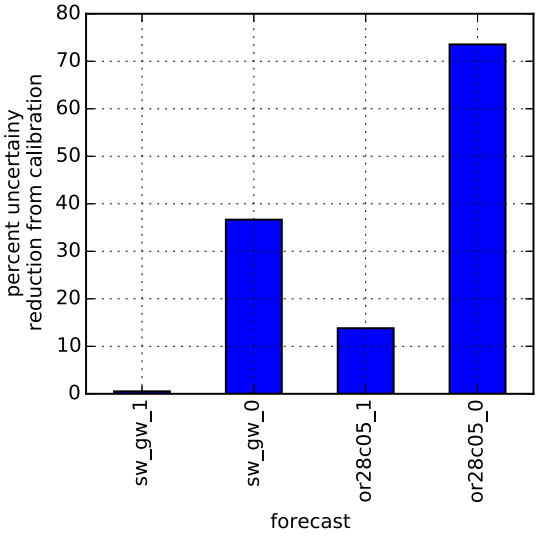

Fig. 2. FOSM-based uncertainty reduction estimates of the four different forecasts. The larger the bar, the more uncertainty is reduced for a given forecast as a results of inversion.

4.1.1. Parameter and forcing effects on forecast uncertainty

As discussed above in Section 2.2.1, we can use pyEMU to evaluate which model inputs (parameters or forcings) have the greatest effect on forecast uncertainty. As shown in Fig. 2, forecasts are not equally informed (using uncertainty reduction as a metric) by the calibration process. In Fig. 3, we break this down further to evaluate which groupings of parameters and forcings reduce forecast uncertainty the most. In the two panels on the left, uncertainty of future forcings (specifically well pumping and recharge) are assumed known as well as current stresses. Note that a 100% variance reduction in Fig. 3 indicates the maximum uncertainty reduction (shown in Fig. 2). In the right two panels, the prior uncertainty of forecast period forcings is assumed to be a higher, representing the lack of knowledge about future climate and future water use; the prior $\log^{10}$ standard deviation for forecast period recharge and pumping rates was increased from 0.0215 (Table 3) to 0.125. In each setup, the upper panel compares the behavior of aquifer properties (e.g. specific storage, river cell conductance, and hydraulic conductivity) with forcings in both the calibration period and the forecast period. The bottom panel in each case breaks these down into the various properties and forcings groups without differentiating between calibration and forecast periods. Recall that `or28c05_0` and `or28c05_1` are water-level forecasts near pumping well 5 for the calibration and forecast times, respectively. Similarly, the forecasts `sw_gw_0` and `sw_gw_1` predict the volumetric exchange between the river and the groundwater system at the calibration and forecast times, respectively.

Examining the top panels in Fig. 3, it is clear that the aquifer properties are most important during the calibration period for all forecasts. Calibration period forcing is not that important for the water-level forecast although it is important for the mass-balance dominated `sw_gw_0` forecast as pumping and recharge directly effects how much water is exchanged. For `or28c05_1`, forecast period forcings start to play a slightly more important role while for `sw_gw_1` forecast period forcings are more important. In the right panel when future forcings are less certain, they play a much more important role for both the forecast-period water-level and exchange forecasts. In fact, if future forcing effects are sufficiently uncertain, it may be more important to focus on learning more about them rather than focusing on traditional calibration exercises using static properties (e.g. Anderson et al., 2015, p. 458). The bottom panels break these findings down further, showing the tradeoff between storage and hydraulic conductivity for transient versus steady-state forecasts. Furthermore, recharge accounts for the majority of forecast uncertainty reduction when future stresses are less certain.

4.1.2. Data worth analysis

As discussed above in Section 2.2.2, Schur's complement can be used to estimate the value of additional information will have on forecast uncertainty, based on the assumptions of FOSM. The various details and aspects of how this information—either directly on parameters, or from existing and potential new observations—affects observation network design are shown in a Jupyter notebook in the pyEMU code repository. Here we highlight the evaluation of potential new water-level observations. Fig. 4 shows a

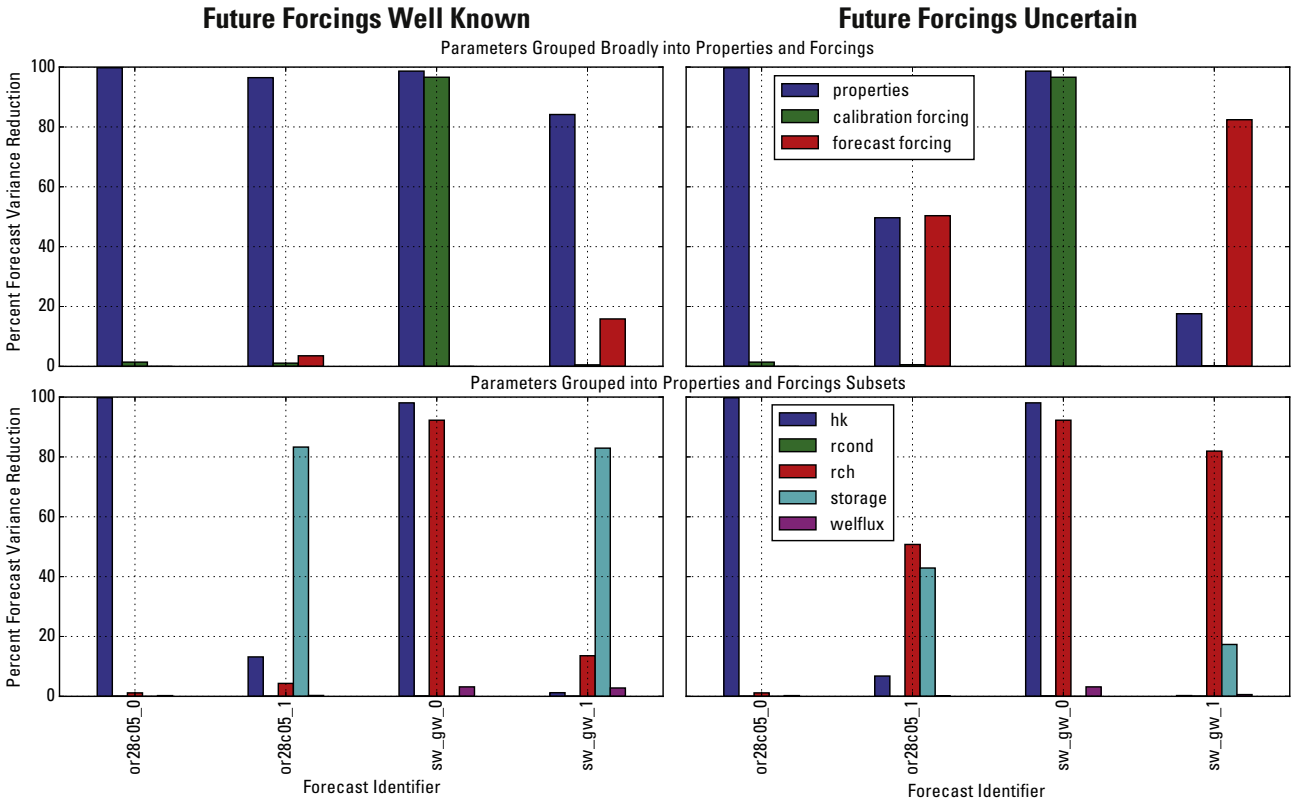

Fig. 3. The upper panels show reduction in forecast uncertainty (variance) grouped by static parameters (aquifer properties: hk, hydraulic conductivity; rcond, river cell conductance; storage, specific storage), calibration period forcings, and forecast period forcings. The lower panels show reduction in forecast uncertainty grouped by subsets of the static parameter groups (properties) and with forcings broken out between hk, rcond, recharge(rch), storage, and well pumping (wellflux) but not split into calibration and forecast periods. The pair of left panels was calculated with the assumption that future forcings have the same uncertainty as the calibration period forcings, whereas the right pair was calculated assuming future forcings are more uncertainty than calibration period forcings. Labels in the lower right panel explanation correspond to the parameter groups in the example problem.

color flood of potential new water-level observation worth to reduce forecast uncertainty overlain on the water-level contours and flow vectors for the calibration and forecast periods. The forecasts of interest are surface-water groundwater exchange volumes in the calibration period (`sw_gw_0`) and forecast period (`sw_gw_1`), respectively. Data worth is quantified as the estimated percent reduction in forecast uncertainty if a new observation is added:

$$\left(1 - \frac{\sigma^2_{+obs}}{\sigma^2_{base}}\right) \times 100. \tag{8}$$

where σ^2_{base} is the forecast uncertainty without new observations and σ^2_{+obs} is forecast uncertainty with the new observations. The result is the percent reduction. This is equivalent to the data worth metric of Fienen et al. (2010) with different symbology.

The calculated data worth is presented in color flood maps along with the MODFLOW water-level and flow velocity results in Fig. 4. The color floods in the left and right panels (calibration and forecast period, respectively) are similar and both reflect the importance of determining where the flow field is partitioned (e.g. the groundwater divide in the center-west portion of the field). This is the classic tradeoff among competing sinks and similar results were observed by Fienen et al. (2010). Note the differences in scale for the data worth. Collecting water-level data in the calibration period conveys much less information for the forecast period than for the calibration period. In fact, if the forcings are considered less certain, the data worth for the forecast period decreases even more (another order of magnitude in the case of this example application, not shown).

An additional capability of pyEMU is the ability to sequentially evaluate the *next* most valuable potential observation. This is done by assuming the most valuable potential observation was collected. The potential observation is incorporated into the notional calibration data set with weight similar to the existing calibration data. Then the potential for new locations is evaluated again and the most valuable potential observation is identified. This is repeated for as many iterations as specified by the user. For the two forecasts in Fig. 4 the top 10 sequentially most valuable points are shown as numbers in red triangles. The main divide remains important as the first most important observation location, but additional observations identified are either associated with boundaries or pumping stresses. This capability is important when designing a new observation scheme because exploring the color flood would cluster all potentially valuable new observations around the initially most valuable point (the divide in this case). The only way to re-evaluate the potential reduction in uncertainty of the *next* most important is to make that calculation condition on the assumed new assemblage of observation data.

4.2. Error variance analysis

As with the Schur's complement example application, we present the use of Eq. (6) as implemented in pyEMU. However, more

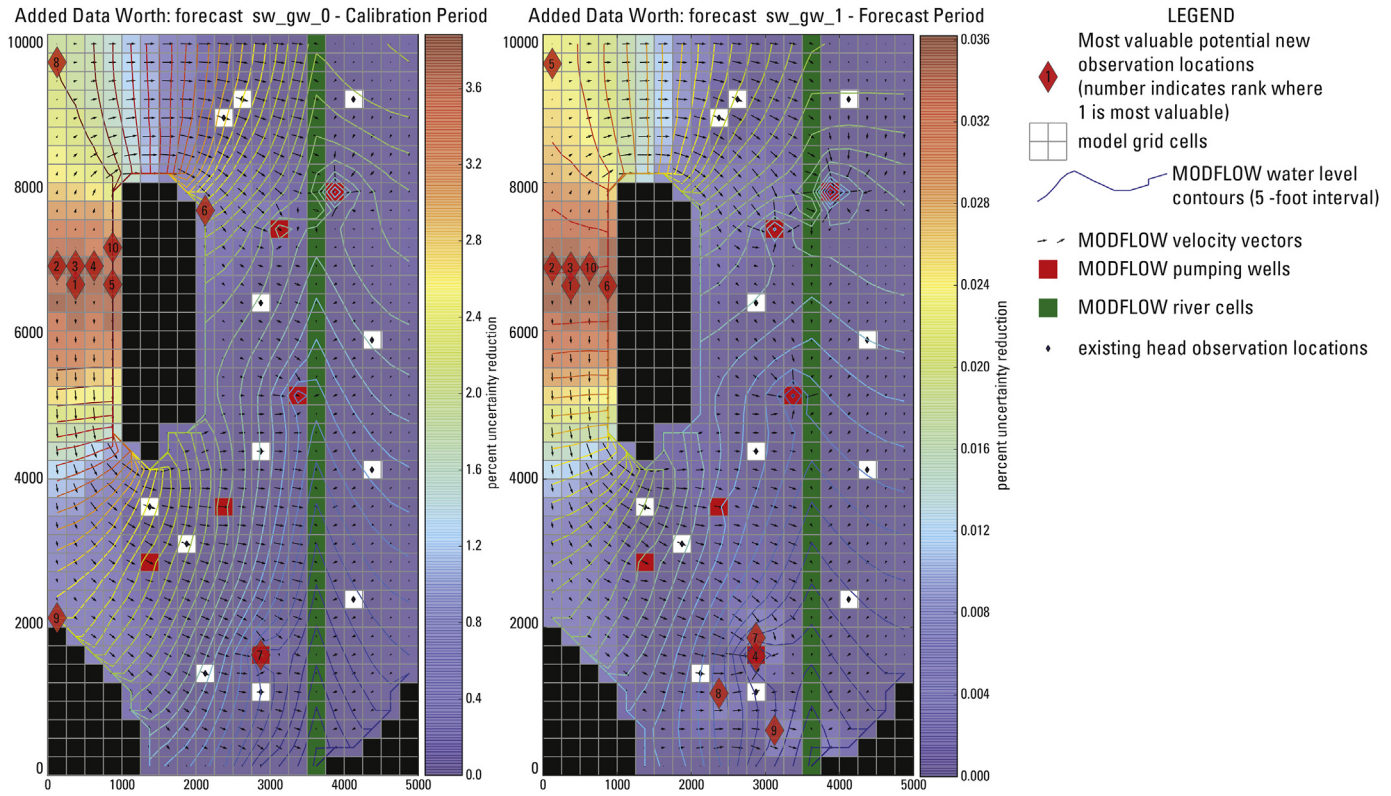

Fig. 4. Map of data worth for additional water-level observations in the calibration period for exchange forecasts `sw_gw_0` (calibration period, left panel) and `sw_gw_1` (forecast period, right panel).

detailed input instructions as well as a working example are provided in example notebooks implemented using Jupyter notebooks (Pérez and Granger, 2007) in the pyEMU code repository. The following code demonstrates how to apply pyEMU to Eq. (6):

```
>>>import pyemu
>>>forecasts = ["sw_gw_0", "sw_gw_1", "or28c05_0", "or28c05_1"]
>>>la = pyemu.ErrVar(jco="freyberg.jco", forecasts=forecasts)
>>>sing_vals = range(0,14)

>>>errvar_df = la.get_errvar_dataframe(sing_vals)
>>>errvar_df.to_latex("errvar.tex")
```

The first two lines of this code block are identical to the code block for Schur's complement. The remaining lines include:

3. `la = pyemu.ErrVar(jco="freyberg.jco", forecasts = =forecasts)` instantiates an error variance object using the binary Jacobian matrix file "freyberg.jco".
4. `sing_vals = range(0,14)` builds a list of integers from 0 to 14
5. `errvar_df = la.get_errvar_dataframe(sing_vals)` calculates the terms of Eq. (6). The method returns a multiindex Pandas DataFrame which include the contribution of each term of Eq. (6) to each of the identified forecasts.
6. `errvar_df.to_latex("errvar.tex")` writes the error variance results to a LaTeXtable. An Excerpt of this table is shown in Table 2; this table shows the contribution to total error variance of each forecast from the first term of Eq. (6). As a Pandas dataframe object, the results returned into `errvar_df` can be output to other formats using other methods as well including `to_csv`, `to_excel`, and others.

5. Verification against existing software

The pyEMU implementation of Eq. (1), Eqs. (3) and (4) and Eq. (6) were verified against results obtained from PREDUNC7, PREDUNC1, and PREDVAR1B, respectively (Doherty, 2010b). The software PREDUNC7, PREDUNC1, and PREDVAR1B are command-line driven, stand-alone executables that are available as part of the PEST software suite (Doherty, 2010a). The Jupyter notebook (Pérez and Granger, 2007) that documents and implements the

Table 2
Excerpt from table of results from pyEMU application to the first term of Eq. (6). The "first" is from the Pandas (McKinney, 2012) multiindexed dataframe that stores the error variance analysis results.

	First			
	or28c05_0	or28c05_1	sw_gw_0	sw_gw_1
0	0.303632	0.744522	2346.665821	93125.784926
1	0.286190	0.735479	2320.763341	93111.468642
2	0.112759	0.644095	2091.267000	93072.383387
3	0.112481	0.643973	1988.328320	93005.539544
4	0.112188	0.643828	1956.305907	92895.692286
5	0.111377	0.642628	1713.097707	92823.789675
6	0.104556	0.635848	1712.675425	92823.463404
7	0.104542	0.635847	1712.623036	92819.347027
8	0.104540	0.635847	1712.436907	92814.867720
9	0.104540	0.635847	1712.427939	92814.858862
10	0.104540	0.635847	1712.215515	92814.790399
11	0.104540	0.635847	1712.131989	92814.631335
12	0.104540	0.635847	1712.127659	92814.479982

Table 3
Comparison of PREDUNC1 and pyEMU results.

Forecast	Prior variance		Posterior variance	
	PREDUNC1	pyEMU	PREDUNC1	pyEMU
sw_gw_1	788.12	788.12	787.68170	787.68180
sw_gw_0	282.200	282.200	212.76030	212.76032
or28c05_1	1.2068	1.2068	1.16162	1.16162
or28c05_0	0.63326	0.63326	0.32388	0.32388

verification process presented herein is included in the pyEMU code repository.

The verification of Eq. (1) indicates the implementation in pyEMU is similar to the implementation in PREDUNC7; the cumulative difference, the relative percent difference, and cumulative absolute difference between the 761 × 761 matrices are −6.64E-6, −4.84e-6, 2.97e-5, respectively. These small differences can be attributed to differences in floating-point implementation between the two codes.

The pyEMU implementation of prior and posterior forecast uncertainty described in Eqs. (3) and (4) applied to the example problem described above in Section 4 yields results that are similar to the results obtained from PREDUNC1 (Table 3). Similarly, the pyEMU implementation of Eq. (6) also yields numerically identical results to those obtained from the application of PREDVAR1B.

Note that several other verification tests are available in the pyEMU code repository.

6. Software availability

pyEMU is available as an open-source code repository at https://github.com/jtwhite79/pyemu. Once downloaded (or cloned), pyEMU can be installed by executing `python setup.py install` within the pyEMU directory. Note that pyEMU requires `NumPy` version 1.7 or greater and `Pandas` version 0.17 or greater.

Several Jupyter notebooks (Pérez and Granger, 2007) are available in the repository that demonstrate the application of pyEMU to help design parameterizations, complete data worth analyses, and investigate parameter and forecast uncertainty analysis for environmental models. Included in the repository are the model input and output files used in the example application, the Jupyter notebooks (Pérez and Granger, 2007) that document the linear analyses applied to the example described herein as well as the workflow used to generate the verification results.

7. Conclusions

This paper documents the development of pyEMU, a python framework for environmental uncertainty analysis. pyEMU is model-independent and strives to be user friendly by accepting a wide range of mixed-type arguments. pyEMU is especially useful in a pre-inversion setting, where posterior parameter and forecast uncertainty can be estimated. Furthermore, pyEMU can also be used in a pre-inversion setting to investigate the source of forecast uncertainty and to evaluate the worth of existing and potential new data. This functionality can help guide the parameterization and objective function formulation process, which are important parts of the inversion process.

When combined with Jupyter notebooks (Pérez and Granger, 2007), pyEMU becomes a powerful platform for completing and documenting uncertainty analyses. This leads to more efficient, reproducible analyses that are less prone to transcription errors.

The utility of pyEMU was demonstrated with a synthetic groundwater model with hundreds of parameters. The importance of the level of uncertainty of future forcings was explored, along with evaluating the worth of potential new data or the removal of existing data. Considering the importance of more than one future observation in a suite of potential new observations is a new area of research. Multiple approaches are being explored and we present one in which the most valuable potential new observation is incorporated into the calibration dataset (assuming it is known rather than unknown) and the calculations are repeated to reveal the next most valuable observation location condition upon the newly updated set. This is repeated as many times as new locations are requested. This simulates the anticipated new knowledge obtained by adding to the calibration dataset.

Some of the basic interaction with pyEMU was provided in detail. However, more detailed and interactive examples of using pyEMU are available as Jupyter notebooks (Pérez and Granger, 2007), which are available in the pyEMU code repository.

Acknowledgments

The authors would like to acknowledge Andrew Leaf for his review of the manuscript, three anonymous reviewers at the journal, as well as Brian Clark for code review and suggestions for improving the code.

Any use of trade, product, or firm names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

Appendix A. pyEMU object structure summary

The object structure of pyEMU uses inheritance to minimize the size of the code base and provide a logical grouping between functional code elements. The object structure of pyEMU is briefly summarized below:

- `LinearAnalysis` base class for FOSM-based analyses (including `Schur`, `ErrVar`, `MonteCarlo` derived classes)
  - initializes and constructs base components
    - `.jco` – Jacobian `Matrix` instance
    - `.pst` – `Pst` instance to handle PEST control files
    - `.parcov` – parameter `Covariance` matrix instance
    - `.obscov` – observation noise `Covariance` matrix instance
    - `.forecasts` – python `dict` of 1D `Matrix` instances
  - ensures coordination and synchronization between components (`Matrix`, `Covariance`, `Pst`)
  - if needed, extracts row(s) from Jacobian `Matrix` and stores as individual forecast sensitivity vector(s) (1D Matrix)
  - logs progress to user nominated log file
  - exposes several common matrices as attributes
    - `LinearAnalysis.xtqx` – normal matrix
    - `LinearAnalysis.qhalfx` – "half" normal matrix
    - `LinearAnalysis.fehalf` – Karhuene-Loeve scaling matrix
  - `LinearAnalysis.prior_forecast` – access to prior forecast uncertainty estimate(s) (via Eq. (3)) as a python `dict` of forecast name, prior variance pairs
- `Schur` derived type
  - `Schur.posterior_parameter` – access to the posterior parameter covariance matrix (Eq. (1))
  - `Schur.posterior_forecast` – access to the posterior forecast variance (Eq. (4))
  - `Schur.get_parameter_summary()` – parameter uncertainty summary
  - `Schur.get_forecast_summary()` – forecast uncertainty summary

  - analyze which parameter(s) are contributing to forecast uncertainty
    - `Schur.get_contribution_dataframe()` – for individual parameter(s) or arbitrary groups of parameters
    - `Schur.get_contribution_dataframe_groups()` – use parameter groups from PEST control file
  - data worth analysis – which observations are most important for reducing posterior forecast uncertainty and which parameters are contributing most to forecast uncertainty
    - `Schur.get_added_obs_importance()` – How adding new, possible yet-to-be-collected observations reduces forecast uncertainty
    - `Schur.get_removed_obs_importance()` – How removing existing observations increases forecast uncertainty
    - `Schur.next_most_important_added_obs()` – iteratively evaluate the value of new observations to reduce forecast uncertainty. At the end of each iteration, the observation(s) that reduce forecast uncertainty the most are added to the base observations. This process identifies the order of the best new observations to collect
    - `Schur.get_par_contribution()` – How arbitrary groups of parameters contribute to forecast uncertainty
    - `Schur.next_most_par_contribution()` – iteratively evaluate which parameter(s) are contributing most to forecast uncertainty. At the end of each iteration, the largest contributing parameter(s) are marked as "known". This process identifies the order of the parameter(s) that are contributing most to forecast uncertainty.
- `ErrVar` derived type
  - accepts additional constructor arguments for omitted parameters ($\boldsymbol{\theta}_o$ of Eq. (6))
    - `omitted_parameters` – a python `list` of parameter names to treat as omitted. These parameters are extracted from the Jacobian, the prior parameter covariance matrix, and the forecast sensitivity vectors
  - implements Eqs. (6), and (7)
    - `ErrVar.get_errvar_dataframe()` – returns a multi-index Pandas dataframe with each forecast and of the terms of Eq. (6). If `omitted_parameters` was not set, only the first two terms of Eq. (6) are included.
  - implements calculation of identifiability of Doherty and Hunt (2009); Hill (2010).
    - `ErrVar.get_identifiability_dataframe()`
- `MonteCarlo` derived type
  - `MonteCarlo.draw()` – make draws from a multivariate normal distribution for parameters and optionally observation noise
  - `MonteCarlo.project_parensemble()` – implements the null-space Monte Carlo operation of Tonkin and Doherty (2009).
  - `MonteCarlo.get_null_proj()` – access to the null-space projection operator.
  - `MonteCarlo.write_psts()` – write new PEST control files using parameter ensemble and optional observation noise ensemble
- `Ensemble` base class for sets of parameters and observations
  - `ParameterEnsembleObservationEnsemble` derived types
  - inherits from Pandas DataFrame
    - includes sophisticated slicing, indexing, and grouping for statistical analysis
    - optimized IO routines
    - concise plotting methods
- `ParameterEnsemble` derived type
  - invisible handling of parameter log transformation
  - `ParameterEnsemble.from_parfiles()` – instantiation from PEST-style parameter files
  - `ParameterEnsemble.to_parfiles()` – write a sequence of PEST-style parameter files
  - `ParameterEnsemble.enforce()` – enforce parameter bounds
- `Matrix` base class for matrices (including `Covariance` derived class)
  - automatically aligned operations (based on row and column names) via operator overloading
  - invisible and efficient treatment of diagonal matrices
  - attribute-style access to common linear algebra operations
    - `Matrix.T` – transpose
    - `Martix.inv` – inverse
    - `Matrix.sqrt` – square root
    - `Matrix.U,Matrix.S,Matrix.V` – singular value decomposition components
  - `Matrix.from_binary()` and `Matrix.to_binary()` – reading and writing PEST-compatible binary files
  - `Matrix.from_ascii()` and `Matrix.to_ascii()` – reading and writing PEST-compatible ASCII files
  - `Covariance.from_uncfile()` and `Covariance.to_uncfile()` – reading and writing PEST-compatible uncertainty files
  - instantiation from PEST control file components
    - `Covariance.from_parbounds()` parameter covariance
    - `Covariance.from_obsweights` observation noise covariance
- `Pst` base class for PEST control files
  - efficient manipulation of control file components with Pandas DataFrames
  - `Pst.load()` and `Pst.write()` – reading and writing control files
  - `Pst.zero_order_tikhonov()` – add/update regularization
  - `Pst.parrep()` – update parameter values
  - multicomponent objective function formulation
    - `Pst.adjust_weights_resfile()` – using a PEST residuals file
    - `Pst.adjust_weights_recfile()` – using a PEST record file
    - `Pst.adjust_weights()` – using individual observations or arbitrary groups of observations

References

Anderson, M.P., Woessner, W.W., Hunt, R.J., 2015. Applied Groundwater Modeling Simulation of Flow and Advective Transport, second ed. Academic Press, London UK.

Bakker, M., 2014. Python scripting: the return to programming. Groundwater 52, 821–822.

Banta, E.R., Poeter, E.P., Doherty, J.E., Hill, M.C., 2006. Jupiter: Joint Universal Parameter Identification and Evaluation of Reliability-an Application Programming Interface (api) for Model Analysis. U.S. Geological Survey Techniques and Methods, 6–E1.

Brakefield, L., White, J., Houston, N., Thomas, J., 2015. Updated Numerical Model with Uncertainty Assessment of 1950-56 Drought Conditions on Brackish-water Movement within the Edwards Aquifer, San Antonio, Texas. U.S. Geological Survey Scientific Investigations Report 2015–5081.

Cho, E., Arhonditsis, G.B., Khim, J., Chung, S., Heo, T.-Y., 2016. Modeling metal-sediment interaction processes: parameter sensitivity assessment and uncertainty analysis. Environ. Model. Softw. 80, 159–174.

Clough, J., Polaczyk, A., Propato, M., 2016. Modeling the potential effects of sea-level rise on the coast of New York: integrating mechanistic accretion and stochastic uncertainty. Environ. Model. Softw. 84, 349–362.

Dausman, A., Doherty, J., Langevin, C., Sukop, M., 2010. Quantifying data worth toward reducing predictive uncertainty. Ground Water 48, 729–740.

Doherty, J., 2010a. PEST, Model-independent Parameter Estimation—User Manual.

with slight additions, fifth ed. Watermark Numerical Computing, Brisbane, Australia.
Doherty, J., 2010b. PEST, Model Independent Parameter Estimation. Addendum to User Manual, fifth ed. Watermark Numerical Computing, Brisbane, Australia.
Doherty, J., 2015. PEST and its Utility Support Software, Theory. Watermark Numerical Publishing.
Doherty, J., Hunt, R.J., 2009. Two statistics for evaluating parameter identifiability and error reduction. J. Hydrology 366, 119–127.
Doherty, J.E., Hunt, R.J., Tonkin, M.J., 2011. Approaches to Highly Parameterized Inversion: a Guide to Using PEST for Model-parameter and Predictive-uncertainty Analysis. Geological Survey Scientific Investigations Report 2010–5211.
Fienen, M., Doherty, J., Hunt, R., Reeves, H., 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 Investigation Report 2010-5159, p. 44.
Fienen, M.N., Hunt, R.J., 2015. High-throughput computing versus high-performance computing for groundwater applications. Groundwater 53, 180–184.
Freyberg, D.L., 1988. An exercise in ground-water model calibration and prediction. Ground Water 26, 350–360.
Gallagher, M., Doherty, J., 2007. Predictive error analysis for a water resource management model. J. Hydrology 334, 513–533.
Glasgow, H.S., Fortney, M.D., Lee, J., Graettinger, A.J., Reeves, H.W., 2003. Modflow 2000 head uncertainty, a first-order second moment method. Ground water 41, 342–350.
Goldstein, M., Wooff, D., 2007. Bayes Linear Statistics, Theory and Methods. John Wiley & Sons.
Golub, G., Van Loan, C., 1996. Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press.
Harbaugh, A.W., 2005. MODFLOW-2005, the U.S. Geological Survey Modular Ground-water Model – the Ground-water Flow Process, vol. 6.
Hester, C.M., Coleman, J., 2014. Between an uncomfortable decision and an absurd one. Groundwater 52, 645–646.
Hill, M.C., 2010. Comment on two statistics for evaluating parameter identifiability and error reduction by John Doherty and Randall J. Hunt. J. Hydrology 380, 481–488.
Hill, M.C., Kavetski, D., Clark, M., Ye, M., Arabi, M., Lu, D., Foglia, L., Mehl, S., 2016. Practical use of computationally frugal model analysis methods. Groundwater 54, 159–170.
James, S.C., Doherty, J.E., Eddebbarh, A.-A., 2009. Practical postcalibration uncertainty analysis: yucca Mountain, Nevada. Ground Water 47, 851–869.
Jones, E., Oliphant, T., Peterson, P., et al., 2001. SciPy: Open Source Scientific Tools for Python [Online; accessed 2016-08-10].
Jupyter, 2016. Jupyter Notebook Open Source Interactive Data Science Computing across over 40 Programming Languages (accessed: 08.07. 2016). http://jupyter.org/.
Keating, E.H., Doherty, J., Vrugt, J.A., Kang, Q., 2010. Optimization and uncertainty assessment of strongly nonlinear groundwater models with high parameter dimensionality. Water Resour. Res. 46.
Kitanidis, P.K., 1995. Quasi-linear geostatistical theory for inversing. Water Resour. Res. 31, 2411–2419.
Koch, K.-R., 1988. Parameter Estimation and Hypothesis Testing in Linear Models. Springer-Verlag New York, Inc, New York, NY, USA.
Leaf, A., Fienen, M., Hunt, R., Buchwald, C., 2015. Groundwater/surface-water Interactions in the Bad River Watershed, Wisconsin. U.S. Geological Survey Scientific Investigations Report 2015–5162, p. 110.
Lu, D., Ye, M., Hill, M.C., 2012. Analysis of regression confidence intervals and bayesian credible intervals for uncertainty quantification. Water Resour. Res. 48.
Lu, D., Ye, M., Hill, M.C., Poeter, E.P., Curtis, G.P., 2014. A computer program for uncertainty analysis integrating regression and Bayesian methods. Environ. Model. Softw. 60, 45–56.
McKinney, W., 2012. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.
Moore, C., Doherty, J., 2005. Role of the calibration process in reducing model predictive error. Water Resour. Res. 41, 1–14.
Moore, C., Doherty, J., 2006. The cost of uniqueness in groundwater model calibration. Adv. Water Resour. 29, 605–623.
National Science and Technology Council, 2007. A Strategy for Federal Science and Technology to Support Water Availability and Quality in the United States. Technical Report Subcommitee on Water Availability and Quality, September 4, 2007, Washington, D.C.
Office of Management and Budget, 2003. Circular A-4 (Technical Report United States Office of Management and Budget).
Oliphant, T.E., 2006. Guide to NumPy. Provo, UT.
Pérez, F., Granger, B.E., 2007. IPython: a system for interactive scientific computing. Comput. Sci. Eng. 9, 21–29.
Poeter, E., Hill, M., Lu, D., Tiedeman, C., Mehl, S., 2014. UCODE_2014, with New Capabilities ToDefine Parameters Unique to Predictions, Calculate Weights Using Simulated Values, Estimate Parameters with SVD, Evaluate Uncertainty with MCMC, and More. GWMI 2014-02. Integrated Groundw ater Modeling Center.
van Rossum, G., 1995. Python reference Manual. Technical Report CWI (Centre for Mathematics and Computer Science) Amsterdam, The Netherlands, The Netherlands.
Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. SIAM.
Tonkin, M., Doherty, J., 2009. Calibration-constrained monte carlo analysis of highly parameterized models using subspace techniques. Water Resour. Res. 45.
Tonkin, M., Tiedeman, C., Ely, D., Hill, M., 2007. OPR-PPR, a Computer Program for Assessing Data Importance to Model Predictions Using Linear Statistics. U.S. Geological Survey Techniques and Methods Report TM-6E2, p. 115.
Uusitalo, L., Lehikoinen, A., Helle, I., Myrberg, K., 2015. An overview of methods to evaluate uncertainty of deterministic models in decision support. Environ. Model. Softw. 63, 24–31.
Wang, C., Duan, Q., Tong, C.H., Di, Z., Gong, W., 2016. A GUI platform for uncertainty quantification of complex dynamical models. Environ. Model. Softw. 76, 1–12.
Welter, D., White, J., Doherty, J., Hunt, R., 2015. PEST++ Version 3, a Parameter Estimation and Uncertainty Analysis Software Suite Optimized for Large Environmental Models. U.S. Geological Survey Techniques and Methods Report, 7–C12, p. 54.
White, J.T., Doherty, J.E., Hughes, J.D., 2014. Quantifying the predictive consequences of model error with linear subspace analysis. Water Resour. Res. 50, 1152–1173.
Wu, Y., Liu, S., 2012. Automating calibration, sensitivity and uncertainty analysis of complex models using the R package flexible modeling environment (FME): SWAT as an example. Environ. Model. Softw. 31, 99–109.
Yen, H., Wang, X., Fontane, D.G., Harmel, R.D., Arabi, M., 2014. A framework for propagation of uncertainty contributed by parameterization, input data, model structure, and calibration/validation data in watershed modeling. Environ. Model. Softw. 54, 211–221.
Zhang, J.L., Li, Y.P., Huang, G.H., Wang, C.X., Cheng, G.H., 2016. Evaluation of uncertainties in input data and parameters of hydrological model using a bayesian framework: a case study of a snowmelt-precipitation-driven watershed. J. Hydrometeorol.
Zheng, Y., Han, F., 2016. Markov chain monte carlo (mcmc) uncertainty analysis for watershed water quality modeling and management. Stoch. Environ. Res. Risk Assess. 30, 293–308.