

Building model analysis applications with the Joint Universal Parameter Identification and Evaluation of Reliability (JUPITER) API[☆]

Edward R. Banta^a, Mary C. Hill^{b,*}, Eileen Poeter^c,
John E. Doherty^d, Justin Babendreier^c

^aUS Geological Survey (USGS), P.O. Box 25046, MS 415, Lakewood, CO 80225, USA

^bUSGS, 3215 Marine Street, Boulder, CO 80303, USA

^cColorado School of Mines and the International Ground Water Modeling Center, Golden, CO 80401, USA

^dWatermark Numerical Computing and University of Queensland, Australia

^eEcosystems Research Division, National Exposure Research Laboratory, US Environmental Protection Agency, 960 College Station Road, Athens, GA 30605, USA

Received 1 October 2006; received in revised form 12 March 2007; accepted 19 March 2007

Abstract

The open-source, public domain JUPITER (Joint Universal Parameter Identification and Evaluation of Reliability) API (Application Programming Interface) provides conventions and Fortran-90 modules to develop applications (computer programs) for analyzing process models. The input and output conventions allow application users to access various applications and the analysis methods they embody with a minimum of time and effort. Process models simulate, for example, physical, chemical, and (or) biological systems of interest using phenomenological, theoretical, or heuristic approaches. The types of model analyses supported by the JUPITER API include, but are not limited to, sensitivity analysis, data needs assessment, calibration, uncertainty analysis, model discrimination, and optimization. The advantages provided by the JUPITER API for users and programmers allow for rapid programming and testing of new ideas. Application-specific coding can be in languages other than the Fortran-90 of the API. This article briefly describes the capabilities and utility of the JUPITER API, lists existing applications, and uses UCODE_2005 as an example. Published by Elsevier Ltd.

Keywords: Sensitivity; Uncertainty; Calibration; Optimization; Model discrimination

0. Introduction

Building models that represent natural systems requires assimilating data into the modeling process. Components of effective assimilation include, for example, sensitivity analysis, data needs assessment, parameter estimation, uncertainty evaluation, and optimization. These issues have been addressed by researchers in many fields. Recent textbooks

[☆] Code available from server at <http://www.iamg.org/CGEditor/index.htm>.

*Corresponding author. Tel.: +1 303 541 3014; fax: +1 303 447 2505.

E-mail addresses: erbanta@usgs.gov (E.R. Banta), mchill@usgs.gov (M.C. Hill), epoeter@mines.edu (E. Poeter), johndoherty@ozemail.com.au (J.E. Doherty), babendreier.justin@epa.gov (J. Babendreier).

include, for example, Parker (1994), Menke (1984), Saltelli et al. (2000, 2004, 2007), Tarantola (2005), and Aster et al. (2005) from the geophysics community; Sun (1994), Ahlfeld and Mulligan (2000), and Hill and Tiedeman (2007) from the ground-water community; Burnham and Anderson (2002) from the biology community; and Cook and Weisberg (1982, 1999), Seber and Wild (1989), Dennis and Schnabel (1996), and Draper and Smith (1998) from the statistics community.

Despite the substantial investment in this field, many problems remain. The problems are difficult because natural systems are complex, available data do not fully characterize the systems, computers are only now becoming adequate to address many of the relevant issues, and societal demands are increasing on many systems of concern. Problem resolution is confounded because (1) researchers are still developing and evaluating methods and ideas and (2) new methods are not readily available to practitioners for evaluation. For example, parameter estimation may be accomplished with single- or multi-objective functions using parsimonious or highly parameterized models (see Hill and Tiedeman, 2007, and references cited therein), and, when multiple models of a system are considered, model weighting might be based on Akaike, Bayesian, Hannan-Quinn, or Kashyap information criteria (see, for example, Burnham and Anderson, 2002; Poeter and Anderson, 2005; Meyer et al., 2004; Poeter and Hill, 2007). To date, testing is insufficient to provide modelers or resource managers with clear guidance about the utility of these and other existing approaches, and new approaches continue to be created. A computer environment that provides capabilities for rapid development of applications (computer programs) with a common input/output structure is needed. Instead of waiting years or decades for ideas and theories to be compared in the complex circumstances of interest to resource managers, the process can be expedited using an appropriately designed programming environment. In this mode, unproductive ideas and theories are revealed more quickly, and productive ideas and theories are more quickly used to address our increasingly difficult societal demands for modeling of natural systems. The JUPITER API (Banta et al., 2006)¹ provides a programming environment with the needed characteristics.

Other open-source, public-domain resources available for application development in these fields

include the DAKOTA toolkit² and the COSU API (Babendreier, 2004). Commercial resources include the International Mathematical and Statistical Library (IMSL), among others. Applications can be built with any combination of the cited products and the 11 Fortran modules (which are programmed using structured programming concepts) that make up the JUPITER API.

Capabilities provided by the JUPITER API not otherwise available or available in a substantially less mature manner include the following:

1. Comprehensive methods for interacting with complex process models.
2. Parallel computing capabilities using a robust dispatcher-runner protocol.
3. An input design that is flexible, largely self-descriptive, and facilitates reuse of constructed input in different JUPITER API applications and site-specific problems.
4. Data-exchange files for improved communication between applications and with other computer programs.
5. Methods that allow the application programmer and (or) user (Fig. 1) to control the level of detail reported to the main output file(s).
6. A useful set of statistical and sensitivity-analysis techniques, some of which are not currently available through other resources.
7. Sophisticated methods of accounting for data error.
8. An equation capability that can be used by the application programmer to allow the user great flexibility in defining parameters, observations, and other quantities.
9. An easy procedure for programming new methods of analysis.

Existing programs, such as early versions of UCODE (Poeter and Hill, 1998)³ and current versions of PEST⁴ (which are both inverse modeling codes that can be used with any process model) have many of the missing capabilities provided by the JUPITER API. However, their structures are not designed to support development of alternative ideas by other developers.

The JUPITER API has already been used to construct a number of applications. Three simple

²<http://endo.sandia.gov/DAKOTA/software.html>.

³<http://typhoon.mines.edu/freeware/ucode/ucode-PREJUPITER/index.shtml>.

⁴<http://www.sspa.com/Pest/index.shtml>.

¹http://water.usgs.gov/nrp/gwsoftware/jupiter/jupiter_api.html.

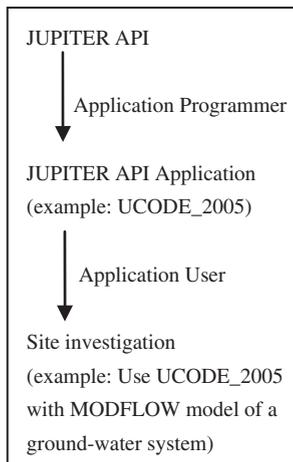


Fig. 1. Role of application programmers and application users.

applications are presented by Banta et al. (2006) (see footnote 1). More complicated applications include the universal inverse code UCODE_2005 (Poeter et al., 2005)⁵, 6 other applications distributed with UCODE_2005 (Residual_Analysis is used to discuss capability 4 in this article), the multi-model averaging code MMA (Poeter and Anderson, 2005; Poeter and Hill, 2007)⁶, and a code named OPR_PPR (Tonkin et al., 2007)⁷ for assessing data importance to process-model predictions using the local sensitivity Observation-PRediction (OPR) and Parameter-PRediction (PPR) statistics (Tiedeman et al., 2003, 2004). The existing applications can provide a useful starting point for building new applications.

Source code, documentation, and example input and output files for the JUPITER API and the applications listed above can be downloaded from the URLs listed in footnotes in this article. The International Ground Water Modeling Center maintains a web site with links to non-US Geological Survey applications.⁸

The rest of this article provides descriptions and examples of the 9 capabilities of the JUPITER API listed above, mostly using examples provided by UCODE_2005.

1. Interacting with complex process models

Many model analyses require that a process model be executed repetitively. Here are two examples:

- (a) When conducting local sensitivity analyses, sensitivities (the derivatives of simulated quantities with respect to parameters) are needed. To calculate sensitivities by a forward-difference perturbation method, the application needs to repeat four tasks: (1) create one or more process-model input files with parameter values that change with different repetitions, (2) execute the process model, (3) extract (read) the simulated values of interest, and (4) use the values to calculate sensitivities. The first repetition uses a base set of parameter values. In subsequent repetitions one parameter value is changed slightly from the base values.
- (b) When conducting Monte Carlo runs, repeated process-model runs are executed in which the model is in some way changed. The changes may be in the parameter values or some other aspect of the system.

In the JUPITER API, the repeated runs are accomplished using control loops, each of which can run the process model one or, if the application uses the parallel capability, multiple times. A control loop structure suitable for many applications and used by UCODE_2005 is diagrammed in Fig. 2 and is discussed in the following text. In the text, the phrases in bold identify tasks listed in Fig. 2. The numbers are consistent with those used in Fig. 2. Other terms used in Fig. 2 or its caption are underlined here.

- **Initialize:** Read input, allocate memory, and perform preparatory steps as needed.
- Start control loop (the JUPITER API provides the programming needed to parallelize repeated executions of all or part of a control loop as discussed below for capability 2.)
 - 1) **Define job:** Define the job of the current execution of the control loop. Control loops can repeat or skip tasks, as needed.
 - 2) **Generate parameter values:** Perform any needed calculations and populate an array with one or more sets of parameter values. For example, the JUPITER API provides tools for generating parameter values needed

⁵<http://typhoon.mines.edu/freeware/ucode/>.

⁶<http://typhoon.mines.edu/freeware/mma/>.

⁷<http://water.usgs.gov/software/OPR-PPR.html>.

⁸<http://www.mines.edu/igwmc/jupiter>.

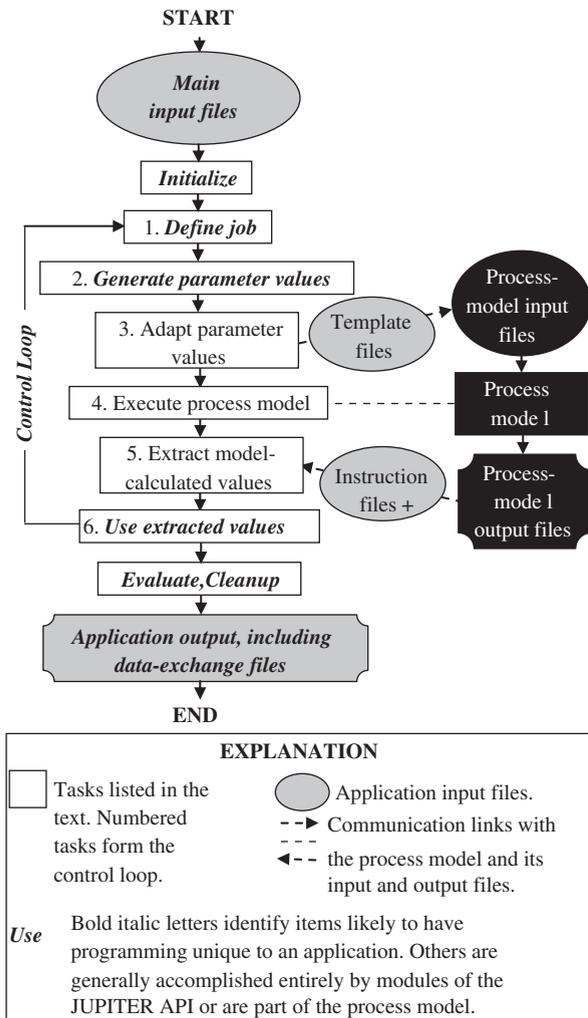


Fig. 2. Flowchart showing how an application such as UCODE_2005 is constructed using JUPITER API. Other applications could be similar or have features removed, added, or reorganized. JUPITER API modules and conventions can be used to design and construct everything shown except process model and its output files, and design of process-model input files. Process model is used to represent system of interest; for example, MODFLOW, PRMS, and (or) HSPF might be used. “+” after “Instruction files” refers to derivatives interface file, which can be used to read sensitivities produced by process model.

to calculate perturbation sensitivities and UCODE_2005 provides application-specific programming for this task to generate parameter values using modified Gauss–Newton methods (see the discussion for JUPITER API capability 9).

3) **Adapt parameter values:** Convert a set of parameter values to a form usable by the process model. The approach depends on

whether the process model is accessed as a program external to or included within an application of the API. When the process model is an external program, as shown in Fig. 2, this task generally involves creating one or more of the process-model input files. JUPITER API modules can use template files to create process-model input files that are ASCII format (also called text only; this is a common format for input files). Template files are constructed by application users by replacing input values with character strings in copies of process-model input files. The approach is quite general and can be used to manage many process-model input files. As indicated by the “Adapt Parameter Values” box in Fig. 2, programmers can make this capability available in an application without additional programming. When the process model is implemented as a subroutine, this task may involve using the parameter values to populate data arrays used by the model.

4) **Execute process model:** Invoke the process model. The application user provides a system command for executing the process model. (Depending on the compiler and operating system used to produce the program executable, minor editing of the JUPITER API source code may be required to enable a system command to be invoked. This is transparent to most users.)

5) **Extract model-calculated values:** Populate an array with selected values produced by a process model. The approach depends on whether the process model is accessed as a program external to or included within an application of the API. When the process model is an external program, this task involves reading one or more process-model output files to obtain the values. JUPITER API modules can navigate and read values from files in ASCII format using instruction files constructed by application users with conventions established by the JUPITER API. The approach is quite general and can be used to read from many process-model output files. If the process model calculates sensitivities, these can be read according to specifications provided by the user in a derivatives interface file. As indicated by the “Extract Model-Calculated Values” box in Fig. 2, programming is not required for most applications

built using the JUPITER API. When the process model is implemented as a subroutine, this task generally requires coding to store values.

- 6) **Use extracted values:** This task performs data manipulation using model-calculated values. For example, calculation of perturbation sensitivities from parameter sets and model-calculated values is done in this task.
- **End control loop:** Loops may end here, or some or all of the Evaluate task may occur within the control loop.
- **Evaluate:** This task performs application-dependent analyses.
- **Cleanup:** Delete unneeded files and (or) deallocate memory.

In JUPITER API applications, control loops are application-specific and require programming by the application developer. The programming of control loops can be achieved using, for example, C, Java, or Fortran. Applications that use the JUPITER API can incorporate all or some of the tasks and loops mentioned above, or additional tasks and loops may be defined.

2. Parallel computing capabilities

The JUPITER API includes parallel computing capabilities using a robust dispatcher-runner protocol. The only requirement is network read and write access between computers. Commonly, the parallelization is applied to executions of the process model so that many of them are run simultaneously, but other aspects of a control loop could also be parallelized. The common case constitutes what is sometimes called an “embarrassingly parallel” problem, and, for tests using the simple application provided with the JUPITER API and using UCODE_2005, networks of computers in typical workplace settings achieved speedups nearly proportional to the number of processors available. For example, if 10 simulations could be run in parallel, using 10 computers, execution times were close to one-tenth of the time required by serial runs.

JUPITER API capabilities 3–6 described next relate to data input and output mechanisms. These mechanisms enable users to easily access a range of JUPITER API applications and, therefore, to try a range of analysis techniques on a problem of interest with a minimum of time and effort spent

learning how to construct input files and control output files.

3. Input design

The JUPITER API uses input blocks to achieve an input design that is flexible, largely self-descriptive, and facilitates reuse of constructed input in different applications and site-specific problems. Example input blocks are shown in Fig. 3. Input blocks have the basic structure:

```
BEGIN Blocklabel [Blockformat]
  Blockbody
END Blocklabel
```

where items in all capital letters need to be included literally. Other items are replaced by the user with options recognized by the application and often defined by the API. Items in square brackets are optional. The blocklabels used in Fig. 3 include, for example, “UCODE_CONTROL_DATA” and “OBSERVATION_DATA”. Each input block is structured according to one of three possible blockformats: TABLE, KEYWORDS, or FILES, as shown in Fig. 3. These options allow the application user to determine the complexity of each input block, as needed to read the data. Fig. 3 also shows keywords such as “optimize” and “maxiter”, which are used to identify the data within each input block. It is the use of blocklabels and keywords that make JUPITER API input files largely self-descriptive. Minimal programming on the part of the application developer is required to use input blocks.

The parameter-information input blocks of Fig. 3 display a useful feature of the JUPITER API. The “PARAMETER_GROUPS” input block defines a set of values that are used as defaults for members of a group of parameters with “GroupName = KPar”. One of those defaults is for “Adjustable”, which controls whether parameter values can be adjusted, for example, as part of parameter estimation. The members of the group are defined in the “PARAMETER_DATA” input block, where exceptions to the defaults and additional data are provided. In Fig. 3, the entries for “Adjustable” in the “PARAMETER_DATA” input block override the defaults. The JUPITER API provides similar capabilities for observations, and groups can readily be defined for application-specific quantities.

```

# -----
# a. OPTIONS
# -----
BEGIN Options TABLE
  NROW=1 NCOL=2 COLUMNLABELS
  Verbose Derivatives_Interface
    0 ..\ufiles\transient.derint
END Options

# -----
# b. UCODE CONTROL
# -----
BEGIN UCODE_CONTROL_DATA KEYWORDS
  ModelName=ex1
#Performance
  optimize=yes
#Printing and output files
  FinalRes=no #residuals
  FinalSens=css #sensitivities
  DataExchange=yes
END UCODE_CONTROL_DATA

# -----
# c. REGRESSION CONTROL
# -----
BEGIN REG_GN_CONTROLS KEYWORDS
  TolPar=0.01
  maxiter=10
  maxchange=2.0
END REG_GN_CONTROLS

# -----
# d. COMMANDS FOR PROCESS MODEL
# -----
BEGIN MODEL_COMMAND_LINES FILES
  ..\ufiles\obs-fwd.command
  ..\ufiles\obs-fwd-der.command
END MODEL_COMMAND_LINES

# -----
# e. PARAMETER INFORMATION
# -----
BEGIN PARAMETER_GROUPS KEYWORDS
  GroupName=KPar1
  Adjustable=yes TolPar=.001
  MaxChange=1.2 SenMethod=-1
END PARAMETER_GROUPS

BEGIN PARAMETER_DATA TABLE
  NROW=2 NCOL=3 COLUMNLABELS
  GroupName=KPar1
  ParamName StartValue Adjustable
    K1 3e-4 yes
    RCH1 46. no
END PARAMETER_DATA

# -----
# f. OBSERVATIONS
# -----
BEGIN OBSERVATION_DATA FILES
  ..\ufiles\hed.obs
  ..\ufiles\flo.obs
END OBSERVATION_DATA

# -----
# g. PROCESS MODEL
# -----
BEGIN MODEL_INPUT_FILES KEYWORDS
  modinfile=..\model\tc1.sen
  templatefile=..\ufiles\tc1.tpl
END MODEL_INPUT_FILES

BEGIN MODEL_OUTPUT_FILES KEYWORDS
  modoutfile=..\model\tc1_os
  instructionfile=..\ufiles\inst
  category=obs
END MODEL_OUTPUT_FILES

```

Fig. 3. This UCODE_2005 main input file is typical of JUPITER API applications. This file refers to other files, which are highlighted with gray. Keywords in bold are referred to in text. Keyword phrase identified by superscript “1” would need to be positioned at the end of preceding line in an actual input file.

To more easily enable a given modeling problem to be analyzed using multiple JUPITER-based applications, input files are designed such that the same input file can be used for multiple applications with little or no alteration. One feature that supports this is that when reading an input file, inapplicable input blocks are ignored. Thus, they need not be removed from the file when another application is used. Another feature is that when reading input blocks, the data associated with inapplicable keywords are ignored. Thus, capabilities that require almost the same set of data in an input block can easily be accommodated by constructing an input block with the data needed for both capabilities. As long as the same keyword is not used in two different ways, the application will simply ignore the keywords not needed for the analysis being pursued.

4. Data-exchange files

Data-exchange files are used primarily by other computer programs (e.g. other JUPITER API applications, or plotting, spreadsheet, and post-processing software). Data-exchange files are designed to be as simple as possible. Metadata related to the contents of data-exchange files are stored in other application-specific files, such as the xyzt files of UCODE_2005 (see Poeter et al., 2005).

Fig. 4 shows a data-exchange file that is produced by a JUPITER application named Residual_Analysis (Poeter et al., 2005), which is distributed with UCODE_2005, and the associated graph produced using Microsoft Excel. Like many applications of the JUPITER API, Residual_Analysis

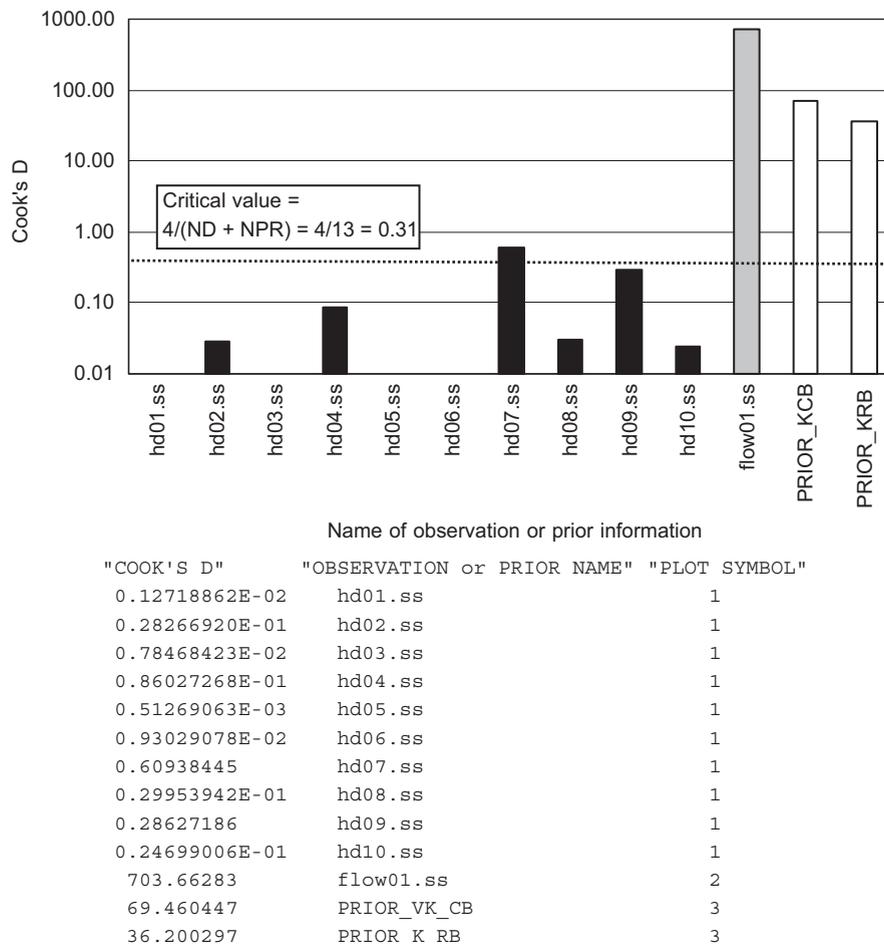


Fig. 4. UCODE_2005 produces JUPITER API data-exchange files that make it easy to produce graphs for evaluating models and data needs. This figure shows a graph of Cook's D, which is described in text. Observations include ten heads and one flow; there are two items of prior information. Graph shows that flow and prior information are most important to parameter values estimated by regression. Content of data-exchange file also is shown. Columns of data, column labels, and presence of a plot symbol variable that can be used in plotting are typical of JUPITER API data-exchange files.

both reads and produces data-exchange files. It reads the parameter variance-covariance matrix, the unscaled sensitivities, weighted residuals, and other data. It produces data-exchange files containing the statistics Cook's D, which measures the influence of each of the observations and prior information item on the set of estimated parameter values (see example in Fig. 4); *DFBETAS*, which measures the importance of each observation and prior information item on each estimated parameter; and data needed for graphical tests of the weighted residuals. See Cook and Weisberg (1999) and Hill and Tiedeman (2007) for additional information about these statistics and tests.

5. Controlling the level of detail reported to the main output file(s)

For each application constructed using the JUPITER API, there are generally one or more main output files designed to be read by the user to monitor program performance and evaluate results. Selected subroutines of the JUPITER API allow output to such files to be reported at one of five levels of detail; more detail generally is needed when the application user is debugging data sets, while less detail is convenient once a working run is achieved. The level of detail can be specified by the application developer, or the developer can allow it

to be determined by the application user. In the UCODE_2005 input file shown in Fig. 3, the user controls the level of output using the “Verbose” keyword in the “Options” input block.

6. Statistical and sensitivity-analysis techniques

The main purpose of the JUPITER API is to provide a platform for the development of new ideas, including new statistics. However, the JUPITER API provides a basic set of statistical capabilities that are available to application developers, as described briefly in Table 1. The statistical techniques for weighting observations and prior information are discussed in the following section. The fit-independent sensitivity-analysis statistics listed in Table 1 are not generally available from other sources; they are described by Hill and Tiedeman (2007).

7. Methods of accounting for data error

The JUPITER API has three design features that make it exceptionally useful in regard to accounting for data error. All are related to the weighting of observations and prior information, which, based on theoretical considerations, is how data error needs to be quantified (Draper and Smith, 1998, pp. 34, 222; Hill and Tiedeman, 2007, Guideline 6).

The first two design features are mentioned in Table 1 under “Weighting observations and prior information”. First, if the errors are thought to be independent, the weights for each observation or piece of prior information theoretically need to be proportional to $1/\sigma_i^2$, where σ_i^2 is the variance of the associated error. Yet variances or their inverse are difficult for most people to understand. More readily understood statistics are the standard deviation or coefficient of variation. The JUPITER API programming allows the user to define the weight, the square-root of the weight, the variance, the standard deviation or the coefficient of variation for each observation or item of prior information. If needed, the weight is then calculated internally. This allows more intuitive quantities to be listed in the input file, which reduces the chance of data input error.

Second, if the errors are thought to be correlated, the JUPITER API allows the user to specify a full error variance-covariance matrix. Examples of how such a matrix might be determined are discussed by Hill and Tiedeman (2007, p. 35, 298). The weight matrix equals the inverse of the error variance-

Table 1
Statistical capabilities provided by JUPITER API

Type of statistic	Comments
Critical values	For a 5-percent significance level for Student- <i>t</i> , χ^2 , and <i>F</i> distributions
Weighting observations and prior information	
Independent errors	Read weights or their square roots, or calculate weights from variances, standard deviations, or coefficients of variation
Correlated errors	Calculate a weight matrix from an error variance-covariance matrix. A complete or compressed matrix can be read. Internally, weight matrix is stored and manipulated as a compressed matrix
Sensitivity analysis	Fit-independent statistics: dimensionless, composite, and one-percent scaled sensitivities; leverage statistics; parameter correlation coefficients. These and unscaled sensitivities are printed to data-exchange files
Analyze model fit	
Objective-functions values	Sum of squared, weighted residuals and maximum-likelihood objective functions
Overall evaluation	Runs statistic: test for random distribution in time or space. Correlation between weighted residuals and standard normal statistics to evaluate normality and independence. Critical values are provided
Graphical evaluation	Produce data sets for spatial and temporal graphical analysis and five graphs, including normal probability graphs
Model discrimination criteria	Calculated error variance, standard error of regression, AICc, BIC, HQ, and KIC, determinant of Fisher information matrix
Uncertainty evaluation	
Estimated parameters	Parameter variance-covariance matrix, correlation coefficients, individual 95-percent confidence intervals
Variance needed for prediction intervals on predictions	Allows specification of variance, standard deviation, or coefficient of variation. Programming for prediction intervals is not included in JUPITER API

covariance matrix. The JUPITER API contains programming to calculate the inverse and the square-root of the inverse of the error variance-covariance matrix, as needed to calculate, for example, objective functions and weighted residuals, respectively.

The third design feature is that the JUPITER API allows matrices to be read as compressed matrices, uses compressed matrices in the typical matrix manipulations common to the types of analyses for which the JUPITER API is designed, and writes compressed matrices to data-exchange files. This is needed to reduce the computer storage that would otherwise be occupied by the typically large, sparse weight matrices commonly used to embody data error. For example, with 1000 observations, the storage required is reduced by a factor of about 500, from 10^6 to about 2×10^3 .

8. Equation capability

The JUPITER API includes an equation capability that can be used by the application programmer to allow the user great flexibility in defining parameters, observations, and other quantities. The equations can include a wide range of functions, such as arithmetic, exponential, geometric, and logarithmic, and any variable names allowed by the application programmer. For example, users of UCODE_2005 can define derived parameters using parameters defined in the Parameter_Data input block or previously defined in the Derived_Parameter input block. This can be used, for example, to lump parameters together or to redefine parameters to obtain quantities more directly related to prior information.

9. Easy procedure for programming new methods of analysis

The JUPITER API does not include, for example, local regression methods or global search methods for generating new sets of parameter values because there are no standard methods that are likely to be used in a large number of JUPITER API applications. Indeed, the JUPITER API is designed to support innovation in just these types of circumstances. The JUPITER API provides support for the more standardized or mechanical aspects of such applications. For example, when constructing UCODE_2005, two regression algorithms (Quasi-Newton and Trust Region with step size controlled

by a double-dogleg or hook-step strategy; see Dennis and Schnabel, 1996) were easily added, allowing rapid experimentation with these alternative methods. The steps needed to add each algorithm were as follows: (a) Program the new capability, using arrays of sensitivities, observations, simulated equivalents, weighting, and so on, produced by programming dominated by JUPITER API modules. (b) If needed, add control variables to the input specific to the algorithm. The modular construction of the API and the application make step (a) easy. The design of input blocks makes step (b) easy.

Programmers using the JUPITER API can take advantage of capabilities developed for earlier applications. For example, regression routines or the control-loop programming in UCODE_2005 might be of considerable use in some other applications.

10. Conclusions

Nine capabilities of the JUPITER API are used to show how the API can be used to build model-analysis applications (computer programs), and how the resulting applications enable users to access readily a range of analysis capabilities. The goal is for the JUPITER API to facilitate development of new ideas for use in the analysis of complex models, thus allowing the utility of these ideas to be evaluated more quickly than previously has been possible.

Acknowledgments

The US Environmental Protection Agency (EPA) provided financial support for Eileen Poeter and John Doherty. Additional assistance was provided by Karl Castleton and Steven Fine of the EPA, and Steve Markstrom, George Leavesley, and Arlen Harbaugh of the USGS. Charles Heywood and Claire Tiedeman of the USGS, Laura Foglia, a student at Eidgenössische Technische Hochschule (Swiss Federal Institute of Technology), in Zurich, Switzerland, and Matthew Tonkin of S.S. Papadopoulos and Associates tested and programmed JUPITER applications during development of the API and provided many useful suggestions. Reviews of the API documentation and this article by Richard Winston, Howard Reeves, Eve Kuniansky, and Steven Regan of the USGS, Philip Meyer of the Pacific Northwest National Laboratory, and

two anonymous reviewers substantially improved this manuscript. The persistence of one of the anonymous reviewers was notable and is responsible for the level of clarity attained in this article.

Although this work was reviewed by the EPA and approved for publication, it may not necessarily reflect official Agency policy. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

References

- Ahlfeld, D.P., Mulligan, A.E., 2000. *Optimal Management of Flow in Groundwater Systems*. Academic Press, San Diego, CA, 185pp.
- Aster, R.C., Borchers, B., Thurber, C.H., 2005. *Parameter Estimation and Inverse Problems*. Elsevier Academic Press, Amsterdam, 301pp.
- Babendreier, J.E., 2004. Calibration, optimization, and sensitivity and uncertainty algorithms application programming interface (COSU-API). In: *Proceedings of the International Workshop on Uncertainty, Sensitivity, and Parameter Estimation for Multimedia Environmental Modeling*, Rockville, MD, pp. A1–A54.
- 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. *US Geological Survey Techniques and Methods Report TM 06–E1*, 268pp.
- Burnham, K.P., Anderson, D.R., 2002. *Model Selection and Multi-Model Inference: A Practical Information-Theoretic Approach*. Springer, New York, 488pp.
- Cook, R.D., Weisberg, S., 1982. *Residuals and Influence in Regression*. Chapman & Hall, New York, 230pp.
- Cook, R.D., Weisberg, S., 1999. *Applied Regression Including Computing and Graphics*. Wiley, New York, 253pp.
- Dennis, J.E., Schnabel, R.B., 1996. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Classics in Applied Mathematics 16, Philadelphia, PA, USA, 378pp.
- Draper, N.R., Smith, H., 1998. *Applied Regression Analysis*, third ed. Wiley, New York, 706pp.
- Hill, M.C., Tiedeman, C.R., 2007. *Effective Groundwater Model Calibration, with Analysis of Sensitivities, Predictions, and Uncertainty*. Wiley, New York, 455pp.
- Menke, W., 1984. *Geophysical Data Analysis, Discrete Inverse Theory*. Elsevier, New York, 289pp.
- Meyer, P.D., Ye, M., Neuman, S.P., Cantrell, K.J., 2004. Combined estimation of hydrogeologic conceptual model and parameter uncertainty. *US Nuclear Regulatory Commission NUREG/CR-6843*, Pacific Northwest Laboratory PNNL-14534, 51pp.
- Parker, R.L., 1994. *Geophysical Inverse Theory*. Princeton University Press, Princeton, NJ, 400pp.
- Poeter, E.P., Anderson, D., 2005. Multi-model ranking and inference in groundwater modeling. *Ground Water* 43 (4), 597–605.
- Poeter, E.P., Hill, M.C., 1998. Documentation of UCODE: a computer code for universal inverse modeling. *US Geological Survey Water-Resources Investigations Report 98-4080*, 116pp.
- Poeter, E.P., Hill, M.C., 2007. MMA, a computer code for multi-model analysis. *US Geological Survey, Techniques and Methods Report TM-6E3*, Reston, VA 113pp.
- Poeter, E.P., Hill, M.C., Banta, E.R., Mehl, S., Christensen, S., 2005. UCODE_2005 and six other computer codes for universal sensitivity analysis, inverse modeling, and uncertainty evaluation. *US Geological Survey Techniques and Methods Report TM 6-A11*, 283pp.
- Saltelli, A., Chan, K., Scott, E.M., 2000. *Sensitivity Analysis*. Wiley, New York, 475pp.
- Saltelli, A., Ratto, M., Andreas, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S., 2007. *Global Sensitivity Analysis, The Primer*. Wiley, New York, 312pp.
- Saltelli, A., Tarantola, S., Campolongo, F., Ratto, M., 2004. *Sensitivity Analysis in Practice*. Wiley, New York, 219pp.
- Seber, G.A.F., Wild, C.J., 1989. *Nonlinear Regression*. Wiley, New York, 768pp.
- Sun, N.-Z., 1994. *Inverse Problems in Ground-Water Modeling*. Kluwer Academic Publishers, Boston, MA, 337pp.
- Tarantola, A., 2005. *Inverse Problem Theory*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 342pp.
- Tiedeman, C.R., Hill, M.C., D'Agnesse, F.A., Faunt, C.C., 2003. Methods for using groundwater model predictions to guide hydrogeologic data collection, with application to the Death Valley regional ground-water flow system. *Water Resources Research* 39 (1), 5-1–5-17.
- Tiedeman, C.R., Ely, D.M., Hill, M.C., O'Brien, G.M., 2004. A method for evaluating the importance of system state observations to model predictions, with application to the Death Valley regional groundwater flow system. *Water Resources Research* 40, W12411.
- Tonkin, M.J., Tiedeman, C.R., Ely, D.M., Hill, M.C., 2007. OPR-PPR, a computer program for assessing data importance to model predictions using linear statistics. *US Geological Survey, Techniques and Methods Report TM-6E2*, Reston, VA 115pp.