Towards Uncertainty Quantification and Parameter Estimation for Earth System Models in a Component-based Modeling Framework

Scott D. Peckham^{a,*}, Anna Kelbert^b, Mary C. Hill^c, Eric W. H. Hutton^a

^a*INSTAAR, University of Colorado, Boulder, CO 80309*

^b*CEOAS, Oregon State University, Corvallis, OR 97331*

^c*Dept. of Geology, University of Kansas, Lawrence, KS 66045*

Abstract

Component-based modeling frameworks make it easier for users to access, configure, couple, run and test numerical models. However, they do not typically provide tools for uncertainty quantification or data-based model verification and calibration. To better address these important issues, modeling frameworks should be integrated with existing, general-purpose toolkits for optimization, parameter estimation and uncertainty quantification.

This paper identifies and then examines the key issues that must be addressed in order to make a component-based modeling framework interoperable with general-purpose packages for model analysis. As a motivating example, one of these packages, DAKOTA, is applied to a representative but nontrivial surface process problem of comparing two models for the longitudinal elevation profile of a river to observational data. Results from a new mathematical analysis of the resulting nonlinear least squares problem are given and then compared to results from several different optimization algorithms in DAKOTA.

Keywords: model uncertainty, modeling frameworks, component-based modeling, optimization, inverse problems, nonlinear least squares, parameter estimation, longitudinal river elevation profiles

*Corresponding author

Email address: `Scott.Peckham@colorado.edu` (Scott D. Peckham)

1. Introduction

Many Earth science domains rely on numerical modeling to gain a better understanding of Earth system processes. Modeling addresses a wide variety of problems in the realms of climate, weather, hydrology, land surface dynamics, geodynamics, geophysics, hydrogeophysics and structural geology, among others. Earth system models are based on physical, chemical, biological and stochastic processes that make it theoretically possible to predict changes likely to occur at, below, or above a particular location on Earth in response to various types of forcing. Data-based model verification and validation – including more formal data integration through model parameter estimation – and quantification of ever-present uncertainty are critical in order to develop reliable numerical models for observed Earth processes.

The Community Surface Dynamics Modeling System, or CSDMS, is one example of a component-based modeling framework (Peckham et al., 2013; Syvitski et al., 2014), employed in the realm of Earth surface process dynamics, with capabilities currently being extended to deep Earth process modeling. Just as CSDMS provides interoperability and coupling mechanisms for process-based models, it could also provide simplified access to model analysis programs. In this paper, we discuss extensions to CSDMS that would be required for its component-based framework to interoperate with uncertainty quantification and parameter estimation (inverse modeling) toolkits.

2. Background: Models and Modeling Frameworks

2.1. What is a Model?

There are many possible definitions of the word *model*. This paper is concerned with *computational models* that predict the evolution of one or more system state variables over time as a function of observations at a given start time. These predictions are made using a set of equations that express laws of physics and other constraints on the problem of interest. Laws of physics are often expressed as differential equations that include a time derivative, and computational models use a discretization of space and time and some combination of *numerical methods* to solve these governing equations. Models generally require values for one or more *input variables*, often used to describe the initial state of the system and often specified as spatial scalar or vector fields. These may be measured or estimated and

are distinct from the model's *design parameters* (also called control, model or configuration parameters), that must be specified in the equations that define the model itself. A model run generates numerical values for *output variables* (i.e. simulated observations or predictions) that can be compared to observations. A very simple example is given by $y = cx^p$, where x and y are input and output variables, respectively, and c and p are design parameters.

2.2. What is a Modeling Framework?

Over the last decade, a number of different *modeling frameworks* have emerged, both within academia and at several different federal agencies. An example from the academic modeling community is the NSF-funded CSDMS project (cited in the Introduction) which primarily serves the Earth surface process modeling community. Other examples from the federal or operational modeling community include:

- ESMF (Earth System Modeling Framework), which primarily serves the atmosphere and ocean modeling community,
- OMS (Object Modeling System), developed by the USDA (US Department of Agriculture) primarily for agricultural modeling and
- FRAMES (Framework for Risk Analysis in Multimedia Environmental Systems), developed by the US EPA (Environmental Protection Agency), primarily for environmental modeling.

(Hill et al. (2004); David et al. (2002); Whelan et al. (1997)). A project called Earth System Bridge, funded as a building block in NSF's EarthCube initiative, is developing adapters that make it easy for any given model to be prepared as a plug-and-play component that can be used in (or moved between) multiple modeling frameworks, including those above.

The intent of all such modeling frameworks is to provide a software environment in which users can choose models from a collection and easily couple them to create customized, composite models in a plug-and-play manner. This facilitates code re-use and interoperability. The models in the collection may span a wide variety of different physical processes and are often written by many different authors, typically experts in their field. In many cases, the input variables required by one model can be provided by another model in the collection, so there is strong motivation to couple them.

69 However, the models typically differ in many ways, such as their program-
70 ming language, computational grid, time-stepping scheme, variable names
71 and units. In addition to providing a simple mechanism for coupling models,
72 modeling frameworks typically contain *service components* or *mediators* that
73 automatically reconcile differences between the models that would otherwise
74 prevent them from sharing variables. Examples of mediators include spa-
75 tial regridders, time interpolators, unit converters and semantic mediators.
76 These mediators and other capabilities of the framework – such as the ability
77 to write composite model output to different file formats with standardized
78 metadata, or to provide a graphical user interface (GUI) and help system –
79 provide both model users and developers with significant added value.

80 There is a strong interest in adding a new capability to modeling frame-
81 works, namely the ability to track and analyze uncertainty either for a single
82 (stand-alone) model or for a coupled set of models. For example, this is one
83 of the major goals of the second funding cycle of the CSDMS project. Since
84 several powerful, integrated packages for uncertainty analysis already exist
85 (Section 4), integrating one or more of them into a modeling framework seems
86 like the best way to achieve this goal. One such package, called DAKOTA
87 (Adams et al., 2013b,a) is of particular interest because it provides a unified
88 interface to a large collection of open-source packages for optimization and
89 uncertainty quantification.

90 DAKOTA and similar packages offer an impressive suite of uncertainty
91 analysis tools, including tools for model sensitivity analysis (e.g. sampling
92 methods to explore the design parameter space) as well as inverse modeling
93 (or parameter estimation). However, the sensitivity analysis tools are easier
94 to integrate within a modeling framework because they don’t usually re-
95 quire capabilities beyond what a typical model (or composite model) already
96 provides. By contrast, inverse modeling requires construction of a suitable
97 objective function and computation of derivatives and is also affected by how
98 models are coupled. So although we are interested in bringing all of the ca-
99 pabilities of packages like DAKOTA into modeling frameworks like CSDMS,
100 this paper will focus on what a modeling framework must do to accommodate
101 inverse modeling. To set the stage, the next section provides a very brief,
102 self-contained overview of inverse modeling. For a more extensive treatment,
103 see Tarantola (2005), Caers (2011) or Aster et al. (2013).

104 3. Background: Inverse Modeling Methods

105 *Forward modeling* simply refers to running a computational model for a
 106 *given* set of input variables and design parameters to generate output vari-
 107 ables. *Inverse modeling* refers to efforts to invert this process, that is, to
 108 determine what a model’s input variables and/or design parameters would
 109 need to be set to in order to generate a *given* set of output variables. In most
 110 cases, this inverse problem is *ill-posed*, meaning that there is not a unique
 111 set of input variables and design parameters that can produce a given out-
 112 put, but rather multiple sets. However, *regularization methods* can be used
 113 to introduce additional criteria that discriminate between and preferentially
 114 select from these multiple sets.

115 A forward model’s performance can be quantified by defining a metric
 116 that measures the “distance” between its output variables (or predictions
 117 or simulated data) and independent measurements (or observations). Dif-
 118 ferences between corresponding observed and predicted values are known as
 119 *residuals*, and this metric – known as the *penalty, cost or objective function*
 120 (Section 3.1) – is typically a function of the residuals, input variables and
 121 design parameters. Inverse modeling is concerned with how to make forward
 122 models perform as well as possible (model calibration), or with seeking the
 123 optimal input variables to predict observations to within measurement error.
 124 They therefore make use of *optimization methods* that seek to minimize an
 125 objective function, often subject to additional constraints.

126 Earth system modelers range widely in their familiarity with and adoption
 127 of inverse modeling methodology. For example, groundwater modelers have
 128 a long history of using inverse models, while sediment transport modelers do
 129 not. Inclusion of these methods in modeling frameworks should encourage
 130 broader use of these methods.

131 3.1. Constructing an Objective Function

132 The *objective function* must be a *metric* that measures a forward model’s
 133 performance, or the abstract “distance” between observed and model-predicted
 134 or simulated values. There are many different metrics that can be used, such
 135 as those based on the one-parameter family of L^p norms, given by

$$136 \quad \|\mathbf{y}^{(obs)} - \mathbf{y}^{(sim)}\| = \left(\sum_{k=1}^n \left| y_k^{(obs)} - y_k^{(sim)} \right|^p \right)^{1/p} \quad (1)$$

137 where $\mathbf{y}$ is a vector with components y_k and $p > 0$ is a scalar. The case
 138 where $p = 2$, or the L^2 norm, is the basis of the popular *least squares* metric.
 139 While this metric gives disproportionate weight to outliers, it ensures that
 140 the derivative of the objective function is continuous where the error is zero.

141 While (1) provides a simple measure of model performance, it is well
 142 known in statistics that model fit can be too good. In inversion, we are
 143 therefore only interested in optimizing the objective function to within the
 144 data errors, and seek to avoid *overfitting* the data. To that effect, the general
 145 penalty function (1) is usually modified to scale the residuals by a data
 146 covariance. For examples, see Doherty and Welter (2010) and Foglia et al.
 147 (2013).

148 As mentioned previously, many inverse problems are ill-posed, meaning
 149 that not one or several, but sometimes a subspace in the model param-
 150 eter space will satisfy the measured data to within the measurement errors.
 151 In some domains, including hydrology (Beven and Binley, 1992; Beven and
 152 Freer, 2001; Beven, 2006), this problem is known as *equifinality*. We are thus
 153 typically interested in obtaining not just any solutions, but the smoothest
 154 among those that are satisfactory. We may also want the solution to be as
 155 close to our *a priori* knowledge about the model (or design) parameters, as
 156 possible, while still fitting the measured data. Many of these objectives are
 157 obtained with a set of methods called *regularization*. Details of these methods
 158 are provided by Tikhonov (1963), Parker (1984), Hill and Tiedeman (2007)
 159 and Renard et al. (2011).

160 3.2. Optimization Methods

161 Given an objective function, there are a variety of optimization methods
 162 for finding either its minima or maxima, as required. Most of these can be
 163 classified as local and global (the others are hybrid). *Local methods* start
 164 somewhere in the design parameter space and then search in that vicinity
 165 for a *local extrema* — the “closest” one to the starting point, in some sense.
 166 Some local methods are gradient-based and some are gradient-free. *Global*
 167 *methods* seek global extrema, and therefore have some way of looking (or
 168 sampling) everywhere, not just near a starting point.

169 A classic optimization test problem is to find the (single) global minimum
 170 of the *Rosenbrock function* (Rosenbrock, 1960):

$$171 \quad f(p_1, p_2) = (a - p_1)^2 + b(p_2 - p_1^2)^2. \quad (2)$$

Figure 1: The Rosenbrock function, a classic test problem in optimization.

172 The second, quartic term is a valley-shaped surface that achieves its minimum
 173 value of zero along the parabola $p_2 = p_1^2$. Increasing the value of b gives
 174 steeper valley walls and increases the difficulty of this optimization problem
 175 (Lampton, 1997). The addition of the first term results in a function, f , with
 176 a single global minimum at the point $(p_1, p_2) = (a, a^2)$, where $f(p_1, p_2) = 0$.
 177 Typically one sets the constants to $a = 1$ and $b = 100$. Here, p_1 and p_2 are
 178 the model *design parameters* that are varied to improve the design of the
 179 model.

180 This problem provides a good test of an optimization algorithm because
 181 the global minimum lies along the bottom of a narrow valley with steep walls
 182 and a very flat bottom. While it is easy for algorithms to find the valley,
 183 it is difficult to converge to the location of the global minimum within this
 184 valley. The Rosenbrock function, shown in Figure 1, is used as an example
 185 in the DAKOTA package and provides context for our example problem to
 186 be examined in Section (5).

187 3.2.1. *Derivative-Based Optimization*

Derivative-based optimization algorithms require computing the gradient and/or Hessian of an objective function. The gradient is the vector of first derivatives of the objective function with respect to each of the continuous *design parameters*, while the *Hessian* is a square matrix of the mixed second derivatives. A *Jacobian* is a matrix of gradient vectors for multiple functions. If $f(x_1, \dots, x_n)$ is the objective function of the design parameters, the gradient and Hessian matrix are

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}, \quad H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \quad (3)$$

188 Derivative-based methods include: Gradient Descent, Conjugate Gradient,
189 Sequential Quadratic Programming (SQP) and Newton Methods and Method
190 of Feasible Directions (MFD) (DAKOTA includes multiple variants of these
191 and more.). All derivative-based methods require repeated derivative evalu-
192 ations at (typically) at least several dozens to several hundreds points in the
193 model parameter space. They are best suited to finding local minima near
194 initial guesses.

195

196 Many models of physical processes are based on mathematical functions
197 that have continuous first and second derivatives with respect to their pa-
198 rameters. In addition, many optimization problems can be formulated in
199 terms of cost (or penalty) functions that have continuous first and second
200 derivatives. For these types of models, it is often possible to find local ex-
201 trema (stationary points) of the function using standard methods of calculus,
202 i.e. by determining locations where derivatives are equal to zero. (See Sec-
203 tion 5.) A second derivative test can then be used to determine whether a
204 minimum or maximum occurs at that location. For these analytic models,
205 the derivative can be computed by *symbolic differentiation* and model code
206 can be written to return the resulting functions evaluated at required points
207 in the parameter space. However, this situation is less common in practical
208 geoscience applications.

209

210 For the typical case where computational models do not provide analytic
 211 derivatives of their output variables with respect to the design parameters,
 212 brute force *numerical differentiation* is typically required. To compute the
 213 derivative of the objective function by numerical differentiation, each of the
 214 model design parameters, x_i , is, in turn, slightly perturbed around its local
 215 value. The objective function is evaluated for the unperturbed and the per-
 216 turbed parameter, and the difference is taken to estimate one entry in the
 217 gradient vector. The number of forward model runs to compute the gradient
 218 vector at a point with this method is thus $(N_m + 1)$, where N_m is the num-
 219 ber of design parameters and becomes impractical for large parameter spaces.

220
 221 The third type of differentiation is distinct from the first two classical
 222 types and is usually called *automatic differentiation*. It exploits the fact
 223 that every computer program, no matter how complicated, performs calcu-
 224 lations by combining elementary arithmetic operations (addition, subtrac-
 225 tion, multiplication, division, powers, etc.) with evaluations of elementary
 226 functions (exp, log, sin, cos, etc.). By applying the chain rule to this se-
 227 quence of operations it is possible to automatically compute derivatives of
 228 arbitrary order that are accurate to the machine’s working precision while
 229 using only several times more operations than the original program. Many
 230 such tools have now reached maturity (e.g., OpenAD, Utke et al. (2014);
 231 see <http://www.autodiff.org/> for a comprehensive list). With automatic
 232 differentiation, the cost of computing the gradient vector varies, but typi-
 233 cally takes as much time as several (e.g. 2 to 12) forward model runs, which
 234 could be a great improvement over numerical differentiation for large model
 235 parameter spaces.

236
 237 Finally, the *adjoint state method* is available for computing the gradients
 238 at the cost of 2 model runs, by making clever use of certain symmetries in
 239 computational model formulations. Errico (1997) provides an accessible in-
 240 troduction to this method. It requires writing an alternative *adjoint model*,
 241 which is similar to the forward model and typically has very similar or identi-
 242 cal physics, except for certain sign conventions. However, various convergence
 243 complications may arise making the writing of the adjoint code a somewhat
 244 nontrivial task. Once the adjoint code exists, it may be used for repeated,
 245 extremely efficient, and accurate derivative calculations.

246 3.2.2. Derivative-Free Optimization

247 Derivative-free methods do not require computing derivatives of the ob-
248 jective function and can therefore be used for a larger class of optimization
249 problems where continuous derivatives may not exist (including problems
250 with discrete parameters). These methods can be classified as either *local*
251 or *global*. *Local methods* use a variety of different algorithms for search-
252 ing the parameter space for optimal solutions and for refining or focusing
253 the search in the vicinity of good solutions to find better solutions. Exam-
254 ples include: Pattern Search methods (e.g. Asynchronous Parallel Pattern
255 Search, COLINY Pattern Search and Mesh Adaptive Search), Simplex meth-
256 ods (e.g. Parallel Direct Search, COBYLA and Nelder-Mead) and Greedy
257 Search Heuristic (e.g. Solis-Wets method). *Global methods* simultaneously
258 search across the entire parameter space. Examples include: Division of
259 RECTangles (DIRECT) and Evolutionary Algorithms (EA), which are based
260 on concepts from Darwin’s Theory of Evolution and concepts from genetics
261 (e.g. natural selection, reproduction, mutation, crossover, inheritance and
262 recombination). Implementations of these and other derivative-free methods
263 are also available and documented within the powerful DAKOTA toolkit.

264 *Population-based optimization* is another important class of gradient-free
265 optimization methods. Examples include: Genetic Algorithms (e.g. from
266 the larger class of Evolutionary Algorithms), Memetic Algorithms (based on
267 the concept of *memes*, which combine an evolutionary or population-based
268 algorithm with individual learning or local improvement procedures), Swarm
269 Algorithms (e.g. Ant Colony Optimization, Particle Swarm Optimization,
270 Intelligent Water Drops), Harmony Search, Cuckoo Search and Differential
271 Evolution.

272 4. General-Purpose Software Packages for Inverse Modeling and 273 Uncertainty Quantification

274 There is a rich foundation in the area of inverse modeling and uncertainty
275 quantification that could potentially be incorporated into a component-based
276 modeling framework. Each of the software packages listed here is easy to
277 obtain (many are open-source) and provide a rich collection of methods.

- 278 • DAKOTA, 2015 (Design Analysis Kit for Optimization and Terascale
279 Applications) <http://dakota.sandia.gov/software.html>

- 280 • UCODE / Jupiter API, 2015 (Joint Universal Parameter Identification
281 and Evaluation of Reliability, (Poeter et al., 2014))
282 <http://igwmc.mines.edu/freeware/ucode/>
- 283 • PSUADE, 2014 (Problem Solving environment for Uncertainty Analy-
284 sis and Design Exploration, (PSUADE, 2014))
285 http://computation.llnl.gov/casc/uncertainty_quantification/
- 286 • PEST, 2015 (Model-Independent Parameter Estimation and Uncer-
287 tainty Analysis) <http://www.pesthomepage.org/Home.php>
- 288 • Ostrich, 2008 (Optimization Software Toolkit)
289 [http://www.civil.uwaterloo.ca/lsmatott/Ostrich/OstrichMain.](http://www.civil.uwaterloo.ca/lsmatott/Ostrich/OstrichMain.html)
290 [html](http://www.civil.uwaterloo.ca/lsmatott/Ostrich/OstrichMain.html)
- 291 • TAO, 2014 (Toolkit for Advanced Optimization), with a focus on gradient-
292 based search methods
293 <http://www.mcs.anl.gov/research/projects/tao/>
- 294 • QUESO, 2014 (Quantification of Uncertainty for Estimation, Simula-
295 tion and Optimization)
296 <https://red.ices.utexas.edu/projects/software/wiki/QUESO>
- 297 • STEPS, 2012 (Stochastic Engine for Pathway Simulation)
298 <http://steps.sourceforge.net/STEPS/default.php>

299 Fortunately, there is a fairly standard mechanism that inverse modeling
300 or uncertainty analysis applications use to interact with process models, such
301 as those in CSDMS, as illustrated in Figure 2. After the user selects and con-
302 figures an analysis method, the application generates the input data needed
303 for that method, saves it to the model’s configuration file (using a blank
304 template), and runs the model repeatedly with different inputs. After each
305 model run, a post-processing script scrapes results from the model’s output
306 files, which may include evaluation of a cost function or its derivatives, and
307 uses these results to perform the analysis. In a modeling framework, the
308 model to be analyzed may be a composition of many separate but connected
309 component models.

310 Many of these general-purpose toolkits are implemented as Python pack-
311 ages, or C++/Fortran libraries, to streamline integration with computational
312 models; some others would be harder to integrate. Each has unique features.

Figure 2: Flowchart showing how an inverse modeling or uncertainty analysis application such as DAKOTA or UCODE (represented by blue and white boxes) typically interacts with a process model, such as those in CSDMS (red boxes). Modified from Banta et al. (2008).

Python packages for uncertainty quantification include: *uncertainty*, *soerp* and *mcerp*. (See <http://pythonhosted.org/uncertainties/> for more information.)

5. Example Problem: Finding Best Fits to Longitudinal Elevation Profiles

We now illustrate a small part of what a toolkit such as DAKOTA has to offer, using a type of problem that will be familiar to many earth surface process modelers. This particular problem — which at first glance appears to be quite simple — actually represents a nontrivial optimization problem (similar to the Rosenbrock problem in Section 3.2) that requires more sophisticated optimization algorithms.

Peckham (2015) reviews three different physics-based derivations that predict a particular functional form for the longitudinal elevation profiles of rivers (i.e. elevation as a function of distance downstream from a drainage divide). These are concave-up functions that are steep near the drainage divide or ridgetop, but which rapidly flatten out with increasing distance downstream. The functions contain parameters that relate to the physics of the problem, but which are not easily measured. For this example, two different models will be considered. The first, derived by Whipple and Tucker

332 (1999), can be written as:

$$333 \quad z(x) = z_0 - (c/p) (x^p - x_0^p), \quad p \neq 0. \quad (4)$$

334 While this looks simple, the parameters p and c are actually functions of eight
 335 other physical and geometric parameters. The design or fitting parameters
 336 for this model are c and p , and $z(x_0) = z_0$ for any values of c and p . Here,
 337 z_0 is the elevation at a distance x_0 from the ridgetop, and we typically take
 338 $x_0 = 0$. However, this model has an unrealistic and infinite slope at $x = x_0$
 339 when $x_0 = 0$. A second model, derived by Peckham (2015), predicts that

$$340 \quad z(x) = z_0 + \frac{1}{p_\gamma R^\star} \{ S_0^{\gamma+1} - [S_0^\gamma + R^\star (x - x_0)]^{p_\gamma} \}. \quad (5)$$

341 Here, $p_\gamma = (\gamma+1)/\gamma$, and the design parameters are γ (an exponent in a slope-
 342 discharge formula) and $R^\star$ (a rescaled, geomorphically effective rainrate).
 343 This function includes S_0 , the measured, finite slope at x_0 , and has $|z'(x_0)| =$
 344 S_0 . The theoretical value of γ typically lies between -1 and 0 . If we treat S_0
 345 as given by direct observation, then both models have two design parameters.

346 For this example, both models are compared to observational data for the
 347 main channel of Beaver Creek, Kentucky, as measured by RiverTools (Peck-
 348 ham, 2009) from a digital elevation model with a grid-spacing of 10 meters.
 349 This data set consists of 2426 (x_k, z_k) pairs, with $x_0 = 0.0$, $z_0 = 668.33$
 350 (meters), and $S_0 = 0.461$. A nonlinear, least-squares cost function was con-
 351 structed from the residuals using the L^2 norm; see (1) and Appendix A.
 352 While most optimization algorithms can find the best-fit parameters for the
 353 second model, the first model leads to a nontrivial optimization problem.
 354 For example, we tried the well-known Levenberg-Marquardt (LM) algorithm
 355 (Moré, 1977) — a robust and adaptive algorithm which combines the ben-
 356 efits of the gradient descent and Gauss-Newton methods, and which is able
 357 to solve the Rosenbrock problem. We tried the implementation of the LM
 358 algorithm in IDL (Interactive Data Language) called LMFIT, and the one in
 359 MatLab, provided as an option for *lsqnonlin*. Both could get close but failed
 360 to converge for this model, even after 5000 double-precision iterations with
 361 a tolerance of 10^{-4} , and even when starting near the best-fit values. The
 362 LM algorithm is used by or available in many other curve-fitting packages,
 363 including Microsoft Excel.

364 A new mathematical treatment of this nonlinear least squares problem
 365 is given in Appendix A, for a class of models that includes (4) as a spe-

Table 1: Best-fit parameters for the Whipple-Tucker (1999) model to the Beaver Creek main profile data.

Optimization Method (DAKOTA or theory)	p_0	c_0	R^2
Method in Appendix A	0.133	14.68	0.90
NL2SOL (analytic gradients)	0.133	14.68	0.90
NL2SOL (numeric gradients)	0.133	14.68	0.90
OPT++ Gauss-Newton (analytic gradients)	0.133	14.68	0.90
OPT++ Gauss-Newton (numeric gradients)	0.133	14.68	0.90
Pattern Search (no gradients)	0.133	14.68	0.90
Evolutionary Algorithm (no gradients)	0.130	14.82	0.90

cial case. It allows the best-fit c and p to be computed quickly and reliably and provides a way to evaluate the results from the various algorithms. Tables 1 and 2 show the best-fit parameters to the Beaver Creek data set for the two profile models as computed by the method in Appendix A and by several optimization algorithms in DAKOTA. The corresponding input files and DAKOTA configuration files are available on GitHub (github.com/mcflugen/peckham_et_al_2016). The NL2SOL routine uses a generalization of the Levenberg-Marquardt algorithm, but converges for both models.

For the Whipple-Tucker model, the analytic gradient with respect to p contains terms with $x^p \ln(x)$, which approach 0 as x goes to zero for $p > 0$. This initially caused problems until the analytic gradient function was modified accordingly. It was also found that all methods were sensitive to the value of x_0 — perturbing x_0 from 0 to a small value such 0.01 resulted in significantly different best-fit parameters. Figure (3) shows the best fit of (4) to the Beaver Creek profile data, obtained with the NL2SOL method in DAKOTA. Figure (4) shows the best fit of (5) to the same profile data, obtained with the NL2SOL method in DAKOTA.

6. Sources of Model Error and How Modeling Frameworks Can Help Reduce Them

In addition to helping quantify model uncertainty, modeling frameworks can also be augmented to help reduce some, but not all sources of model error. To see how, it is helpful to consider various sources of error in some

Table 2: Best-fit parameters for the Peckham (2015) model to the Beaver Creek main profile data.

Optimization Method (DAKOTA)	γ	R^*	R^2
LMFIT in IDL (analytic gradients)	-0.701	0.0035	0.99
NL2SOL (analytic gradients)	-0.701	0.0035	0.99
NL2SOL (numeric gradients)	-0.702	0.0035	0.99
OPT++ Gauss-Newton (analytic gradients)	-0.701	0.0035	0.99
OPT++ Gauss-Newton (numeric gradients)	-0.702	0.0035	0.99
Pattern Search (no gradients)	-0.741	0.0041	0.99
Evolutionary Algorithm (no gradients)	-0.678	0.0031	0.99

Figure 3: Best fit of Whipple-Tucker (1999) model to Beaver Creek main channel profile, using NL2SOL algorithm in DAKOTA.

Figure 4: Best fit of Peckham (2015) model to Beaver Creek main channel profile, using NL2SOL algorithm in DAKOTA.

389 detail. Sources of error can be classified into four main categories, namely
 390 (1) model inadequacy, (2) input data inadequacy, (3) model parameter and
 391 input data uncertainty and (4) developer and user errors.

392 6.1. Model Inadequacy

393 Models may be inadequate for their intended purpose for a variety of
 394 reasons, such as

- 395 • Lack of knowledge of the true, underlying physics. For example, it was
 396 impossible to explain the observed precession of the planet Mercury
 397 using classical (Newtonian) mechanics, but it was finally explained in
 398 1916 by Einstein's more complete theory of general relativity (curvature
 399 of space-time near the Sun.) This lack of knowledge was responsible
 400 for what has been called an *unknown unknown*.
- 401 • Neglected effects or simplifying assumptions (e.g. air resistance)
- 402 • Numerical method and approximation problems (convergence, stability,
 403 consistency, fidelity, etc.)
- 404 • Model coupling problems (e.g. feedbacks, conservation problems)

405 The main ways to reduce uncertainty due to model inadequacy are: (1)
 406 scientific research to better understand physical processes and systems that

407 are poorly understood, (2) careful selection of numerical methods and (3)
408 various forms of testing. As for testing, there are five main things that
409 models can be tested or evaluated against, namely:

- 410 • Analytic solutions and test problems
- 411 • Measured data (i.e. observed vs. predicted)
- 412 • Valid range or reasonableness (e.g. sanity tests)
- 413 • Other models (especially for complex models, e.g. climate models)
- 414 • Their former selves (e.g. regression and unit tests, often automated)

415 It is straightforward to build this type of testing into a modeling framework,
416 and CSDMS has already started to build a collection of analytic solutions and
417 test data sets for this purpose. In addition, modeling frameworks result in
418 models being used by large groups of people, which leads to them becoming
419 more reliable and robust, particularly when their source code is open.

420 6.2. *Input Data Inadequacy*

421 Inadequate input data is another key source of model error. Note, that
422 even if the mathematics and physics of a model were perfect, perfect predic-
423 tions would not be possible because input data (e.g. initial conditions over
424 the model domain) will always be imperfectly known and incomplete. For
425 example, for spatial models, surrogates for actual measurements across the
426 model domain based on remotely sensed imagery are often the best available
427 data for initial conditions (e.g. soil moisture or rainfall rates). Issues include:

- 428 • Poor spatial or temporal resolution
- 429 • Poor quality
- 430 • Storage or transfer errors (e.g. byte order, data type, truncated files,
431 formatting, etc.)

432 The main ways to reduce this type of uncertainty are: (1) better data
433 collection methodologies and (2) careful data preparation and documentation
434 (with provenance metadata). Modeling frameworks provide an ideal platform
435 for guiding users through input data preparation steps, checking data for
436 errors, displaying documentation and managing metadata. As with models,
437 data sets become more reliable and robust when they are used by large groups
438 of people.

439 6.3. Model Parameter and Input Data Uncertainties

440 This category includes model calibration problems, which arise when the
441 model design parameters are not set to their optimal (or even reasonable)
442 values. Measurement or observation error in input data can also result in bi-
443 ased or incorrect simulated data or model predictions (a phenomenon known
444 as *aleatoric uncertainty*). DAKOTA and similar toolkits can provide model-
445 ing frameworks with powerful tools for assessing and quantifying this source
446 of uncertainty.

447 6.4. Developer and User Errors

- 448 • Incorrect implementation or developer error (e.g. bugs)
 - 449 – Regressions (bugs due to updates or improvements)
 - 450 – Bugs may be in software dependencies or in the model itself.
 - 451 – Coordinate projection, sign convention, or units mismatch (and
 - 452 failure to convert)
 - 453 – Problems at domain boundaries
- 454 • Preparation of input data (model setup)
- 455 • Incorrect or unintended use (e.g. unawareness of limitations)

456 Developer error can be reduced through a variety of best software devel-
457 opment practices, such as various types of testing (see above), version control,
458 use of design principles such as *separation of concerns*, use of standards and
459 frequent use. User errors can be addressed in a variety of ways, including:

- 460 • Documentation (e.g. tech tips, FAQs, manuals, tutorials, context help)
- 461 • GUIs (that can restrict possible inputs based on context)
- 462 • Software to check inputs, conditions, compatibility, etc.
- 463 • Training (and certification)
- 464 • Supervision by an expert

465 Most of these strategies would be straightforward to implement within a
466 modeling framework.

467 7. Towards Including Inverse Modeling and Uncertainty Quantifi- 468 cation in a Component-Based Modeling Framework

469 Inclusion of uncertainly quantification, parameter estimation and inver-
470 sion tools in a modeling framework inevitably requires a certain compromise
471 on efficiency, but has the potential to democratize these techniques, providing
472 Earth system modelers with fingertip access to model validation approaches,
473 without the typically steep learning curve and software development require-
474 ments. Here however, we aim to show that in a smart framework, this effi-
475 ciency compromise may not be a significant drawback. For this, we consider
476 several common use case scenarios, illustrated in Figure (5).

477 7.1. *Stand-alone Models*

478 In a modeling framework with access to “uncertainty tools”, a stand-
479 alone computational model could immediately make use of external param-
480 eter estimation tools based on derivative-free optimization (Section 3.3.2).
481 Derivative-based optimization methods (Section 3.3.1) could also be directly
482 employed, using numerical differentiation to iteratively compute derivatives
483 of the penalty function. However, both types of methods are only applica-
484 ble for models that have relatively few significant input model parameters
485 that are free to vary. For models with a large model parameter space, ad-
486 joint techniques must be employed to compute penalty function derivatives
487 for derivative-based optimization (see more details on the adjoints in Sec-
488 tion 3.3.1), and to compute the data sensitivities (or the full Jacobian) needed
489 by the uncertainly quantification techniques. These adjoint codes are most
490 efficient if hand-coded, and the model developer could provide an adjoint for
491 the system to use, if such a tool already exists. They would then immedi-
492 ately enjoy the full variety of techniques available in the framework, while
493 sacrificing little on the efficiency. For models that do not have an adjoint,
494 the use of automatic differentiation could be considered. (It should be noted
495 that an adjoint version of every generic interpolation routine supported by
496 the framework would also need to be provided to enable this functionality.)
497 Thus, the methods described here could be beneficial to model developers
498 who are interested in using and/or testing out a variety of inversion tech-
499 niques with minimal additional development investment. However, the true
500 value of the framework-based approach to inversion becomes evident when
501 inversion and/or parameter estimation for coupled models is considered. We
502 now discuss two distinct use cases.

(a) Stand alone computational model in a model coupling framework.

(b) Model coupling in series.

(c) Model coupling in parallel.

Figure 5: Model coupling configurations in a component-based framework.

503 7.2. Models Coupled in Series

504 This happens when the output of a numerical model A provides input to
505 a different numerical model B. The output of model B may then be directly
506 compared with measured data. The user may be interested in calibrating
507 the input parameters of model A to better match observations. For example,
508 a lithospheric deformation code could be coupled to a landscape evolution
509 code, which could in turn be validated by surface topography.

510 In this scenario, the model coupling framework will essentially bundle the
511 sequence of models into one for the purposes of an external model calibration
512 or inversion tool. For derivative-based optimization techniques, the derivative
513 of the penalty function with respect to the input parameters to the model
514 sequence would be obtained by chain rule, specifically applying the adjoint
515 of model B to the data residuals, and the adjoint of model A to the results
516 of the adjoint on model B to obtain the complete derivative of the penalty
517 function. For simpler models, brute-force derivative multiplication would be
518 performed. All of these options would be handled at the framework level,
519 making it possible to streamline complicated model validation and calibration
520 scenarios.

521 7.3. Models Coupled in Parallel

522 A very different model coupling setup occurs when a single set of param-
523 eters is input to more than one computational model. In this case, models
524 A and B have the same inputs, but they predict different variables, and are
525 validated by different data. The goal is then to use all data jointly to obtain
526 the best matching set of input parameters. Such is, for example, the problem
527 of joint inversion in geophysics: parameters such as temperature, pressure,
528 presence of melt, chemical impurities, and water content are the true pa-
529 rameters of interest. These may be used to compute the indirectly observed
530 Earth parameters, such as seismic velocities or electrical conductivity, which
531 are in turn used to compute the predicted observables at the Earth's surface.
532 A joint inversion would use both seismic and electromagnetic measurements
533 to constrain the Earth's structure. In general, this problem involves both
534 series and parallel model coupling.

535 In the parallel model coupling scenario, the adjoint would be obtained by
536 applying the two model adjoints, separately, to their respective data resid-
537 uals, and the resultant variations in the input model parameters would be
538 summed up to obtain the complete penalty function derivative. Again, this
539 would be handled at the framework level, allowing the complicated model

540 coupling scenario to be wrapped up for direct use in an external inversion
541 routine.

542 *7.4. New Modeling Framework Functionality*

543 Brute-force penalty function derivative computations for coupled models
544 require computing, storing and multiplying the Jacobian matrices. This be-
545 comes prohibitive for models that are nonlinear, with more than a few design
546 parameters, and/or long run times. If N_d is the dimension of the output vec-
547 tor field or the number of data points, and N_m is the the number of model
548 design parameters for one model, its Jacobian has $N_d \times N_m$ entries and re-
549 quires at least $\min(N_d + 1, N_m + 1)$ model runs if the adjoint is available, and
550 $(N_m + 1)$ model runs otherwise. The matrix would need to be computed for
551 each model in the coupling sequence, and for nonlinear models these com-
552 putations need to be performed repeatedly while the optimization algorithm
553 iteratively converges to a solution.

554
555 Fortunately, as discussed in Section 3.3.1, a derivative of the penalty
556 function may be obtained in $N_m + 1$ coupled model runs by numerical differ-
557 entiation, and in as few as 2 coupled model runs by an adjoint method. The
558 latter entails a call to apply the adjoint code to the weighted data residuals,
559 which in essence implements multiplication by transposed Jacobian to obtain
560 a perturbation of the model design parameters, without a direct computa-
561 tion or storage of the Jacobian matrix. These methods come at no additional
562 storage cost except for that of the intermediate solutions, needed to evaluate
563 the total penalty function derivative, and therefore make arbitrarily complex
564 coupling problems potentially tractable.

565
566 To summarize, several new capabilities must be added to a model frame-
567 work in order to support user-friendly inverse modeling. Specifically, it must
568 be able to: (1) store and perform arithmetic operations with the parameters
569 and data for each model, (2) interpolate in space and time to provide predic-
570 tions at observation locations, (3) read data in various formats and compute
571 data residuals, (4) provide a range of penalty functions to work with, (5) nu-
572 merically compute the derivative of the penalty function, or run the adjoint
573 if provided, (6) make use of numerical efficiencies to compute and manipulate
574 Jacobian matrices and higher order derivatives.

575 8. Summary and Recommendations

576 Computational models are a powerful means of understanding the Earth
577 system, making predictions and guiding decisions. Too often, however, mod-
578 els are used without any analysis of their uncertainty. The integration of
579 toolkits such as DAKOTA into component-based modeling frameworks will
580 help to resolve this by drawing attention to the problem and providing easy
581 access to powerful methods, thereby making it much easier for users to quan-
582 tify, assess and understand the uncertainty in models. Background informa-
583 tion on modeling frameworks and inverse modeling were provided in Sections
584 (2) and Section (3) and a list of major software toolkits for uncertainty quan-
585 tification and inverse modeling was given in Section (4). Since most of these
586 toolkits use the same approach to interacting with models, it appears feasible
587 for a modeling framework to provide access to more than one such package.
588 Section (5) used an example surface process problem to illustrate some of the
589 issues that are encountered in real applications and to give a taste of what
590 toolkits like DAKOTA have to offer modelers. In Section (6) it was argued
591 that the primary sources of error in computational models can be usefully or-
592 ganized into four groups — model inadequacy, input data inadequacy, model
593 parameter and input data uncertainty and developer and user errors — and
594 that modeling frameworks could address many of them. Finally, several tech-
595 nical issues regarding the inclusion of inverse modeling in a model coupling
596 framework were discussed in Section (7).

597 Key design criteria for including inverse modeling in a component-based
598 modeling framework are: (1) minimal changes to models, (2) minimal loss of
599 performance and (3) minimal effort for developers and users. Optimization
600 methods that are derivative-free or that work well with numerical derivatives
601 will be easiest to provide, but will still require tools for defining cost functions
602 and the ability to ingest observational data in different formats. Methods
603 that use analytic or automatic differentiation, as well as adjoint methods,
604 will require service components to be added to the modeling framework that
605 compute, store and manipulate Jacobians and Hessians. Separate methods
606 will be required for models coupled in series or in parallel and changes to com-
607 ponent model interfaces, such as the CSDMS Basic Model Interface (BMI),
608 may also be required.

609 Acknowledgments

610 The authors gratefully acknowledge funding through a cooperative agree-
 611 ment with the National Science Foundation (CSDMS, EAR-0621695) and
 612 NSF grant number (EarthCube, ICER-1343811).

613 Appendix A: Best-Fit Parameters for a Class of Profile Models

614 Peckham (2015) reviews several physics-based derivations that predict
 615 functional forms for the longitudinal elevation profiles of rivers (i.e. elevation
 616 as a function of distance downstream from a drainage divide). Several of these
 617 profiles can be expressed in the form

$$618 \quad z(x) = z_0 - c H(p, x, x_0), \quad (6)$$

619 where $H(p, x_0, x_0) = 0$. The results of this section apply to an otherwise
 620 arbitrary function H . The Whipple-Tucker profile is in a special subclass
 621 where $H(p, x, x_0) = f(p, x) - f(p, x_0)$, with $f(p, x) = (1/p)x^p$. Note that
 622 $x \geq x_0$ and the profile is *pinned* at the upper end to measured values since
 623 $z(x_0) = z_0$. Since real elevation profiles are decreasing, upward-concave
 624 functions of downstream distance, x , we require $z(x)$ to have these features.
 625 This can occur with $c > 0$ and $H(p, x, x_0)$ an increasing function of x , or
 626 with $c < 0$ and $H(p, x, x_0)$ a decreasing function of x . It is also desirable
 627 for the slope at x_0 , $S_0 = |z'(x_0)|$, to be *finite*, and for some models this only
 628 occurs if $x_0 > 0$.

629 For elevation profiles predicted by theory, the design parameters c and
 630 p are functions of both physical-process and empirical parameters, such as
 631 those that model a steady-state fluvial landscape where rainfall-induced ero-
 632 sion is exactly balanced by a steady and spatially uniform tectonic uplift
 633 rate. Theory may constrain the signs of c and p .

634 A *nonlinear least squares* fit of the model (6) to a measured profile of ele-
 635 vations, (x_k, z_k) , $k \in \{0, \dots, n\}$ seeks a parameter pair (c_0, p_0) that minimizes
 636 the following *cost function*:

$$637 \quad E(c, p) = \sum_{k=1}^n [z_k - z(x_k)]^2 = \sum_{k=1}^n [(z_k - z_0) + c H(p, x_k, x_0)]^2 \quad (7)$$

638 Figure (6) shows a contour plot for this cost function over the (c, p) plane for
 639 a particular set of measured elevation profile values, (x_k, z_k) and the special

Figure 6: Contour plot for a typical cost function when $f(x, p) = (1/p)x^p$, with extreme vertical exaggeration to show global minimum. For this example, $c_0 = 25$ and $p_0 = 0.166$.

Figure 7: The (difficult to distinguish) curves $c_1(p)$ (yellow) and $c_2(p)$ (blue) that cross at the best-fit (c, p) . For this example, they cross at $c = 28.2491$, $p = 0.14075$.

640 subclass mentioned above with $f(p, x) = (1/p)x^p$. Note that the global
641 minimum lies in a broad, flat valley, very similar to what happens for the
642 well-known Rosenbrock function (see Figure 1). This similarity appears to
643 explain why many curve-fitting algorithms struggle or fail to converge to the
644 minimum, at least when $f(p, x) = (1/p)x^p$. As with the Rosenbrock function,
645 it is easy for optimization algorithms to find the valley but remains difficult
646 for them to find the global minimum.

647 The nature of this broad, flat valley can be understood geometrically.
648 First, notice that for $n = 1$, $E(c, p)$ plots as a valley in the (c, p) plane that
649 attains its minimum value (zero in this case) along the entire curve given by
650 $c = p(z_0 - z_1)/(x_1^p - x_0^p)$. This curve diverges at $p = 0$ and rapidly decreases
651 with increasing p . The position of the valley depends on (x_0, z_0) and (x_1, z_1) ,
652 but there is no global minimum. The cost function (7) may be viewed as
653 a sum of such valley surfaces, all offset somewhat from one another, which
654 results in a surface with a global minimum that lies in a broad valley.

655 If the cost function (7) has a minimum, it must occur where all com-
656 ponents of its gradient vector (i.e. the partial derivatives with respect to c
657 and p) are equal to zero. Computing the derivative of (7) with respect to c ,
658 setting it to zero and solving for c , gives

$$659 \quad c_1(p) = \frac{\sum_{k=1}^n (z_0 - z_k) H(p, x_k, x_0)}{\sum_{k=1}^n H^2(p, x_k, x_0)}. \quad (8)$$

660 This is a curve in the (c, p) plane along which $\partial E/\partial c = 0$. Computing the
661 derivative of (7) with respect to p , setting it to zero and again solving for c ,
662 gives

$$663 \quad c_2(p) = \frac{\sum_{k=1}^n (z_0 - z_k) \partial H(p, x_k, x_0)/\partial p}{\sum_{k=1}^n H(p, x_k, x_0) \partial H(p, x_k, x_0)/\partial p}. \quad (9)$$

664 This is a curve in the (c, p) plane along which $\partial E/\partial p = 0$. As shown in
665 Figure (7), the curves $c_1(p)$ and $c_2(p)$ are very close to one another, and close
666 to the bottom of the valley, but they cross at a single point which gives the
667 best-fit (c, p) pair. Since the best-fit pair lies on the curve $c_1(p)$, we can insert
668 $c = c_1(p)$ into the cost function (7) to get a new cost function that depends
669 only on p , namely $E(p) = E(c_1(p), p)$. After simplification, we find that

$$670 \quad E(p) = \sum_{k=1}^n (z_0 - z_k)^2 - \frac{[\sum_{k=1}^n (z_0 - z_k) H(p, x_k, x_0)]^2}{\sum_{k=1}^n H^2(p, x_k, x_0)}. \quad (10)$$

671 Notice the power of 2 in the numerator of the last term, not present in (8).

672 For both $f(p, x) = (1/p)x^p$ and $f(p, x) = x^p$, this simplifies further to

673
$$E(p) = \sum_{k=1}^n (z_0 - z_k)^2 - \frac{[\sum_{k=1}^n (z_0 - z_k)(x_k^p - x_0^p)]^2}{\sum_{k=1}^n (x_k^p - x_0^p)^2}. \quad (11)$$

674 The best-fit p -value, p_0 , minimizes $E(p)$ for a given set of measured values
675 (x_k, z_k) . We can therefore find p_0 by computing the derivative of (11) with
676 respect to p , setting the result to zero and solving for p . While there isn't a
677 closed form expression for p_0 , a numerical root finder can be used. Another
678 simple approach is to evaluate (11) at 1000 equally-spaced p -values in the
679 interval $[0, 1]$ and then find the $p = p_0$ for which $E(p)$ is smallest. This
680 rapidly yields the best-fit value, p_0 , to three significant digits. Once p_0 has
681 been found, c_0 can be computed as $c_0 = c_1(p_0)$.

682 References

- 683 Adams, B., Ebeida, M., Eldred, M., Jakeman, J., Swiler, L., Bohnhoff, W.,
684 Dalbey, K., Eddy, J., Hu, K., Vigil, D., Bauman, L., 2013a. DAKOTA:
685 A multilevel parallel object-oriented framework for design optimization,
686 parameter estimation, uncertainty quantification, and sensitivity analysis,
687 version 5.3.1 reference manual. Tech. rep., U.S. Department of Energy.
- 688 Adams, B., Ebeida, M., Eldred, M., Jakeman, J., Swiler, L., Dalbey, K.,
689 Eddy, J., Hu, K., Vigil, D., Bauman, L., Hough, P., 2013b. DAKOTA:
690 A multilevel parallel object-oriented framework for design optimization,
691 parameter estimation, uncertainty quantification, and sensitivity analysis,
692 version 5.3.1 user's manual. Tech. rep., U.S. Department of Energy.
- 693 Aster, R. C., Borchers, B., Thurber, C. H., 2013. Parameter Estimation and
694 Inverse Problems. Academic Press, Amsterdam.
- 695 Banta, E. R., Hill, M. C., Poeter, E., Doherty, J. E., Babendreier, J., 2008.
696 Building model analysis applications with the Joint Universal Parameter
697 Identification and Evaluation of Reliability (JUPITER) API. *Computers*
698 *& Geosciences* 34 (4), 310–319.
- 699 Beven, K., 2006. A manifesto for the equifinality thesis. *Journal of Hydrology*
700 320 (1), 18–36.
- 701 Beven, K., Binley, A., 1992. The future of distributed models: model cali-
702 bration and uncertainty prediction. *Hydrological Processes* 6 (3), 279–298.
- 703 Beven, K., Freer, J., 2001. Equifinality, data assimilation, and uncertainty
704 estimation in mechanistic modelling of complex environmental systems
705 using the GLUE methodology. *Journal of Hydrology* 249 (1), 11–29.
- 706 Caers, J., 2011. Modeling Uncertainty in the Earth Sciences. Wiley-
707 Blackwell.
- 708 David, O., Markstrom, S. L., Rojas, K. W., Ahuja, L. R., Schneider, I. W.,
709 2002. The Object Modeling System. *Agricultural system models in field*
710 *research and technology transfer*, 317–331.
- 711 Doherty, J., Welter, D., 2010. A short exploration of structural noise. *Water*
712 *Resour. Res.* 46, 1–14.

- 713 Errico, R. M., 1997. What is an adjoint model? *Bulletin of the American*
714 *Meteorological Society* 78 (11), 2577–2591.
- 715 Foglia, L., Mehl, S. W., Hill, M. C., Burlando, P., 2013. Evaluating model
716 structure adequacy: The case of the Maggia Valley groundwater system,
717 southern Switzerland. *Water Resour. Res.* 49 (November 2012), 260–282.
- 718 Hill, C., DeLuca, C., Suarez, M., da Silva, A., et al., 2004. The architec-
719 ture of the Earth System Modeling Framework. *Computing in Science &*
720 *Engineering* 6 (1), 18–28.
- 721 Hill, M. C., Tiedeman, C. R., 2007. Effective groundwater model calibration:
722 with analysis of data, sensitivities, predictions, and uncertainty. *John Wi-*
723 *ley & Sons*.
- 724 Lampton, M., 1997. Damping-undamping strategies for the levenberg-
725 marquardt nonlinear least-squares problem. *Computers in Physics* 11 (1),
726 110–115.
- 727 Moré, J., 1977. The levenberg-marquardt algorithm: Implementation and
728 theory. In: *Numerical Analysis: Lecture Notes in Mathematics* 630.
729 Springer-Verlag, pp. 105–116.
- 730 Parker, R. L., 1984. The inverse problem of resistivity sounding. *Geophysics*
731 49, 2143–2158.
- 732 Peckham, S. D., 2009. Geomorphometry in RiverTools. In: Hengl, T., Reuter,
733 H. (Eds.), *Geomorphometry: Concepts, Software and Applications*. Vol. 33
734 of *Developments in Soil Science*. Elsevier, Ch. 18, pp. 411–430.
- 735 Peckham, S. D., 2015. Longitudinal elevation profiles of rivers: Curve fit-
736 ting with functions predicted by theory. In: Jasiewicz, J., Zwolinski, Z.,
737 Mitsova, H., Hengl, T. (Eds.), *Geomorphometry for Geosciences*. Inter-
738 national Society for Geomorphometry, pp. 137–140.
- 739 Peckham, S. D., Hutton, E. W., Norris, B., 2013. A component-based ap-
740 proach to integrated modeling in the geosciences: The design of CSDMS.
741 *Computers & Geosciences* 53, 3–12.

- 742 Poeter, E. P., Hill, M. C., Lu, D., Tiedeman, C. R., Mehl, S., 2014.
743 UCODE_2014, with new capabilities to define parameters unique to predic-
744 tions, calculate weights using simulated values, estimate parameters with
745 svd, evaluate uncertainty with MCMC, and more. Tech. rep., Integrated
746 Groundwater Modeling Center, Report Number GWMI 2014-02.
- 747 PSUADE, 2014. Problem solving environment for uncertainty analysis
748 and design exploration. [https://computation.llnl.gov/casc/uncertainty-](https://computation.llnl.gov/casc/uncertainty-quantification/)
749 [quantification/](https://computation.llnl.gov/casc/uncertainty-quantification/), [accessed 16 January 2015].
- 750 Renard, B., Kavetski, D., Leblois, E., Thyer, M., Kuczera, G., Franks, S. W.,
751 2011. Toward a reliable decomposition of predictive uncertainty in hydro-
752 logical modeling: Characterizing rainfall errors using conditional simula-
753 tion. *Water Resour. Res.* 47.
- 754 Rosenbrock, H. H., 1960. An automatic method for finding the greatest or
755 least value of a function. *The Computer Journal* 3, 175–184.
- 756 Syvitski, J. P. M., Hutton, E. W. H., Piper, M. D., Overeem, I., Kettner,
757 A. J., Peckham, S. D., 2014. Plug and play component modeling — the CS-
758 DMS 2.0 approach. In: Ames, D. P., Quinn, N. W. T., Rizzoli, A. E. (Eds.),
759 *Proceedings of the 7th Intl. Congress on Env. Modelling and Software*. In-
760 *ternational Environmental Modelling and Software Society (iEMSs)*, pp.
761 431–438.
- 762 Tarantola, A., 2005. *Inverse Problem Theory and Methods for Model Param-*
763 *eter Estimation*. Society for Industrial and Applied Mathematics, Philadel-
764 *phia*.
- 765 Tikhonov, A. N., 1963. Solution of incorrectly formulated problems and the
766 regularisation method. *Sov. Math. Dokl.* 4, 1035–1038.
- 767 Utke, J., Naumann, U., Lyons, A., 2014. *OpenAD/F: User Manual*. Tech.
768 *rep.*, U.S. Department of Energy, Argonne National Lab.
- 769 Whelan, G., Castleton, K., Buck, J., Hoopes, B., Pelton, M., Strenge, D.,
770 Gelston, G., Kickert, R., 1997. Concepts of a framework for risk analysis
771 in multimedia environmental systems. Pacific Northwest National Labora-
772 tory, Richland, Wash., Oct.

773 Whipple, K. X., Tucker, G. E., 1999. Dynamics of the stream-power river
774 incision model: Implications for height limits of mountain ranges, land-
775 scape response timescales, and research needs. *J. Geophys. Res.* 104 (B8),
776 17,661–17,674.