Computers & Geosciences 90 (2016) 152–161

Contents lists available at ScienceDirect

Computers & Geosciences

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

Case study

# 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, United States
[b] CEOAS, Oregon State University, Corvallis, OR 97331, United States
[c] Department of Geology, University of Kansas, Lawrence, KS 66045, United States

ARTICLE INFO

*Article history:*
Received 20 January 2015
Received in revised form
2 March 2016
Accepted 3 March 2016
Available online 4 March 2016

*Keywords:*
Model uncertainty
Modeling frameworks
Component-based modeling
Optimization
Inverse problems
Nonlinear least squares
Parameter estimation
Longitudinal river elevation profiles

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.



## 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

* Corresponding author.
*E-mail address:* Scott.Peckham@colorado.edu (S.D. Peckham).

http://dx.doi.org/10.1016/j.cageo.2016.03.005
0098-3004/

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 $y = c\,x^p$, 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. However, the models typically differ in many ways, such as their programming language, computational grid, time-stepping scheme, variable names and units. In addition to providing a simple mechanism for coupling models, modeling frameworks typically contain *service components* or *mediators* that automatically reconcile differences between the models that would otherwise prevent them from sharing variables. Examples of mediators include spatial regridders, time interpolators, unit converters and semantic mediators. These mediators and other capabilities of the framework – such as the ability to write composite model output to different file formats with standardized metadata, or to provide a graphical user interface (GUI) and help system – provide both model users and developers with significant added value.

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

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

## 3. Background: inverse modeling methods

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

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

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

### 3.1. Constructing an objective function

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

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

where $\mathbf{y}$ is a vector with components $y_k$ and $p > 0$ is a scalar. The case where $p=2$, or the $L^2$ norm, is the basis of the popular *least squares* metric. While this metric gives disproportionate weight to

outliers, it ensures that the derivative of the objective function is continuous where the error is zero.

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

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

### 3.2. Optimization methods

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

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

$$f(p_1, p_2) = (a - p_1)^2 + b(p_2 - p_1^2)^2. \tag{2}$$

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

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

#### 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 $f(x_1, \ldots, x_n)$ is the objective function of the design parameters, the gradient and Hessian matrix are

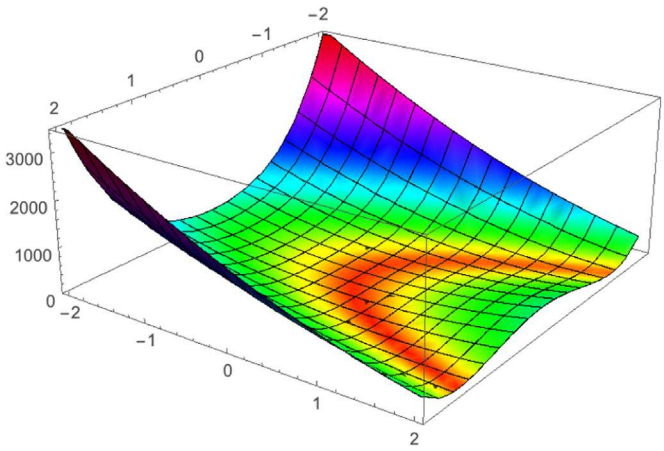



**Fig. 1.** The Rosenbrock function, a classic test problem in optimization.

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

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

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

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

The third type of differentiation is distinct from the first two classical types and is usually called *automatic differentiation*. It exploits the fact that every computer program, no matter how

complicated, performs calculations by combining elementary arithmetic operations (addition, subtraction, multiplication, division, powers, etc.) with evaluations of elementary functions (exp, log, sin, cos, etc.). By applying the chain rule to this sequence of operations it is possible to automatically compute derivatives of arbitrary order that are accurate to the machine's working precision while using only several times more operations than the original program. Many such tools have now reached maturity (e.g., OpenAD, Utke et al., 2014; see http://www.autodiff.org/ for a comprehensive list). With automatic differentiation, the cost of computing the gradient vector varies, but typically takes as much time as several (e.g. 2–12) forward model runs, which could be a great improvement over numerical differentiation for large model parameter spaces.

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

#### 3.2.2. *Derivative-free optimization*

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

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

## 4. General-purpose software packages for inverse modeling and uncertainty quantification

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

- DAKOTA, 2015 (Design Analysis Kit for Optimization and Terascale Applications) http://dakota.sandia.gov/software.html
- UCODE/Jupiter API, 2015 (Joint Universal Parameter IdenTification and Evaluation of Reliability, Poeter et al., 2014) http://igwmc.mines.edu/freeware/ucode/
- PSUADE, 2014 (Problem Solving environment for Uncertainty Analysis and Design Exploration, PSUADE, 2014) http://computation.llnl.gov/casc/uncertainty_quantification/
- PEST, 2015 (Model-Independent Parameter Estimation and Uncertainty Analysis) http://www.pesthomepage.org/Home.php
- Ostrich, 2008 (Optimization Software Toolkit) http://www.civil.uwaterloo.ca/lsmatott/Ostrich/OstrichMain.html
- TAO, 2014 (Toolkit for Advanced Optimization), with a focus on gradient-based search methods http://www.mcs.anl.gov/research/projects/tao/
- QUESO, 2014 (Quantification of Uncertainty for Estimation, Simulation and Optimization) https://red.ices.utexas.edu/projects/software/wiki/QUESO
- STEPS, 2012 (Stochastic Engine for Pathway Simulation) http://steps.sourceforge.net/STEPS/default.php

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

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

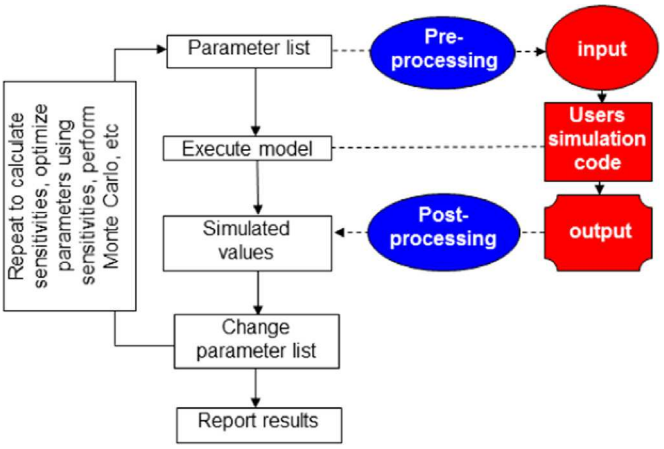



**Fig. 2.** Flowchart showing how an inverse modeling or uncertainty analysis application such as DAKOTA or UCODE (represented by blue and white boxes) typically interacts with a process model, such as those in CSDMS (red boxes). Modified from Banta et al. (2008). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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 (1999), can be written as

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

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

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

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

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

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

**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 |

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

| Optimization method (DAKOTA) | $\gamma$ | $R^\star$ | $R^2$ |
|---|---|---|---|
| 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 |

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 $p > 0$. These initially caused problems until the analytic gradient functions were 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. Fig. 3 shows the best fit of (4) to the Beaver Creek profile data, obtained with the NL2SOL method in DAKOTA. Fig. 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 detail. Sources of error can be classified into four main categories, namely (1) model inadequacy, (2) input data inadequacy, (3) model parameter and input data

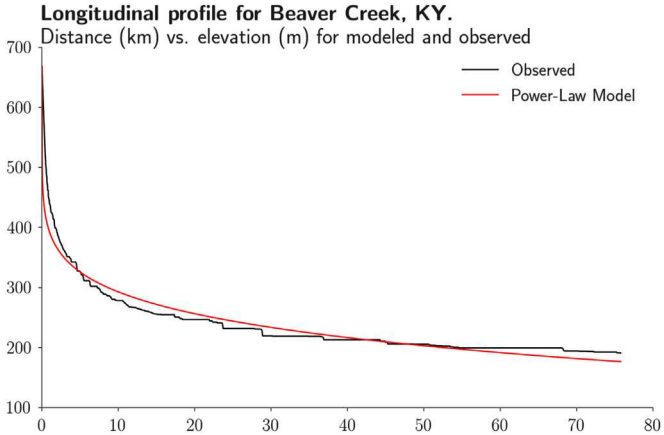



**Fig. 3.** Best fit of Whipple–Tucker (1999) model to Beaver Creek main channel profile, using NL2SOL algorithm in DAKOTA.

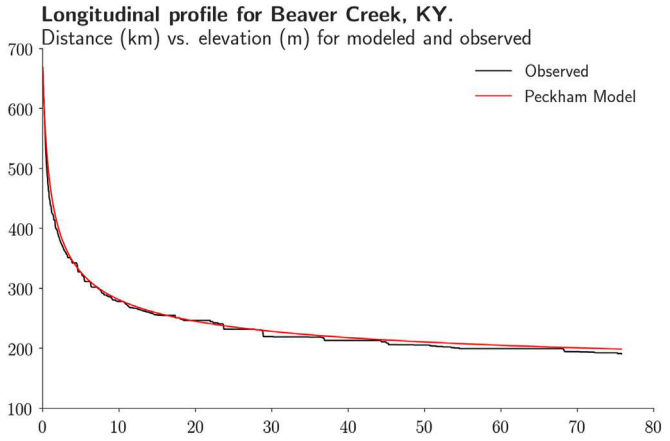



**Fig. 4.** Best fit of Peckham (2015) model to Beaver Creek main channel profile, using NL2SOL algorithm in DAKOTA.

uncertainty and (4) developer and user errors.

### 6.1. Model inadequacy

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

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

The main ways to reduce uncertainty due to model inadequacy are (1) scientific research to better understand physical processes and systems that are poorly understood, (2) careful selection of numerical methods and (3) various forms of testing. As for testing, there are five main things that models can be tested or evaluated against, namely:

- analytic solutions and test problems;
- measured data (i.e. observed vs. predicted);
- valid range or reasonableness (e.g. sanity tests);
- other models (especially for complex models, e.g. climate models);
- their former selves (e.g. regression and unit tests, often automated).

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

### 6.2. Input data inadequacy

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

- poor spatial or temporal resolution;
- poor quality;
- storage or transfer errors (e.g. byte order, data type, truncated files, formatting, etc.)

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

### 6.3. Model parameter and input data uncertainties

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

### 6.4. Developer and user errors

- incorrect implementation or developer error (e.g. bugs)
  - regressions (bugs due to updates or improvements);
  - bugs may be in software dependencies or in the model itself;
  - coordinate projection, sign convention, or units mismatch (and failure to convert);
  - problems at domain boundaries;
- preparation of input data (model setup);
- incorrect or unintended use (e.g. unawareness of limitations).

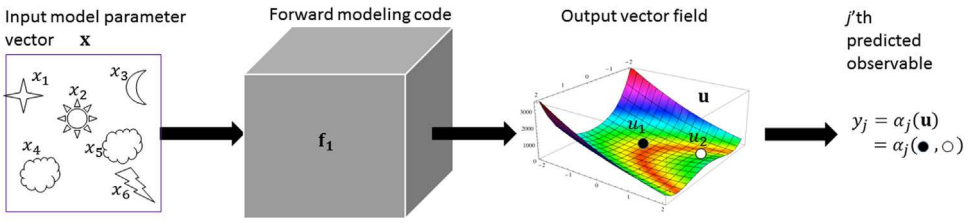
Developer error can be reduced through a variety of best software development practices, such as various types of testing (see above), version control, use of design principles such as *separation of concerns*, use of standards and frequent use. User errors can be addressed in a variety of ways, including:

- documentation (e.g. tech tips, FAQs, manuals, tutorials, context help);
- GUIs (that can restrict possible inputs based on context);
- software to check inputs, conditions, compatibility, etc.;
- training (and certification);
- supervision by an expert.

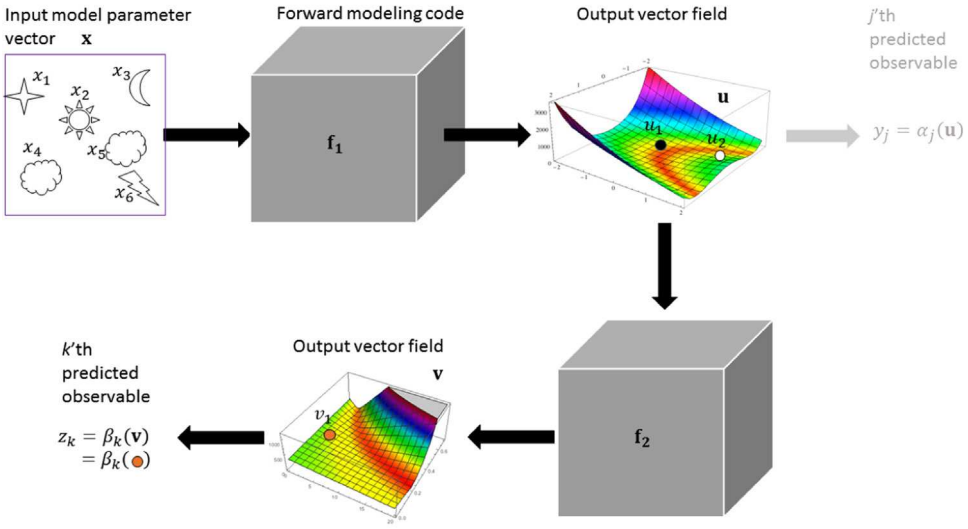
Most of these strategies would be straightforward to implement within a modeling framework.

## 7. Towards including inverse modeling and uncertainty quantification in a component-based modeling framework

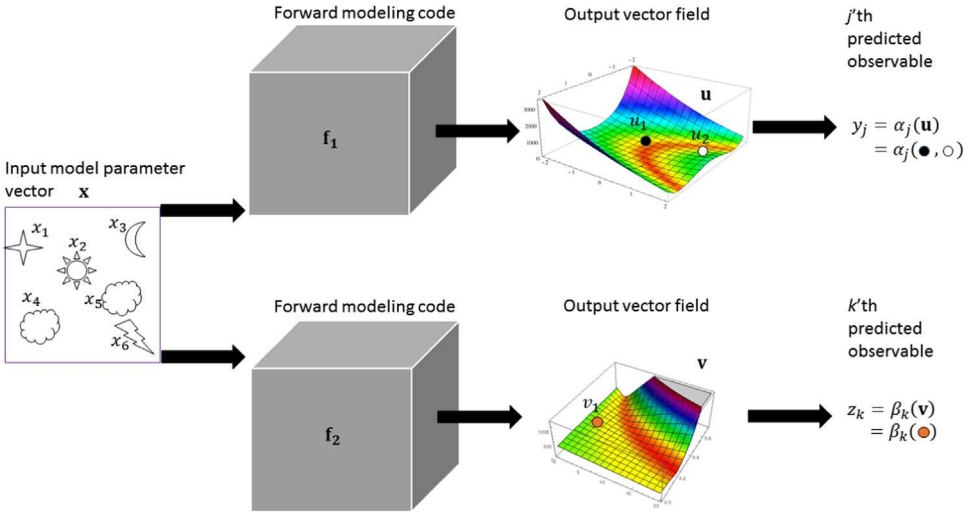
Inclusion of uncertainly quantification, parameter estimation and inversion tools in a modeling framework inevitably requires a certain




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




(b) Model coupling in series.




(c) Model coupling in parallel.

**Fig. 5.** Model coupling configurations in a component-based framework.

compromise on efficiency, but has the potential to democratize these techniques, providing Earth system modelers with fingertip access to model validation approaches, without the typically steep learning curve and software development requirements. Here however, we aim to show that in a smart framework, this efficiency compromise may not be a significant drawback. For this, we consider several common use case scenarios, illustrated in (5).

### 7.1. Stand-alone models

In a modeling framework with access to "uncertainty tools", a stand-alone computational model could immediately make use of external parameter estimation tools based on derivative-free optimization (Section 3.3.2). Derivative-based optimization methods (Section 3.3.1) could also be directly employed, using numerical differentiation to iteratively compute derivatives of the penalty

function. However, both types of methods are only applicable for models that have relatively few significant input model parameters that are free to vary. For models with a large model parameter space, adjoint techniques must be employed to compute penalty function derivatives for derivative-based optimization (see more details on the adjoints in Section 3.3.1), and to compute the data sensitivities (or the full Jacobian) needed by the uncertainly quantification techniques. These adjoint codes are most efficient if hand-coded, and the model developer could provide an adjoint for the system to use, if such a tool already exists. Users would then immediately enjoy the full variety of techniques available in the framework, while sacrificing little on the efficiency. For models that do not have an adjoint, the use of automatic differentiation could be considered. (It should be noted that an adjoint version of every generic interpolation routine supported by the framework would also need to be provided to enable this functionality.) Thus, the methods described here could be beneficial to modelers who are interested in using and/or testing out a variety of inversion techniques with minimal additional development investment. However, the true value of the framework-based approach to inversion becomes evident when inversion and/or parameter estimation for coupled models is considered. We now discuss two distinct use cases.

### 7.2. Models coupled in series

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

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

### 7.3. Models coupled in parallel

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

In the parallel model coupling scenario, the adjoint would be obtained by applying the two model adjoints, separately, to their respective data residuals, and the resultant variations in the input model parameters would be summed up to obtain the complete penalty function derivative. Again, this would be handled at the framework level, allowing the complicated model coupling scenario to be wrapped up for direct use in an external inversion routine.

### 7.4. New modeling framework functionality

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

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

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

## 8. Summary and recommendations

Computational models are a powerful means of understanding the Earth system, making predictions and guiding decisions. Too often, however, models are used without any analysis of their uncertainty. The integration of toolkits such as DAKOTA into component-based modeling frameworks will help to resolve this by drawing attention to the problem and providing easy access to powerful methods, thereby making it much easier for users to quantify, assess and understand the uncertainty in models. Background information on modeling frameworks and inverse modeling were provided in Sections 2 and 3 and a list of major software toolkits for uncertainty quantification and inverse modeling was given in Section 4. Since most of these toolkits use the same approach to interacting with models, it appears feasible for a modeling framework to provide access to more than one such package. Section 5 used an example surface process problem to illustrate some of the issues that are encountered in real applications and to give a taste of what toolkits like DAKOTA have to offer modelers. In Section 6 it was argued that the primary sources of

error in computational models can be usefully organized into four groups — model inadequacy, input data inadequacy, model parameter and input data uncertainty and developer and user errors — and that modeling frameworks could address many of them. Finally, several technical issues regarding the inclusion of inverse modeling in a model coupling framework were discussed in Section 7.

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

## Acknowledgments

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

## Appendix A. Best-fit parameters for a class of profile models

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

$$z(x) = z_0 - c\,H(p, x, x_0), \tag{6}$$

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

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

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

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

Fig. 6 shows a contour plot for this cost function over the $(c,p)$ plane for a particular set of measured elevation profile values, $(x_k, z_k)$ and

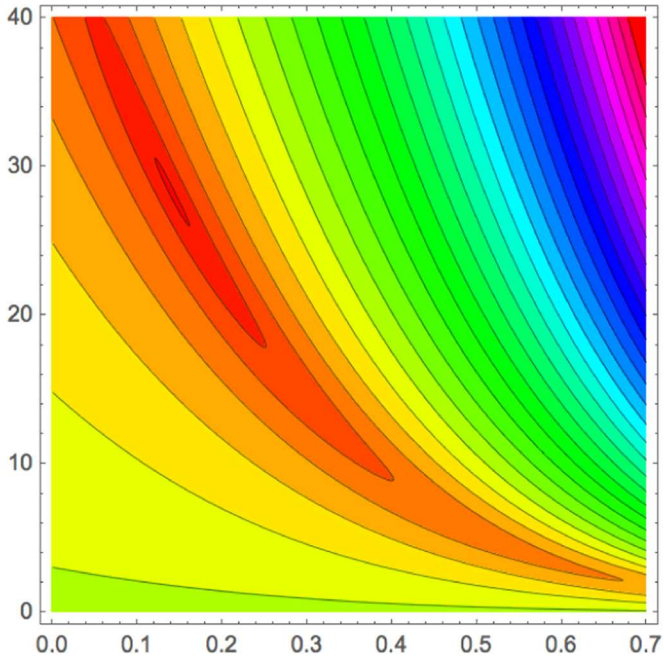



**Fig. 6.** Contour plot for a typical cost function when $f(x, p) = (1/p)x^p$, with extreme vertical exaggeration to show global minimum. For this example, $c_0 = 25$ and $p_0 = 0.166$.

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

The nature of this broad, flat valley can be understood geometrically. First, notice that for $n=1$, $E(c, p)$ plots as a valley in the $(c,p)$ plane that attains its minimum value (zero in this case) along the entire curve given by $c = p(z_0 - z1)/(x_1^p - x_0^p)$. This curve diverges at $p=0$ and rapidly decreases with increasing $p$. The position of the valley depends on $(x_0, z_0)$ and $(x_1, z_1)$, but there is no global minimum. The cost function (7) may be viewed as a sum of such valley surfaces, all offset somewhat from one another, which results in a surface with a global minimum that lies in a broad valley.

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

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

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

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

This is a curve in the $(c,p)$ plane along which $\partial E/\partial p = 0$. As shown in Fig. 7, the curves $c_1(p)$ and $c_2(p)$ are very close to one another, and close to the bottom of the valley, but they cross at a single point which gives the best-fit $(c,p)$ pair. Since the best-fit pair lies on the curve $c_1(p)$, we can insert $c = c_1(p)$ into the cost function (7)

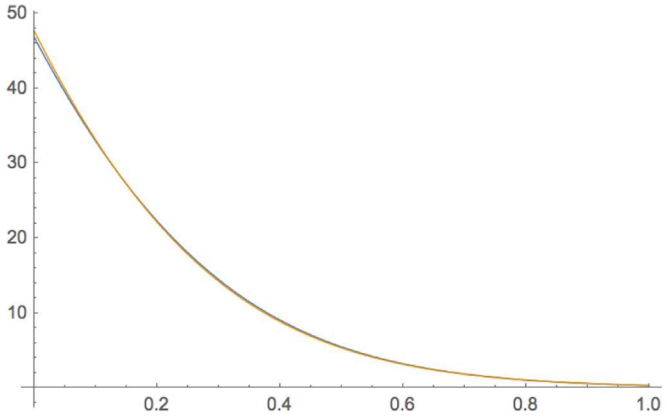



**Fig. 7.** The (difficult to distinguish) curves $c_1(p)$ (yellow) and $c_2(p)$ (blue) that cross at the best-fit $(c,p)$. For this example, they cross at $c=28.2491$, $p=0.14075$. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

to get a new cost function that depends only on $p$, namely $E(p) = E(c_1(p), p)$. After simplification, we find that

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

Notice the power of 2 in the numerator of the last term, not present in (8). For both $f(p, x) = (1/p)x^p$ and $f(p, x) = x^p$, this simplifies further to

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

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

## References


Adams, B., Ebeida, M., Eldred, M., Jakeman, J., Swiler, L., Bohnhoff, W., Dalbey, K., Eddy, J., Hu, K., Vigil, D., Bauman, L., 2013a. DAKOTA: a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis, version 5.3.1 reference manual. Technical Report, U.S. Department of Energy.

Adams, B., Ebeida, M., Eldred, M., Jakeman, J., Swiler, L., Dalbey, K., Eddy, J., Hu, K., Vigil, D., Bauman, L., Hough, P., 2013b. DAKOTA: a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis, version 5.3.1 user's manual. Technical Report, U.S. Department of Energy.

Aster, R.C., Borchers, B., Thurber, C.H., 2013. Parameter Estimation and Inverse Problems. Academic Press, Amsterdam.

Banta, E.R., Hill, M.C., Poeter, E., Doherty, J.E., Babendreier, J., 2008. Building model analysis applications with the Joint Universal Parameter IdenTification and Evaluation of Reliability (JUPITER) API. Comput. Geosci. 34 (4), 310–319.

Beven, K., 2006. A manifesto for the equifinality thesis. J. Hydrol. 320 (1), 18–36.

Beven, K., Binley, A., 1992. The future of distributed models: model calibration and uncertainty prediction. Hydrol. Process. 6 (3), 279–298.

Beven, K., Freer, J., 2001. Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the GLUE methodology. J. Hydrol. 249 (1), 11–29.

Caers, J., 2011. Modeling Uncertainty in the Earth Sciences. Wiley-Blackwell, Hoboken, NJ.

David, O., Markstrom, S.L., Rojas, K.W., Ahuja, L.R., Schneider, I.W., 2002. The Object Modeling System. Agricultural System Models in Field Research and Technology Transfer, pp. 317–331.

Doherty, J., Welter, D., 2010. A short exploration of structural noise. Water Resour. Res. 46, 1–14.

Errico, R.M., 1997. What is an adjoint model? Bull. Am. Meteorol. Soc. 78 (11), 2577–2591.

Foglia, L., Mehl, S.W., Hill, M.C., Burlando, P., 2013. Evaluating model structure adequacy: the case of the Maggia Valley groundwater system, southern Switzerland. Water Resour. Res. 49 (November (2012)), 260–282.

Hill, C., DeLuca, C., Suarez, M., da Silva, A., et al., 2004. The architecture of the Earth System Modeling Framework. Comput. Sci. Eng. 6 (1), 18–28.

Hill, M.C., Tiedeman, C.R., 2007. Effective Groundwater Model Calibration: With Analysis of Data, Sensitivities, Predictions, and Uncertainty. John Wiley & Sons, Hoboken, NJ.

Lampton, M., 1997. Damping–undamping strategies for the Levenberg–Marquardt nonlinear least-squares problem. Comput. Phys. 11 (1), 110–115.

Moré, J., 1977. The Levenberg–Marquardt algorithm: implementation and theory. In: Numerical Analysis: Lecture Notes in Mathematics, vol. 630. Springer-Verlag, Berlin, Germany, pp. 105–116.

Parker, R.L., 1984. The inverse problem of resistivity sounding. Geophysics 49, 2143–2158.

Peckham, S.D., 2009. Geomorphometry in RiverTools. In: Hengl, T., Reuter, H. (Eds.), Geomorphometry: Concepts, Software and Applications, Developments in Soil Science vol. 33. Elsevier, pp. 411–430 (Chapter 18).

Peckham, S.D., 2015. Longitudinal elevation profiles of rivers: curve fitting with functions predicted by theory. In: Jasiewicz, J., Zwolinkski, Z., Mitasova, H., Hengl, T. (Eds.), Geomorphometry for Geosciences. International Society for Geomorphometry, Poznan, Poland, pp. 137–140.

Peckham, S.D., Hutton, E.W., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12.

Poeter, E.P., Hill, M.C., Lu, D., Tiedeman, C.R., Mehl, S., 2014. UCODE_2014, with new capabilities to define parameters unique to predictions, calculate weights using simulated values, estimate parameters with SVD, evaluate uncertainty with MCMC, and more. Technical Report, Integrated Groundwater Modeling Center, Report Number GWMI 2014-02.

PSUADE, 2014. Problem solving environment for uncertainty analysis and design exploration. ⟨https://computation.llnl.gov/casc/uncertainty-quantification/⟩ (accessed 16.01.2015).

Renard, B., Kavetski, D., Leblois, E., Thyer, M., Kuczera, G., Franks, S.W., 2011. Toward a reliable decomposition of predictive uncertainty in hydrological modeling: characterizing rainfall errors using conditional simulation. Water Resour. Res. 47.

Rosenbrock, H.H., 1960. An automatic method for finding the greatest or least value of a function. Comput. J. 3, 175–184.

Syvitski, J.P.M., Hutton, E.W.H., Piper, M.D., Overeem, I., Kettner, A.J., Peckham, S.D., 2014. Plug and play component modeling — the CSDMS 2.0 approach. In: Ames, D.P., Quinn, N.W.T., Rizzoli, A.E. (Eds.), Proceedings of the 7th International Congress on Environmental Modelling and Software. International Environmental Modelling and Software Society (iEMSs), San Diego, CA, pp. 431–438.

Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. Society for Industrial and Applied Mathematics, Philadelphia.

Tikhonov, A.N., 1963. Solution of incorrectly formulated problems and the regularisation method. Sov. Math. Dokl. 4, 1035–1038.

Utke, J., Naumann, U., Lyons, A., 2014. OpenAD/F: user manual. Technical Report, U. S. Department of Energy, Argonne National Lab.

Whelan, G., Castleton, K., Buck, J., Hoopes, B., Pelton, M., Strenge, D., Gelston, G., Kickert, R., 1997. Concepts of a framework for risk analysis in multimedia environmental systems. Pacific Northwest National Laboratory, Richland, Wash. (October).

Whipple, K.X., Tucker, G.E., 1999. Dynamics of the stream-power river incision model: implications for height limits of mountain ranges, landscape response timescales, and research needs. J. Geophys. Res. 104 (B8), 17661–17674.