LA-UR-17-28326
Approved for public release; distribution is unlimited.

Title: An Integrated Approach to Parameter Learning in Infinite-Dimensional Space

Author(s): Boyd, Zachary M.
Wendelberger, Joanne Roth

Intended for: Report

Issued: 2017-09-14

An Integrated Approach to Parameter Learning in Infinite-Dimensional Space

Zachary M. Boyd and Joanne R. Wendelberger

September 13, 2017

Abstract

The availability of sophisticated modern physics codes has greatly extended the ability of domain scientists to understand the processes underlying their observations of complicated processes, but it has also introduced the curse of dimensionality via the many user-set parameters available to tune. Many of these parameters are naturally expressed as functional data, such as initial temperature distributions, equations of state, and controls. Thus, when attempting to find parameters that match observed data, being able to navigate parameter-space becomes highly non-trivial, especially considering that accurate simulations can be expensive both in terms of time and money. Existing solutions include batch-parallel simulations, high-dimensional, derivative-free optimization, and expert guessing, all of which make some contribution to solving the problem but do not completely resolve the issue.

In this work, we explore the possibility of coupling together all three of the techniques just described by designing user-guided, batch-parallel optimization schemes. Our motivating example is a neutron diffusion partial differential equation where the time-varying multiplication factor serves as the unknown control parameter to be learned. We find that a simple, batch-parallelizable, random-walk scheme is able to make some progress on the problem but does not by itself produce satisfactory results. After reducing the dimensionality of the problem using functional principal component analysis (fPCA), we are able to track the progress of the solver in a visually simple way as well as viewing the associated principle components. This allows a human to make reasonable guesses about which points in the state space the random walker should try next. Thus, by combining the random walker's ability to find descent directions with the human's understanding of the underlying physics, it is possible to use expensive simulations more efficiently and more quickly arrive at the desired parameter set.

1 Three existing techniques

Parameter search problems can be tackled using up to three techniques, which we propose to combine.

First, parameter searches in computer simulation are often amenable to *batch parallel* approaches, in which many simulations are executed at the same time using multi-processor computing. The gains from parallel computing are almost perfect in this context, since there is generally no dependency at all between the different runs. The weakness of relying on batch parallelism alone is that it is susceptible to the curse of dimensionality, as the number of grid points required for a thorough search grows exponentially with the dimension of the problem. This means that some judicious approach is needed when selecting points at which to run simulations. The next two techniques help with that problem.

Second, parameter search can be conducted using an automated optimizer. When working with a complicated simulation package and trying to find parameters that reproduce observed data approximately, we are essentially working with an optimization problem where we attempt to minimize the difference between the simulated solution and the observed data. The problem will be made precise below, but we can already make some important observations regarding the nature of the task.

First observation: There is no particular reason to believe that the problem is convex, since the relationship between input parameters and simulation output is complicated, which was the very reason that simulations were invoked. Therefore, local search methods will not be adequate. In fact, one can prove that unless some additional assumption is made about the structure of the problem, any algorithm that can successfully optimize arbitrary continuous objective functions must have a search pattern that is dense in the parameter space. Usually, however, we do not have good guarantees about structure, such as Lipschitz continuity, that might apply to the problem.

Second observation: There is, in general, no gradient information available, and getting approximate gradients requires $O(N)$ simulations, where N is the dimension of the search space. This means that many traditional methods, such as gradient descent, Newton's method, ADMM, etc. are all impractical. Essentially the only option available is to evaluate the simulation for different input parameters and make inferences from that about the global structure of the optimization problem.

Third observation: We are in very high dimensions (infinite dimensions, since some of the input parameters are functions), so an exhaustive search is out of the question. For instance, if we were to check each point on a grid discretization of the parameter space, the number of grid points grows exponentially with the dimension of that space.

These three observations make the problem tremendously challenging, and even the state of the art optimization schemes are not able to deal with such problems effectively. For instance, in a recent survey of available techniques, the authors of [6] tested 22 different software packages against a large number of problems in dimensions between 30 and 300 and found that most solvers were successful less than 5% of the time at finding a global minimizer to within 1% error, with the best solver tested solving about 33% of the problems within the first 10 attempts, each of which was allowed 2,500 simulations. Since some of these problems were convex, the picture for nonconvex problems does not seem very sunny.

Finally, the third technique available in parameter searches is to make use of a human domain expert. This is useful because, in practice, many parameter combinations don't make a lot of sense. For instance, for the motivating problem in this paper, based on the data in fig. 2, a human expert would immediately conclude that the cause of the steep increase at the end was a high rate of neutron production, so the multiplication factor must be roughly increasing as a function of time. This immediately eliminates large regions in the parameter space from consideration. The expert can also select a more plausible initialization for an automated optimization scheme than a computer could without extra information.

2 The model problem

Our model problem is a one-dimensional neutron diffusion partial differential equation (PDE) described by

$$\phi_t = \phi_{xx} + (K(t) - 1)\phi. \tag{1}$$

Here ϕ is the neutron scalar flux, as a function of space and time. It is proportional to the neutron population. The subscripts t and xx denote the first time derivative and the second space derivative, respectively. K is a multiplication factor. When $K(t) > 1$, the term $(K(t) - 1)\phi$ leads to an overall increase in ϕ, and when $K(t) < 1$, ϕ should be decreasing overall. $K(t) = 1$ corresponds to a state of equilibrium. We allow K to depend on time to allow some means of controlling ϕ dynamically. The term ϕ_{xx} leads to a diffusion, or spreading out, of ϕ, so that its value will tend to become more uniform over time. We seek a solution for $0 < x < 1$ and $0 < t < 1$ and impose the boundary conditions

$$\phi(x = 0, t) = \phi(x = 1, t) = 0 \tag{2}$$

and the initial condition

$$\phi(x, t = 0) = x. \tag{3}$$

In a real physics simulation, there would be additional dimensional constants or functions incorporated into eq. (1), which we have omitted for simplicity of presentation and experiments.While this PDE is linear, the dependence of the solution on K is nonlinear, so the problem of determining K given ϕ is nontrivial.

Equation (1) is solved using a simple explicit, finite difference scheme, implemented in only a few lines of **R**. The resolution was not very fine (e.g. twenty grid points and a time step that respects the stability restriction for explicit diffusion schemes), which introduces some error into the solution. We consider this a good thing, since all real simulations will have error of this kind, and we are interested only in techniques that are robust to error.

All of our approaches were tested against a "ground truth" solution, which involved generating K and then solving for ϕ as a function of r and t. The goal was then to reconstruct K based only on a knowledge of ϕ, or even selected parts of ϕ, such as its value at $t = 1$. This is invariably done by selecting values of K, running the simulation, and comparing the output ϕ to the ground truth ϕ. When the two match, it is presumed that the value of K chosen by the user is approximately the same as the ground truth K. We used the L^2-norm difference between the computed ϕ and the ground-truth ϕ as the objective function to be minimized, sometimes with the addition of a regularizer on the curvature of K, i.e.

$$\min_K \int_0^1 \int_0^1 (\phi - \phi_{\text{gt}})^2 \mathrm{d}\, x \mathrm{d}\, t + \lambda \int_0^1 (K''(t))^2 \mathrm{d}\, t \tag{4}$$

where $\lambda \geq 0$ is a constant (usually set to zero).

3 Representation of K

In any computer implementation, it will be necessary to represent K in some finite way. A standard method to do so is using a system of basis functions, so that K is approximated as

$$K(t) \approx \sum_{i=0}^{N} \alpha_i b_i(t), \tag{5}$$

where the α_i are constants to be chosen, and b_i are fixed basis functions. Common choices for b_i include

- monomials: $b_i(t) = t^i$,
- sine waves: $b_i(t) = \sin\left(\frac{it}{2\pi}\right)$, and
- B-spline basis functions, in which case the representation is less straightforward.

These threee sets of basis functions correspond to function spaces containing

- polynomials,
- Fourier series (of odd functions), and
- B-splines.

In our experiments, we used a monomial or B-spline basis. The polynomials were up to quadratic, and the splines ranged from 13 to 400 coefficients. For a given representation of K, we selected a default set of coefficients, which we used to construct the coefficients for the ground truth K by adding normally-distributed noise with mean zero and standard deviation equal to the sample standard deviation of the default coefficients. This allowed us to initialize our solvers at the default values, while controlling how close the initialization was to the ground truth. The ground truth K produced by this method was fairly irregular, as illustrated in fig. 1. We also considered cases where the standard deviation of the noise was reduced somewhat and found that the solvers performed better when they were initialized closer to the true answer. The default coefficients for the monomial case were $(1, 80, 0.1)$, corresponding to the polynomial $1 + 80t + 0.1t^2$, and for the spline case they were the coefficients that arose from fitting a discretization of $80\sin(x) + 1$, so that the coefficients could have a similar interpretation depending on the number of basis functions being used. These defaults were chosen because they yielded interesting ground-truth solutions, featuring an initial drop in ϕ, followed by a rapid buildup towards the end of the simulation, but only in the right-hand side of the interval. The solutions thus incorporated several scales on ϕ for added challenge. An example ground truth ϕ can be seen in fig. 2

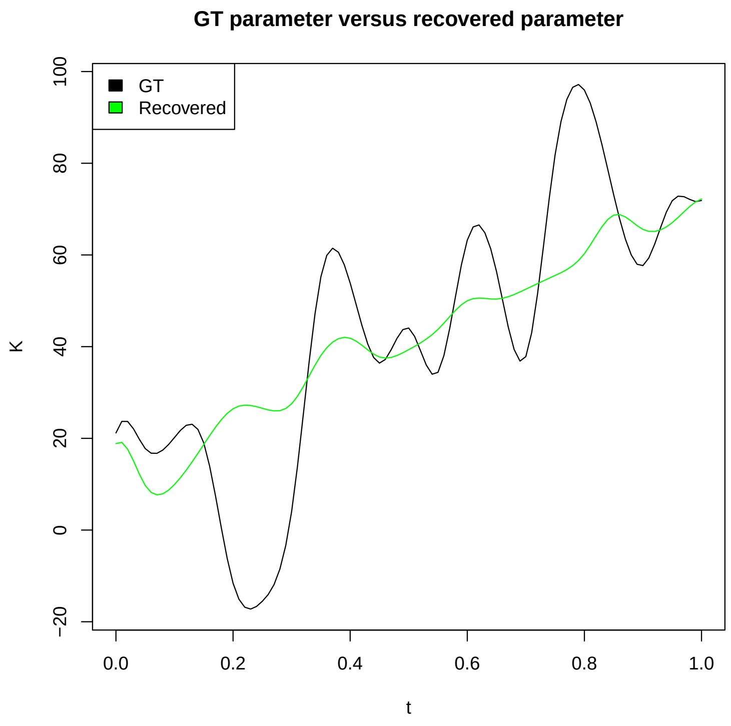


Figure 1: Example plot of a ground truth K used in our experiments, together with the curve recovered using a descent-based random walk solver.

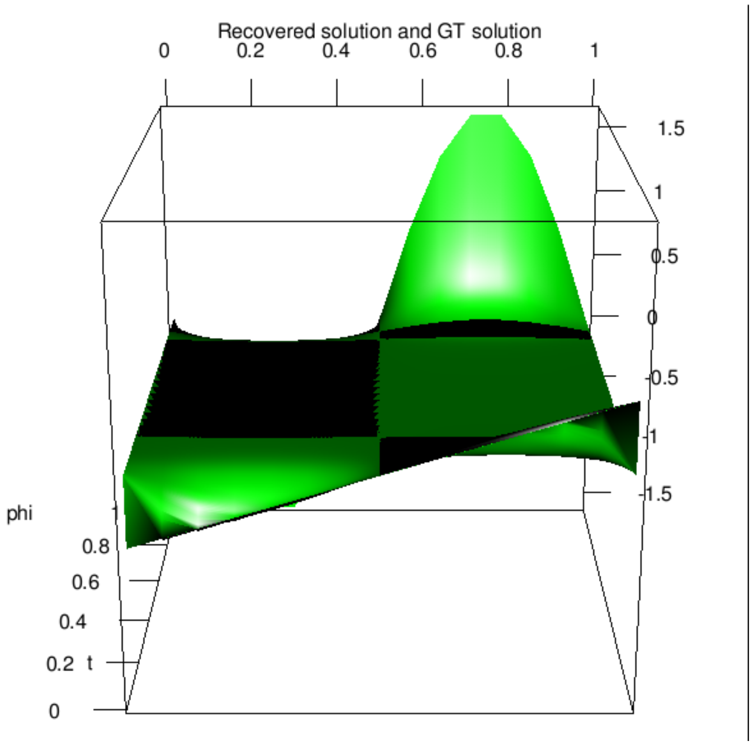


Figure 2: Example plot of a ground truth ϕ used in our experiments, together with the curve recovered using a descent-based random walk solver. The green surface is the ground truth, and the black surface is the evolution recovered by the solver. Plot generated with **R**'s RGL package [1].

4 Results of three automated solvers

Three derivative-free optimizers were used to see if the problem could be solved without human intervention. The following are brief descriptions of them

- *Random walk* A simple random walk scheme where, for any given value of the coefficients, a coefficient was chosen at random and tentatively increased or decreased. The modified set of coefficients was then used to generate a new value of the objective function. If improvement over the previous objective value was found, then the tentative change was made permanent; otherwise the change was reverted. A constant (user-selected) step size or a step size proportional to one over the number of iterations was used. Between 100 and 1,000 steps were allowed.

- *Simulated annealing* The GenSA package in **R** was used [7]. The main drawback of this package in our context was that it attempted to compute a discrete gradient to set the cooling schedule, which was prohibitively expensive in our context.

- *Nelder-Mead* The neldermead package in **R** was used [2]. As a local descent method, this was not able to recover the ground truth K. In addition, the overhead of requiring one evaluation for each dimension just to establish a base simplex proved burdensome as the dimensionality increased.

Overall, each of these solvers succeeded at making some improvement to on the initial conditions for some problems, but they generally did not converge to a global optimum. Of these, random walk and simulated annealing are the most amenable to batch parallel approaches, either by stepping different directions from the current optimum simultaneously or by seeding several random walks at different places.

5 fPCA representation

Given a sequence of proposed functions $K_1, \cdots, K_n$, perhaps generated by one of the optimization schemes, we can perform fPCA to extract the two functions κ_1, and κ_2, which are the two functions which capture the most variance among the K_i, along with the scores a_{ij}, $i \in 1, \cdots, n$, $j \in 1, 2$ which allow an optimal approximation $K_i \approx a_{i1}\kappa_1 + a_{i2}\kappa_2$. Then, plotting these scores as points in the plane, yields a visual representation of the path which the solver has traversed in the parameter space, see fig. 3. In general, it is possible to incorporate more than just the first two components in the visualization, in which case multiple plots are helpful. The components are ordered by significance, and we have found that in practice, the first two or three components usually contain most of the significant information about the search path. The iterates of K are shown in fig. 4a, and the first principal component, which captures most of the variability in the iterates, is shown in fig. 4b. All of our fPCA computations were performed using the FDA package in **R** [5].

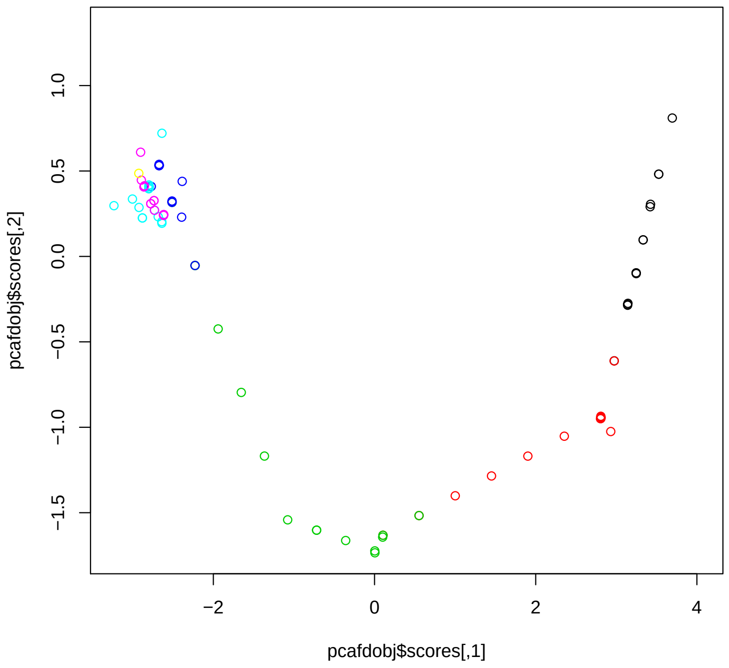
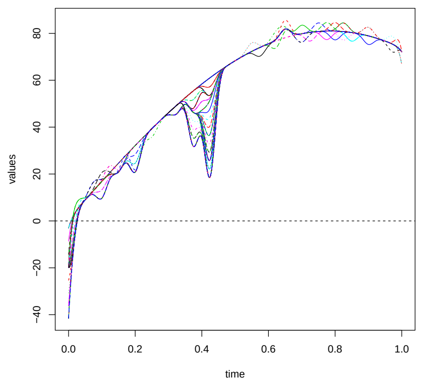
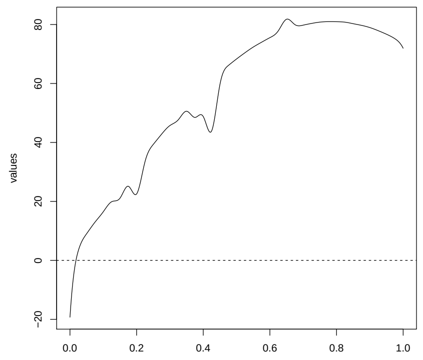


Figure 3: Example plot of the scores for a run of the random walk solver. The colors run from black to magenta as the solver progresses.



(a) Example plot of all the iterates of K that were tried by the solver.



(b) Observe how the first principal component captures most of the deviation in the iterates. The x-axis here is time.

6 Guided Exploration

The fPCA representation described in section 5 provides a valuable tool for further exploration of the functional parameter space. By examining the points along the fPCA path, a human expert can make decisions about how to proceed with the solver optimization process. This can include human-selected points or further exploration of the space using a statistical experiment design approach to investigate the 2-dimensional fPCA space and characterize the behavior of the error surface in the region around the set of points where the optimizer appears to be converging. Alternatively, other parts of the fPCA space could be explored, either manually or using statistical experiment design techniques, to judiciously select candidates for constructing new K function start values that can then be run through the solver of choice.

The guided exploration process uses information from the solver searches to produce a low-dimensional representation of the search path in fPCA space. This path can provide insights to the human expert about the progress of the simulation and clues about where to search next. Simple statistical experiment designs such as full factorials, possibly supplemented with axial points, can be overlaid on the fPCA space to form the basis for additional searches either in the local area of the current solution path endpoint or to start new searches in other areas of the functional parameter space. While 2 fPCA components may explain a sufficient portion of the variability, a larger number of components can be used by extending the statistical designs to 3 or more dimensions, possibly using fractional factorial designs to control the resulting number of candidate points.[1]

The proposed approach allows the user to explore the progress of a solver for a problem with an infinite dimensional functional input while working in a manageable low-dimensional space. Thus, the user is able to design new functional inputs to try to further improve the solution by using the coefficients associated with selected points in the fPCA space to construct new input functions using the specified basis representation. By incorporating guided exploration using human interaction and statistical experiment design, perhaps the dismal performance of solvers described in [6] can be improved upon.

References

[1] Adler, D., Murdoch, D., and others. *rgl: 3D Visualization Using OpenGL*, 2017. R package version 0.98.1.

[1]In 2-space, a full factorial is just the corner points of a rectangle around a specified center point. Sometimes the center point is also included (if not already run), and sometimes axial points from the center emanating out through the center points of the 4 edges. This extends naturally to higher dimensions. To conserve runs, higher dimensional searches in fPCA space could use fractional factorial designs which exploit the geometry of higher dimensional rectangles to estimate low order Taylor Series approximations using only a subset of the points in the multidimensional design region.

[2] Bihorel, S., and Baudin, M. *neldermead: R port of the Scilab neldermead module*, 2015. R package version 1.0-10.

[3] Jones, D. R. A taxonomy of global optimization methods based on response surfaces. *J. of Global Optimization 21*, 4 (Dec. 2001), 345–383.

[4] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.

[5] Ramsay, J. O., Wickham, H., Graves, S., and Hooker, G. *fda: Functional Data Analysis*, 2014. R package version 2.4.4.

[6] Rios, L. M., and Sahinidis, N. V. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization 56*, 3 (Jul 2013), 1247–1293.

[7] Yang Xiang, Gubian, S., Suomela, B., and Hoeng, J. Generalized simulated annealing for efficient global optimization: the GenSA package for R. *The R Journal Volume 5/1, June 2013* (2013).