OPTIMIZING NOISY, HIGH-DIMENSIONAL FUNCTIONS

JORDAN R. HALL

Abstract. We consider a noisy map f from a high-dimensional Λ to $\mathcal{D}$ of specified dimension, where the gradient of f exists, but may be inaccessible. We apply Derivative-Free Optimization (DFO) schemes as in [2]. The effectiveness of the optimization algorithm may be enhanced by performing dimension reduction in Λ as in [3, 6] and using machine learning to identify needed parameters.

Contents

Introduction

In this section we provide a literature review of inverse problem theory, derivative-free (DF) optimization, and dimension reduction. In the process, we will build a theoretical framework upon which we will pose research questions, state initial results, and form a research plan.

We begin our discussion by defining a parameter space Λ of dimension N, a map or "model" f, and a data space $\mathcal{D}$. We assume that $\mathcal{D}$ has dimension M, typically with $M \leq N$. Points in $\mathcal{D}$ may be known values of $f(\lambda), \lambda \in \Lambda$; as such, we may write $d = f(\lambda)$ to denote the particular datum corresponding to the evaluation of a point $\lambda \in \Lambda$ under f. Points in $\mathcal{D}$ may also be *observed* data, denoted d_{obs}, where the corresponding $\lambda \in \Lambda$ may be unknown. We allow for realizations of f to be noisy. Hence, we may model draws of f with $\hat{f}(\lambda) = f(\lambda) + \epsilon$, which is an additive noise structure, or $\hat{f}(\lambda) = f(\lambda)(1 + \epsilon)$, which is a multiplicative noise structure.

Derivative-Free Optimization Many important physical systems possess turbulent or chaotic behavior. The physical state of the system $u(x, \lambda)$ and the corresponding parameter to observable map $f(u(x, \lambda))$ may be modeled as a stochastic process, or as a deterministic function with additive or multiplicative noise. In this setting, the efficient extraction of accurate gradients of f in parameter space is a challenging undertaking, as popular techniques based on linearization, including adjoint methods, are inaccurate [**?**, **?**]. The finite-difference approximations to ∇f_Λ involve $N = \dim\Lambda$ additional, usually nonlinear model solves for the physical system state $u(x, \lambda_i + \delta\lambda_i)$, and is greatly polluted by the noise in f.

As a consequence of these difficulties, the approximate data-consistent solution of a SIP in this setting by optimization techniques will need algorithms that do not require gradient information from f. We consider derivative-free optimization (DFO) algorithms suited for additive and multiplicative noise as in [2]. This technique requires nothing more than evaluations of the noisy model and random draws from a normal distribution. Briefly, this method finds a new iterate by randomly perturbing the previous iterate in Λ; iterates are not allowed to stray much, though, due to relatively small smoothing factors and step sizes. The smoothing factor and step size in the DF algorithms are of great importance to their convergence and termination. As in [2], both the smoothing factor and step size will depend on a scale factor of the L_1 Lipschitz constant of f. As such, it will be of interest to obtain estimates of L_1, which is not straightforward in a gradient-free setting. We refer to [1, 5] for Lipschitz constant learning in this setting, and discuss this problem more in the proceeding sections.

In detail, as in [2], we consider the problem

$$\min_{\lambda \in \mathbb{R}^N} \quad \mathbb{E}\left[f(\lambda) + \nu(\lambda; \epsilon)\right], \tag{1}$$

where the authors assume that:

(i.) $f : \mathbb{R}^N \to \mathbb{R}$ is convex;
(ii.) ϵ is a random variable with probability density $P(\epsilon)$;
(iii.) for all λ the additive noise model ν is independent and identically distributed, has bounded variance σ_a^2, and is unbiased; i.e., $\mathbb{E}_\epsilon(\nu(\lambda; \epsilon)) = 0$.

The use of the $\mathbb{E}(\cdot)$ expected value notation is a mathematically precise way of expressing the fact that the authors view function evaluations as a random draw, in the sense that true function values $f(\lambda)$ are perturbed by a random vector in an additive or multiplicative fashion.

Chen and Wild propose the *STARS (STep-size Approximation in Randomized Search)* algorithm, which: uses small perturbations in the domain $\Lambda = \mathbb{R}^N$ given by the addition of a random

vector with components drawn from a normal distribution; computes the function value at the randomly perturbed point with additive or multiplicative noise drawn from a specified distribution; and updates iterates using a Gaussian-smoothed finite-difference scheme for approximate gradient information in a gradient descent scheme. We will examine the STARS algorithm in closer detail in the following sections. For now, we point out that the algorithm requires the ability to evaluate the model, and needs access to random draws from a normal distribution and the distribution from which noise is drawn. All in all, the algorithm can be implemented in about 10 lines of code in `python`.

Dimension Reduction We consider functions from a high-dimensional space Λ to a data space of smaller dimension. Many functions of interest actually represent postprocessed quantities from the solution of complex physical models. It is not often the case that every parameter has an equal impact on function values; usually some parameters matter more than others. If it is possible to mimic the response of f by processing fewer parameters, we can expect computational savings in many of the problems (i.e., minimization and forward problems) considered here.

In our setting, it may be advantageous to perform dimension reduction on f. In particular, we shall consider Active Subspace methods described by Paul Constantine in [3] and an equivalent method by T.M. Russi in [6]. These techniques seek to explain outputs $f(\Lambda)$ in a subspace $\mathcal{A} \subset \Lambda$ for which the $\dim(\mathcal{A}) < N$. Here we discuss the theoretical formulation of $\mathcal{A}$, but also how it may be found in practice.

We begin by noting $\nabla f(\lambda) \in \Lambda$ is a column vector of the same dimension as inputs λ with rows containing the N partial derivatives of f, which for this discussion we assume exist, and are square integrable in Λ equipped with some probability density that is positive everywhere in Λ and 0 otherwise; for instance, we could consider $\pi_\Lambda^{\text{prior}}(\lambda)$, the density describing our prior state of knowledge, which we choose to abbreviate as π_Λ. In classical active subspace analysis, one transforms inputs λ to the origin with some fixed variance, typically so that $\lambda \in [-1, 1]^N$. Then, as in [4], we may write

$$W = \int_\Lambda \nabla f(\lambda) \nabla f(\lambda)^\top \pi_\Lambda(\lambda) d\lambda, \tag{2}$$

which is an $N \times N$ symmetric positive semi-definite matrix which defines a certain covariance structure of ∇f over Λ. This interpretation of (2) leads one to the idea of computing the Singular Value Decomposition of W,

$$W = U \Sigma V^*, \tag{3}$$

where U is $N \times N$ unitary, Σ is $N \times N$ diagonal with the singular values of W along its diagonal, and V^* is $N \times N$ unitary. With the singular values of W in hand, we search for a drop-off in the spectrum of W. In detail, we plot the singular values, $\{\sigma_i\}_{i=1}^n$ and seek a drop-off in magnitude between some pair of singular values, σ_j and σ_{j+1}. The active subspace is the span of $u_1, \ldots, u_j$, which are the first j columns of U, the left singular vectors of W.

The fact that $u_1, \ldots, u_j$ correspond to the nontrivial singular values is exactly why they account for the most amount of variance in function values. In fact, one can view active subspace analysis as an artful choice of principal components after a *full* Principal Component Analysis (PCA) is performed in the gradient space W; for more details on this viewpoint, we refer the interested reader to Section 6.4 in [6].

For a point $\lambda \in \Lambda$, we define

$$\mathcal{P}_{\mathcal{A}}(\lambda) = \sum_{i=1}^{j} \left(u_i^T \lambda\right) u_i \in \mathcal{A}, \tag{4}$$

which is a projection of the point λ in the active directions of f. We call this projection an *active variable*, which is a point in the active subspace $\mathcal{A}$. We have arrived at the property that

$$f\left(\mathcal{P}_{\mathcal{A}}(\lambda)\right) \approx f(\lambda). \tag{5}$$

In practice, finding an active subspace of f will require forming an approximation to W in a Monte Carlo fashion; see [4]. We choose to present a Monte Carlo approach that is simple to implement, as in [6]. In short, for a random draw $\lambda_i \in \Lambda$, we will find its evaluation under the approximate or analytic gradient (depending on whether we have ∇f analytically), and store the row vector $\nabla f(\lambda_i)^\top$ in a matrix with an SVD corresponding to (3), up to scaling.

In the following, we assume that we lack an analytic form of ∇f; if the analytic gradient is available, the proceeding Monte Carlo method remains valid, and all notations corresponding to approximating ∇f may be dropped.

One initializes the method by performing S random draws of $\lambda_i \in \Lambda$. We then compute $f(\lambda_i)$ for all $i = 1, \ldots, S$ samples, which we note will require, at the very least, S evaluations of f; in a realistic setting, this would require S solves of a model such as a PDE-constrained system. We define $D_S = \{(\lambda_i, f(\lambda_i))\}_{i=1}^S$, a set of S pairs of samples λ_i and their function values. Next, we need $\nabla_\Lambda f$ evaluated at λ_i for all $i = 1, \ldots, S$, which we assume that we do not have in closed analytic form. Hence, we generally need some gradient approximation method [7], and typically a locally linear approximation to the gradient is a fair balance between reasonably estimating the gradient and not pushing computational expenses to an unreasonable regime. With this approximation formed, we denote each estimation to $\nabla f(\lambda_i)$ with $\widehat{\nabla f}(\lambda_i)$ and we define the $N \times S$ matrix $\tilde{W}$ (which is presented below as $\tilde{W}^\top$)

$$\tilde{W}^\top := \begin{bmatrix} \widehat{\nabla f}(\lambda_1) \cdots \widehat{\nabla f}(\lambda_S) \end{bmatrix}. \tag{6}$$

Forming the SVD of $\tilde{W}$, $\tilde{W} = \tilde{U}\tilde{\Sigma}\tilde{V}^*$, we search for a drop off in the magnitude of the singular values $\{\tilde{\sigma}_i\}_{i=1}^S$. Assuming such a drop off occurs for an index $j : 1 < j < S$, we have the j corresponding left singular vectors,$\tilde{u}, \ldots, \tilde{u}_j$. We let $\mathcal{A}(f; D_S) := \text{span}\{\tilde{u}, \ldots, \tilde{u}_j\}$ denote the active subspace of f with respect to the samples D_S. We choose to use a notation with D_S included to emphasize the dependence of the active subspace on the random draws made in Λ, which led to our particular D_S set of samples.

In practice, we can check the extent to which the active subspace accounts for functions values $f(\lambda)$ by checking for resolution in a *sufficient summary plot* [3], where we plot active variables against function values. In these plots, we hope to see a pattern between the active variables versus their function values. For example, if f is quadratic in its active variables, then we expect to see quadratic-resolved sufficient summary plots.

Methods

Learning and Sampling With the viewpoint that evaluations of the model are costly, we want to compute $f(\lambda)$ as little as possible with learning as much as possible about the function. We are interested in several problems that involve sampling Λ and evaluating f including solving the forward problem, finding an active subspace via Monte Carlo, and performing DFO. We also investigate the possibility of learning the L_1 Lipschitz constant of f from random sampling.

Learning the Active Subspace We are interested in several research questions regrading the active subspace learned from different sets of samples in Λ. Generally, for fixed f, we are interested in comparing active subspaces obtained from sampling f with few samples, or with samples generated from some other process, such as a DFO algorithm.

A challenge in learning the active subspace is exploring Λ thoroughly enough to resolve f in fewer variables. The samples (i.e., iterates) formed in a DFO algorithm are only allowed to stray from previous samples by a value proportional to the L_1 Lipschitz constant of f; hence, iterates may explore the space in Λ near the initial iterate in a DFO scheme very thoroughly while missing other global phenomenon. If we only wish to find an active subspace of f in some neighborhood of a point $\lambda \in \Lambda$, such samples may suffice; if we hope to achieve a global understanding of A, we may need to consider supplementing the samples with other, more explorative draws in Λ.

Lipschitz Constant Learning We explore ways in which one may approximate the L_1 Lipschitz constant of f using sampling. Since many DFO algorithms use parameters for step sizes and smoothing that depend on L_1, the literature [1, 5] contains attempts to learn Lipschitz constants from sampling alone. We are most interested in methods that require few evaluations of f.

Using the Active Subspace Generally, if an active subspace A exists for a model f, dimension reduction may be performed by, for instance, projecting inputs λ into $\mathcal{A}$ then evaluating f for the projection; i.e., form $f\left(\mathcal{P}_{\mathcal{A}}(\lambda)\right)$. Computational expense may be saved by projecting points λ into their active variables, since such projections are of lower dimension than λ. This property gives the ability to save computational expense for a number of problems in Uncertainty Quantification, including optimization, representation and solving inverse problems. In the following, we consider ways in which we may use the information given by an active subspace $\mathcal{A}(f; D_S)$.

DFO in the Active Variables Only Given some f and its corresponding active subspace $\mathcal{A}(f; D_S)$ found by the Monte Carlo method discussed in the preceding section, we are interested in investigating the effectiveness of only optimizing f in its active variables. There are several approaches one may consider, and handful of those approaches and their corresponding results are discussed in the proceeding section. The most compelling approach we have observed is to modify the DFO algorithm discussed above to only take random walks in directions lying in $\mathcal{A}$. That is, at iteration k, standard DFO algorithms [2] use random walks given by drawing a random vector $v^{(k)}$ of dimension N in which every component $v_i^{(k)}, i = 1, \ldots, N$ of v is drawn from a specified normal distribution. Instead, given the first j singular unit vectors $u_1, \ldots, u_j$ corresponding to the SVD of $\hat{W}$, one may take j draws from a specified normal distribution, which we denote with $s_i \sim N(\mu, \sigma^2)$, and form the random vector v for the k-th step in a DFO algorithm as $v^{(k)} = \sum_{i=1}^{j} s_i u_i$, which is just a linear combination of the active variables of f with coefficients given by random draws; we see that such a vector could be interpreted as a random walk in $\mathcal{A}$.

Relevant research directions include: considering other ways to embed the information from $\mathcal{A}$ within a DFO scheme; analyzing the difference between minimizing f in all its variables versus only in its active subspace; considering ways in which the problem may be solved with the mentioned techniques in tandem, such as using a "burn-in" phase in which f is minimized over all of Λ followed by an active variables minimization.

Representations and Surrogate Modeling We are generally interested in the representation and surrogates one may form with an active subspace. If f is truly a black box, then it lacks an analytic form we can access. By performing dimension reduction, it is possible to test the resolution of function values f versus active variables by forming a sufficient summary plot or *response surface* [3]. Depending on the quality and form of the resolution, one may (or may not) be able to discern the relation between f and its active variables visually, by fitting a surrogate through the response surface and testing the fit, or both. Representing f with a lower-dimensional surrogate may help with solving other problems of interest. In particular, we may be able to solve a forward problem at lower cost, or pose the convex optimization for solving an inverse problem with such a representation used to form the misfit functions considered in the preceding section.

Results

In this section, we re-visit the questions from the previous section to discuss preliminary results, if any, and present a plan to begin investigating questions that are unanswered. We present a guiding example which is discussed in below sections in the context of relevant research questions and results.

Example 3. *Let* $\Lambda = [-1,1]^{11}$ *and define*

$$f(\lambda) = \sum_{i=0}^{10} 2^{(-1)^i i} \lambda_i^2 + \epsilon(\lambda),$$

where $\epsilon(\lambda)$ *is a draw of additive noise corresponding to the input* λ*; here, we take draws of* ϵ *of order* 10^{-4}*. We see that* $\mathcal{D} = [0, 2^{10}]$ *and* $N = 11$*,* $M = 1$*. Note that the minimimum of* f *is given by* $0 \in \Lambda$*. Here, as* i *increases, terms in* f *become either more important or less important, depending on whether* i *is even or odd.*

Learning and Sampling We are particularly interested in the quantitative and qualitative difference between active subspaces learned from random sampling versus active subspaces learned from deterministic draws that come from another algorithm such as DFO.

We revisit Example 3, and present two different active subspaces of f – one active subspace was formed in a Monte Carlo fashion with 1000 random draws in Λ; the other subspace was formed with only 100 random sample of f where sample points are in fact iterates from a DFO scheme. To consider a realistic scenario, we form the active subspace in both cases by making global quadratic approximations to ∇f, which will work well in our regime since f is quadratic in Λ.

In Figure 2, we show the results from the first active subspace of f, which was obtained from 1000 iid uniform random samples in $[-1,1]^{11}$; i.e., for any sample λ_i, we have $\lambda_i \sim U[-1,1]^{11}$. The plot on the left-hand side of Figure 2 shows the resulting eigenvalues (the squares of the singular values) from taking the SVD of $\hat{W}$, which we recall is simply a matrix (in this case 1000×10) where row i is the transpose of approximated gradient of f evaluated at a sample λ_i, $i = 1, \ldots, 1000$. Notice that there is one dominating eigenvalue, which is on the order of 10^6; this

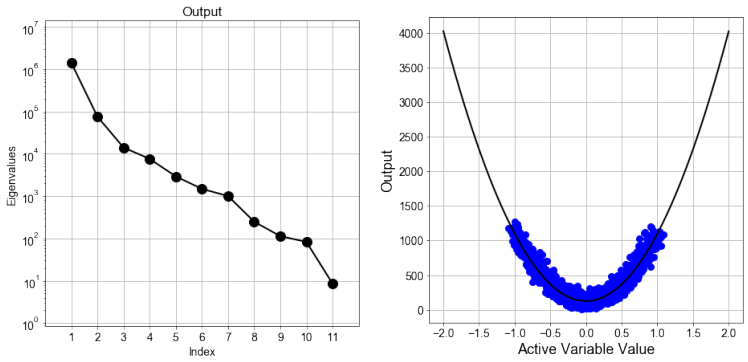

Figure 2: Left: A plot of the eigenvalues of the matrix $\hat{W}$ formed from 1000 Monte Carlo samples in Λ. We see one dominant eigenvalue on the order of 10^6. Right: A sufficient summary plot where all 1000 samples are projected into $\mathcal{A}$ and plotted against their function values.

matches intuition since the final term in $f(\lambda)$, given by $2^{10}\lambda_{10}^2$, will be, at most, on the order of 10^3 (and again, eigenvalues are the squares of singular values). This eigenvalue corresponds to λ_{10}, meaning that the value of λ_{10} has the most impact out of all other parameters on the function values $f(\lambda)$. Notice that the eigenvalue near 10^6 does not dominate the other eigenvalues by much; indeed, another eigenvalue appears around the order 10^5. Since the second-to-last term in f is small (with coefficient 2^{-9}), this order 10^5 eigenvalue must correspond to $2^8\lambda_8^2$.

The active subspace implementation used here [3] detects the 10^6 order eigenvalue as dominating enough to determine that $\mathcal{A}$ is one-dimensional, spanned by the singular vector corresponding to the direction $(0, \ldots, 0, 1)$, which is a unit vector in the axis of λ_{10}, orthogonal to all other axes in Λ.

The plot on the right-hand side of Figure 2 shows the sufficient summary plot, where the 1000 samples are projected into $\mathcal{A}$ using (4) and their values are plotted along the horizontal axis against their original function values on the vertical axis. The sufficient summary plot can be thought of as a visualization of f along the axes for which f is active; thus, we are seeing a quadratic response in f versus points projected onto the λ_{10} axis. We use the mentioned software tools to fit the response with a polynomial surrogate, which is a fit where $R^2 \approx 0.9$. Indeed, points do not fall perfectly along the surrogate since there are other parameters which contribute to the value of f.

Now we investigate the effectiveness of learning $\mathcal{A}$ with different samples. Since we want to avoid evaluating f heavily and we want to perform several analyses of f, it will be advantageous to "recycle" function values whenever possible.

To experiment, we consider trying to learn $\mathcal{A}$ from 100 iterates and their function values generated from performing the DFO algorithm STARS [2] on f. The iterates generated from a DFO algorithm may make large jumps (limited by a step size and smoothing parameter) through regions of Λ space early in the routine, but will begin to take smaller and smaller steps as the routine hones in on a local minimum. We note that typically for a fair comparison between sampling methods, one would use the same number of samples; however, we choose to use much less samples in the DFO case for the exact reason that the samples will cease to explore parameter space in an informative manner once the iterates begin to converge to a local minimum. In Figure 3, we present the sufficient plot obtained from learning $\mathcal{A}$ from 100

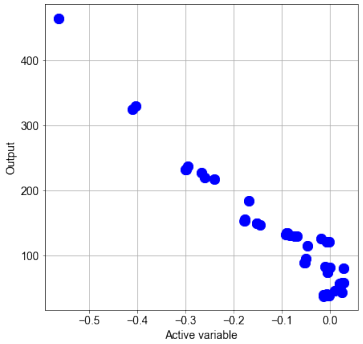

Figure 3: A sufficient summary plot where all 100 samples are iterates from a DFO algorithm.

DFO samples. The eigenvalue plot obtained from the analysis was virtually identical to that in Figure 2, but the sufficient summary plot here in Figure 3 looks very different than the plot in Figure 2 since the sampled points were not random, but deterministic. Notice that the large function value occurs in the top-left corner of the sufficient summary plot in Figure 3; this value corresponds to the initial iterate $\lambda^{(0)}$, which had a function value of approximately 460. Iterates drive down the value of f, and we see that the large majority of the iterates (samples) occur in the bottom-right corner of the plot, where the DFO algorithm is converging to a local minimum.

We find that in our case the samples generated from the DFO algorithm explored Λ enough to give an active subspace with negligible differences to that obtained in the case of 1000 Monte Carlo draws.

We are interested in exploring the extent to which $\mathcal{A}$ can be learned for more general functions f, particularly those arising in certain applications. We are also interested in learning the L_1 Lipschitz constant of f in this setting. For the example analyzed here, the step size and smoothing parameters, which are functions of the value of L_1, were formed heuristically. A more careful analysis is left to be desired, especially in the case that ∇f is inaccessible so that L_1 must be estimated. Techniques in the literature need to be considered and tested.

Using the Active Subspace We are generally interested in using the active subspace of a function whenever it may reduce computational expense while maintaining a representations that does not deviate too far from the true action of f. Here we consider the effectiveness of DFO performed on f from Example 3 in a standard way [2] using 500 iterations versus a scheme that blends STARS and active subspace analysis using only 150 iterations.

We find that after 500 iterations of the standard DFO implementation the routine closes in on the minimum value of f, finding that $f\left(\lambda^{(500)}\right) \approx 0.84$. In Figure 4, the left-hand plot shows a *log-log* plot of the logarithm of the iteration on the horizontal axis against the logarithm of the function evaluated at that iterate. We begin to see convergence to a minimum in 500 iterations. We modified the DFO algorithm to find a similar result with only 150 iterations.

To make a modified DFO routine, we begin with a "burn in" of 50 iterations of standard DFO, which actually gives 100 samples of the function at different points in Λ, as one step of the DFO routine has two calls of f. From the 100 samples denoted with D, we form $\mathcal{A}(f, D)$. The results of this analysis were discussed above; recall, we obtain a one-dimensional $\mathcal{A}$ spanned in the direction of λ_{10}. Though $\mathcal{A}$ is one-dimensional, there are still other important parameters in the model, including λ_8 and λ_6, which make order 10^2 and 10^1 contributions to f respectively. After the standard DFO burn-in, which was used to both minimize f and explore Λ to obtain

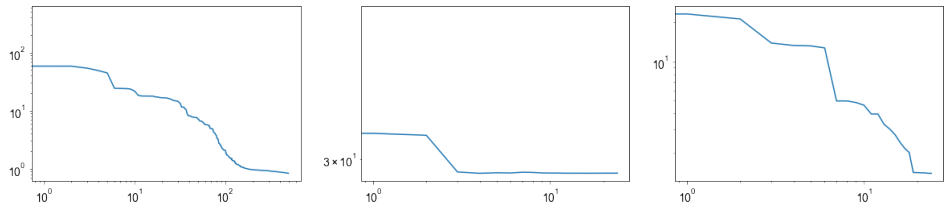

Figure 4: 3 log-log plots from (left to right): 500 iterations of standard DFO; 25 iterations of minimizing f with DFO along the λ_8 axis; 25 iterations of minimizing f with DFO along the λ_6 axis.

an approximate $\mathcal{A}$, we begin to use DFO only in the directions of the active and some inactive variables. Since we have only a single direction to minimize along in the active subspace, we modify the DFO algorithm to take random steps in the direction of the λ_{10} axis only. We do so by taking draws from a $d \sim N(0,1)$ and forming the randomly scaled vector du_1, where u_1 is the singular vector corresponding to the active direction in λ_{10}. Now, each iterate will only be perturbed in its last component in the DFO scheme. We find that in this example, the STARS burn-in phase reduces the 10th component of iterates so that $\lambda_{10}^{(50)} \approx -0.01$ and $f(\lambda^{(50)}) \approx 38.02$. Performing DFO in the λ_{10} axis with 25 iterations gives $\lambda_{10}^{(75)}$ on the order of 10^{-4}, but $f(\lambda^{(75)}) \approx 37.74$, which is only a small drop in the function value. Next, 25 iterations of DFO was performed along the λ_8 axis – this time, we see a significant drop in the value of f, where $f^{(100)} \approx 29.34$. Despite the fact that the λ_8 direction is not considered active, we do see the value of f decrease as f is minimized along its axis. Another 25 iterations of DFO was performed along the direction corresponding to the third singular vector, dropping the value of f by about 6. Finally, 25 iterations of DFO was performed along the axis corresponding to the fourth singular vector, which minimizes f to an order similar to the results from 500 iterations of standard DFO. Indeed, $f(\lambda^{(150)}) \approx 1.38$.

In the middle plot of Figure 4, we show the log-log plot obtained from using 25 steps of DFO in the λ_8 direction; the right-hand log-log plot shows the results of 25 steps of DFO in the λ_6 direction.

We note that we have presented just one of several algorithms that have in some way blended the active subspace of f with a DFO scheme. We are interested in investigating the best way to use the information of $\mathcal{A}$ in DFO. Likewise, we are interested in representing f with surrogates based on $\mathcal{A}$ and investigating whether $\mathcal{A}$ can lessen the computational expense in the forward problem.

From model problems to fusion applications We propose to investigate the approximate solution of data-consistent inverse problems via optimization approaches. For derivative free optimization, we plan to transition from our noisy functions to the data consistent inversion of a subgrid LES model for isotropic turbulent flow. While this would certainly be a publishable result, the hope is to investigate approximate DCI when applied to anomalous diffusion in the axisymmetric gyrokinetic code XGCa.

In the case of models where λ-gradients are available and not of suspect quality, we will transition from an approximate DCI for a model elliptic problem with a parameterized forcing function, with parameter gradients obtained via adjoint methods. Success here will allow us to extend the method to the determination of data-consistent MHD equilibria (via the Grad-Shafranov equation).

Software Dissemination Currently, only some of the software used to produce results in this paper are publicly available on GitHub.com. Some of the algorithms used here and other algorithms that are of interest are within open-source packages available online including those of Constantine and Butler et al. Other schemes considered in this paper make modifications to given algorithms and remain under development. A major goal of this thesis proposal will be developing well-documented, open-source software complete with python Jupyter Notebooks and illustrative, replicable examples.

Conclusion and Discussion

References

[1] Jan-Peter Calliess. "Lipschitz optimisation for Lipschitz interpolation." In 2017 American Control Conference (ACC 2017), Seattle, WA, USA, May 2017.
[2] Chen and Wild. "Randomized Derivative-Free Optimization of Noisy Convex Functions." Funded by the Department of Energy. 2015.
[3] Constantine, Paul G. "Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies." SIAM, 2015.
[4] Constantine, Eftekhari, Wakin. "Computing Active Subspaces Efficiently with Gradient Sketching." Conference paper, 2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP).
[5] Kvasov and Sergeyev. "Lipschitz gradients for global optimization in a one-point-based partitioning scheme." Journal of Computational and Applied Mathematics. Volume 236, Issue 16, pp. 4042-4054. 2012.
[6] Russi, Trent M. "Uncertainty Quantification with Experimental Data and Complex System Models." Dissertation, University of California Berkeley. 2010.
[7] Smith, Ralph."Uncertainty Quantification: Theory, Implementation, and Applications." SIAM, 2013.
[8] Tarantola, Albert. "Inverse Problem Theory and Methods for Model Parameter Estimation." SIAM. 2005.
[9] Wildey, T., Butler, T., Jakeman, J., Walsh, S. " A Consistent Bayesian Approach for Stochastic Inverse Problems Based on Push-forward Measures." SAND2017-3436PE. 2017.