# Model Inversion for Complex Physical Systems Using Low-Dimensional Surrogates

Jimmie Adriazola [1], Manuchehr Aminian [2], Pau Batlle [3], Changqing Cheng [4], David A. Edwards [5]†, Taras Lakoba [6] and Olivia Pomerenk [7]

[1] *University of California, Santa Barbara; Santa Barbara, CA, USA*
[2] *Cal Poly Pomona; Pomona, CA, USA*
[3] *California Institute of Technology; Pasadena, CA, USA*
[4] *State University of New York at Binghamton; Binghamton, NY, USA*
[5] *University of Delaware; Newark, DE, USA*
[6] *University of Vermont; Burlington, VT, USA*
[7] *New York University; New York, NY, USA*

**Study Group:** Mathematical Problems in Industry Workshop, New Jersey Institute of Technology, June 12–16, 2023
**Communicated by:** Linda Cummings
**Industrial Partner:** Pacific Northwest National Laboratory
**Presenter:** David A. Barajas-Solano
**Team Members:** Charlotte Beckford, University of Tennessee; Arum Lee, Penn State; Richard McQueen, Rensselaer Polytechnic Institute; Arnab Roy, University of Delaware; Gaohui Zhang, University of Minnesota
**Industrial Sector:** Environment

**Key Words:** Model reduction; model inversion; inverse regression; contaminant transport.

**MSC2020 Codes:** 62J02, 62H12, 35B30.

**Summary**

In order to predict groundwater contamination levels, it is convenient to estimate the transmissivity field which governs contaminant transport. It is prohibitively expensive to solve the underlying Darcy flow equations as a direct solver in a parameter-fitting process. The number of input parameters can be reduced to just a few using linear or nonlinear techniques. The dependence of the (measured) pressure on those input parameters can be likewise reduced to a simple functional form. These reductions greatly speed the computation while preserving accuracy. Discussion of more complicated dimension reduction techniques (as well as neural network implementation) are also presented.

† Corresponding Author: dedwards@udel.edu

## lanford Site

ponsible for the Hanford Site, “one of
cleanup efforts in the world, managing
of five decades of nuclear weapons

groundwater flow and contaminant
odels of the Hanford Site is crucial for
emediation strategies and performing
ssessments

f sparsely-distributed observation wells
surements of hydraulic pressure and
through curves from tracer
s

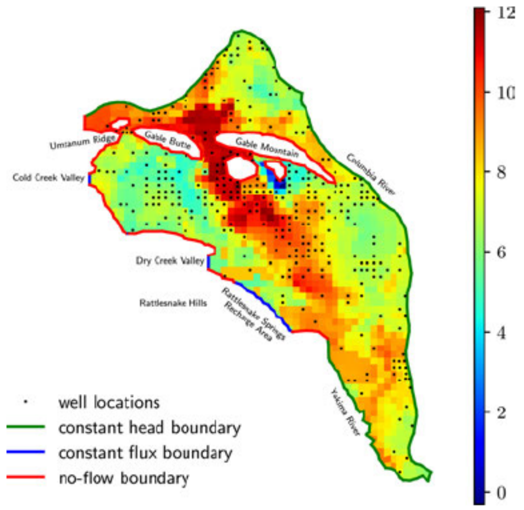


Figure 1. Schematic of Hanford site.

In order to predict contaminant levels in groundwater at the Hanford waste site, scientists need a model for the steady-state velocity profile $\mathbf{q}(\mathbf{x})$ of the groundwater. The velocity profile is affected by the *transmissivity field* $T(\mathbf{x})$, which describes how the various material characteristics of the soil, etc., affect the fluid flow. It is this transmissibility field we wish to estimate.

The function $T(\mathbf{x})$ can be decomposed in the following way:

$$\log T(\mathbf{x}) \equiv y_{\text{tm}} + y(\mathbf{x}), \qquad y(\mathbf{x}) = m(\mathbf{x}) + \sum_{k=1}^{N_\xi} \xi_k \psi_k(\mathbf{x}), \tag{1.1}$$

where $y_{\text{tm}}$ is the “total mean” of $\log T$ over $\mathbf{x}$, $m(\mathbf{x})$ is a centering function, the $\psi_k(\mathbf{x})$ are basis functions, and the $\xi_k$ are weights. This is known as the *Karhunen-Loève Expansion* (KLE) [6, §16.12].

These functions are presumed known, and hence the dependence of $\mathbf{q}$ on $T$ can be replaced by dependence on the *weight vector* $\boldsymbol{\xi}$, where $\boldsymbol{\xi} \in \mathbb{R}^{N_\xi}$. Hence our optimization problem becomes one of choosing the best $\boldsymbol{\xi}$ so that our computed transmission field matches the true $T(\mathbf{x})$.

Unfortunately, we have no direct measurements of $T$ or even $\mathbf{q}$. However, the velocity can be computed in a straightforward manner from the pressure $u(\mathbf{x}; \boldsymbol{\xi})$, and it is that quantity which we wish to study. In particular, the pressure profile satisfies the Darcy flow system

$$\begin{aligned} -\nabla \cdot (T(\mathbf{x})\nabla u(\mathbf{x})) &= 0, \quad \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= u_1(\mathbf{x}) \quad \mathbf{x} \in \partial\Omega_1, \\ \frac{\partial u}{\partial n} &= u_2(\mathbf{x}) \quad \mathbf{x} \in \partial\Omega_2, \end{aligned} \tag{1.2}$$

where $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$. The problem cannot be solved analytically because of the complicated domain (see Fig. 1), so one may use a finite-volume approach and calculate the pressure within $N_\mathrm{c}$ cells (for actual values, see the Appendix).

The weights $\boldsymbol{\xi}$ are unknown, so we must estimate them based on experimental measurements. We do not have measurements in all the cells, just the ones that have wells in them where measurements can be taken. We then construct a vector $\mathbf{u}(\boldsymbol{\xi}) \in \mathbb{R}^{N_\mathrm{w}}$ whose components are the values in only those cells for which we have observations. Here the subscript "w" stands for wells.

Formally, we would then solve

$$\boldsymbol{\xi}_* = \arg\min_{\boldsymbol{\xi}} ||\mathbf{u}_\mathrm{w} - \mathbf{u}(\boldsymbol{\xi})||. \tag{1.3}$$

Unfortunately, the problem as stated is poorly conditioned, as different values of $\boldsymbol{\xi}$ can yield similar values of $u$. Hence we introduce a regularization term:

$$\boldsymbol{\xi}_* = \arg\min_{\boldsymbol{\xi}} ||\mathbf{u}_\mathrm{w} - \mathbf{u}(\boldsymbol{\xi})|| + \gamma\mathcal{R}(\boldsymbol{\xi}), \tag{1.4}$$

where $\gamma > 0$ is a regularization parameter and $\mathcal{R}$ is a regularization function which penalizes unwanted behavior. Typical forms for $\mathcal{R}$ include

$$||\boldsymbol{\xi}||, \qquad ||\mathbf{y}||, \qquad ||\overrightarrow{\nabla_\mathbf{x} y}||. \tag{1.5}$$

Here $\mathbf{y}$ is a vector of measurements of the log-transmissivity field $y$ and in the last term we are constructing the vector of (numerical discretized) gradients at the observation points.

The ideal way to do the minimization would be to use (1.2) to calculate $u$ given $\boldsymbol{\xi}$, and then optimize as described. Unfortunately, this is incredibly expensive computationally given the complicated nature of the domain in Fig. 1. Hence we want to create low-dimensional models that can still capture enough of the behavior to provide good estimates for $\boldsymbol{\xi}$. In the next five sections, we discuss various procedures and implementations to perform such model reduction. The final section before the conclusion provides a more theoretical discussion of more complicated model reduction methods.

## 2 Breaking Up the Problem: Basic Algorithms

Unfortunately, the number of cells and weights is quite large, which causes lengthy calculations. Therefore, we reduce the number of parameters to be estimated by assuming that *for each cell j*,

$$u(\boldsymbol{\xi}) \approx f(\boldsymbol{\eta}(\boldsymbol{\xi}); \mathbf{c}), \tag{2.1}$$

where $\boldsymbol{\eta}$ is a *simplified weight vector* and $\mathbf{c}$ is a *small* number of parameters describing the behavior of $f$. Note that these vectors must be found (and may be different) for each $j$; however, for notational simplicity we suppress that dependence for now.

Importantly, the vector $\boldsymbol{\eta}$ is assumed to be of much smaller dimension than $\boldsymbol{\xi}$ (reducing the complexity of the independent variable). Since the forward solver is expensive, this saves significant computational work when (in the final step) we optimize over $\boldsymbol{\xi}$. In fact, in the work done at the workshop, we assumed $\boldsymbol{\eta}$ to be a scalar (see (2.2) below). Techniques to extend this to more dimensions $N_\eta \ll N_\xi$ are considered in §7.

Each of the above steps must be done optimally. Therefore, we break up the optimization into discrete steps as follows.

## 2.1 Finding $\eta$

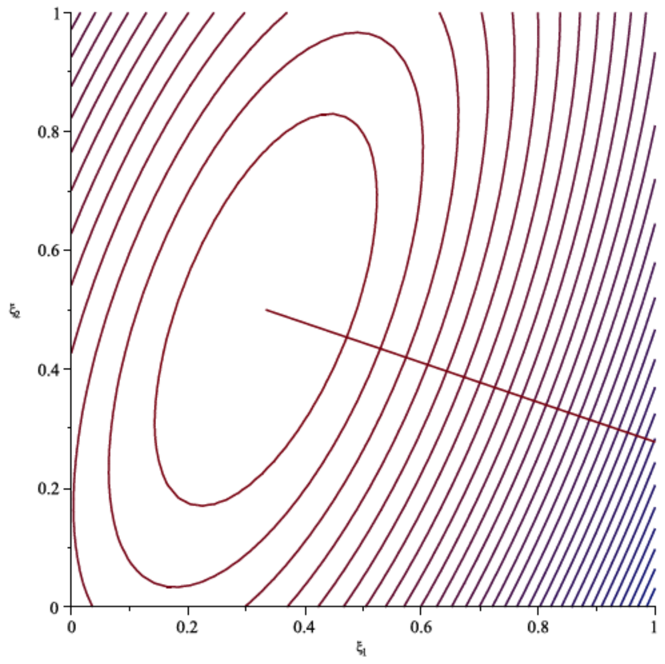


Figure 2. Schematic of $u$ variation and associated $\hat{\mathbf{a}}$.

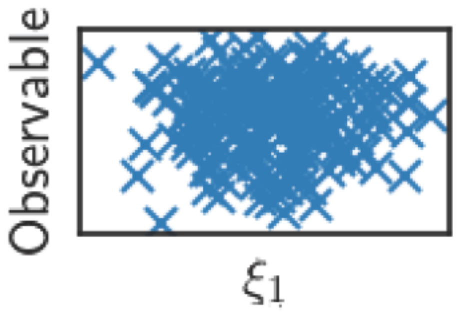
First consider how $u$ at a particular point $\mathbf{x}_j$ depends on the parameters $\boldsymbol{\xi}$. For simplicity of illustration, we take $N_\xi = 2$ and plot a typical profile in Fig. 2. For any point on the line shown, variations in $u$ along the line are much greater than those perpendicular to it. Hence the specific values of $\boldsymbol{\xi}$ are not as important as the distance along the line shown.

How this would look for particular measurements is shown in Fig. 3. If we look at the observable $u$ *vs.* $\xi_1$, we may see no discernible pattern. But if we look at distance along the line $\eta_1$, we see a definite trendline.

Therefore, if we define the unit vector in that direction as $\hat{\mathbf{a}}$, then $\boldsymbol{\eta}$ just becomes the

$\langle \hat{\mathbf{a}}, \boldsymbol{\xi} \rangle , \quad \hat{\mathbf{a}} := \mathbf{a}/\|\mathbf{a}\|$

## 1. Find the rotation matrix $\mathbf{A}$



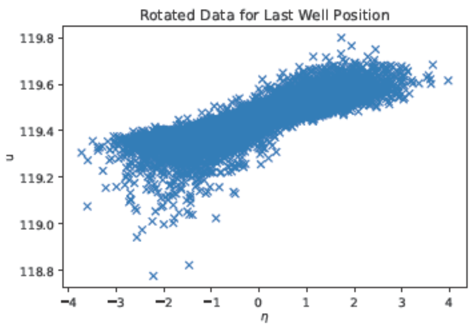
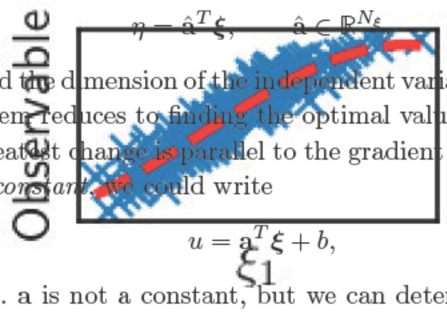


Figure 3. Trendline emerges when plotting against proper variable.

## 2. Find the regression function $f(\cdot)$

scalar projection of the corresponding value of $\boldsymbol{\xi}$ along that line:

$$\eta = \hat{\mathbf{a}}^T \boldsymbol{\xi}, \qquad \hat{\mathbf{a}} \in \mathbb{R}^{N_\xi} \tag{2.2}$$



Hence we have reduced the dimension of the independent variable from $N_\xi$ to 1. Therefore, this part of the problem reduces to finding the optimal value of $\hat{\mathbf{a}}$ for each cell.

The direction of greatest change is parallel to the gradient vector, so we have $\mathbf{a} = \nabla_{\boldsymbol{\xi}} u$. Therefore, *if* $\mathbf{a}$ *were constant*, we could write

$$u = \mathbf{a}^T \boldsymbol{\xi} + b, \tag{2.3}$$

where $b$ is a constant. $\mathbf{a}$ is not a constant, but we can determine a value good for each cell by doing a least-squares fit of (2.3). Note that this is a relatively quick computation since the right-hand side is linear.

Therefore, we construct $N_\mathrm{r}$ different choices of $\boldsymbol{\xi}$ (here the subscript "r" denotes "realization"). We denote each realization by $\boldsymbol{\xi}^{(i)}$, and using the forward solver determine the computed pressure $u^{(i)}$ at each point. We then wish to find the vector $\mathbf{a}$ which is the best least-squares fit of

$$u^{(i)} = \mathbf{a}^T \boldsymbol{\xi}^{(i)} + b, \qquad i = 1, 2, \ldots, N_\mathrm{r}. \tag{2.4}$$

(Though we will obtain the constant $b$ as part of our fitting procedure, we will never use it.)

At this point it is now convenient to include the dependence on the cell number $j$ explicitly. We define $\mathbf{u}_j$ to be the vector whose $i$th component is $\mathbf{u}(\mathbf{x}_j; \boldsymbol{\xi}^{(i)})$, so $\mathbf{u}_j \in \mathbb{R}^{N_\mathrm{r}}$. Similarly, we define $\Xi$ to be the matrix whose $ik$th entry is $\xi_k^{(i)}$ (or equivalently, whose $i$th row is $\boldsymbol{\xi}^{(i)}$). Then the above least-squares problem can be written as

$$\mathbf{u}_j = \Xi \mathbf{a}_j + b_j \mathbf{1}, \qquad \Xi \in \mathbb{R}^{N_\mathrm{r} \times N_\xi}, \quad \mathbf{1} \in \mathbb{R}^{N_\mathrm{r}}. \tag{2.5}$$

The resulting best-fit vector $\mathbf{a}_j$ can be normalized to construct $\hat{\mathbf{a}}_j$, and we may define

$$\eta_j = \hat{\mathbf{a}}^T \boldsymbol{\xi}. \tag{2.6}$$

Moreover, this process can be streamlined even further. Define $U$ to be the matrix whose $j$th column is $\mathbf{u}_j$, and similarly for $A$. Define $\mathbf{b}$ such that its $j$th entry is $b_j$. Then

(2.5) can be rewritten as

$$U = \Xi A + \mathbf{1}\mathbf{b}^T; \qquad U \in \mathbb{R}^{N_\mathrm{r} \times N_\mathrm{w}}, \quad A \in \mathbb{R}^{N_\xi \times N_\mathrm{w}}, \quad \mathbf{b} \in \mathbb{R}^{N_\mathrm{w}}. \tag{2.7}$$

We then can find the best-fit matrix $A$ all at once...in theory. In practice, it seems that the default `scipy` linear fitting algorithm takes only vectors as left-hand sides, not matrices. So it may be better to use (2.5) for each $j$. Alternatively, we may use the normal equation approach by rewriting (2.7) as

$$\begin{aligned} U &= \begin{pmatrix} \Xi & \mathbf{1} \end{pmatrix} \begin{pmatrix} A \\ \mathbf{b}^T \end{pmatrix} \equiv \tilde{\Xi}\tilde{A} \\ \tilde{\Xi}^T U &= \tilde{\Xi}^T \tilde{\Xi}\tilde{A} \\ \tilde{A} &= (\tilde{\Xi}^T\tilde{\Xi})^{-1}\tilde{\Xi}^T U. \end{aligned} \tag{2.8}$$

Once $A$ has been found, it is straightforward to find the desired vectors $\hat{\mathrm{a}}$. We normalize each column of $A$, and let $\hat{A}$ to be the matrix whose $j$th column is $\hat{\mathrm{a}}_j$. Then since (2.2) is defined for all $j$, we may combine all those values to form the new equation $\boldsymbol{\eta} = \hat{A}^T\boldsymbol{\xi}$, so $\boldsymbol{\eta} \in \mathbb{R}^{N_\mathrm{w}}$, and $N_\mathrm{w} < N_\xi$.

## 2.2 Finding $f$ and c

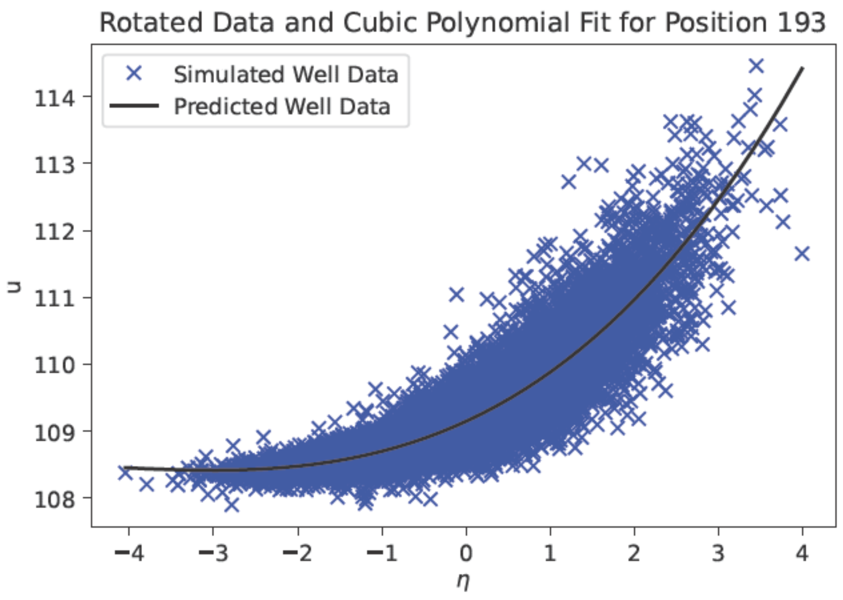


Figure 4. Best-fit cubic curve.

Now that $\eta$ has been defined, we can find the optimal $f$ to approximate the pressure. Again, this computation must be done for each observation point. To do this, we first posit a functional form for $f$. The first form suggested was a third-order Hermite polynomial, but this is functionally equivalent to a standard cubic:

$$f(\eta;\mathbf{c}) = \sum_{k=0}^{3} c_k \eta^k. \tag{2.9}$$

(Note that we use the same index $k$, since the $c_k$ play the role of weights in this equation.) In this case, we need fit only the four constants $c_k$ to obtain the best-fit curve (see Fig. 4). This type of fit can be handled easily by the `numpy` function `polyfit`.

Mathematically, it is fast because it is a linear fit, for if we define the matrix $M$ as follows:

$$M_{ik} = [\eta^{(i)}]^{k-1}, \qquad M \in \mathbb{R}^{N_\mathrm{r}\times 4}, \tag{2.10}$$

then this optimization problem can be written in the following least-squares format:

$$\mathbf{u}_j = M\mathbf{c}_j, \tag{2.11}$$

which is done at each $j$. Similar to before, we may define a matrix $C$ whose $j$th column is $\mathbf{c}$, which yields

$$U = MC, \qquad C \in \mathbb{R}^{4\times N_\mathrm{w}}. \tag{2.12}$$

Given that we already have the expression (2.3), why not just do a linear fit for $f$ instead? From the theoretical perspective, we expect that $u$ will be bounded by certain values (namely the boundary conditions), and in theory a linear profile could exceed those bounds for extremal $\eta$. We present more details of other possible forms in §3.2.

## 2.3 Finding $\boldsymbol{\xi}_*$

Now that we have completed the first two parts, (1.4) can be rewritten as

$$\boldsymbol{\xi}_* = \arg\min_{\boldsymbol{\xi}} ||\mathbf{u}_\mathrm{w} - \mathbf{f}(\hat{A}^T\boldsymbol{\xi};\mathbf{c})|| + \gamma\mathcal{R}(\boldsymbol{\xi}), \tag{2.13}$$

which should be much faster to calculate than using a forward solver of (1.2) to calculate $\mathbf{u}$. (Again we would use a nonlinear optimization subroutine.) As a first attempt, we solve the minimization with $\gamma = 0$ (no regularization). We know this problem will be very ill-posed: since $N_\mathrm{w} < N_\xi$, we have (over 600) more unknowns than equations. Hence in later sections we will introduce regularization.

As a first visualization of performance, we plot the predicted values of $u$ *vs.* the reference value in Fig. 5.

Recall that the true quantity we estimate is $T(\mathbf{x})$. Therefore, our optimization proceeds as follows. We were given a reference field $y_\mathrm{ref}(\mathbf{x})$, whose values are illustrated in Fig. 6. Given $y_\mathrm{ref}(\mathbf{x})$, we can use the provided forward solver to compute $\mathbf{u}_\mathrm{w}$ given that transmissivity field. We then optimize using (2.13) and use those values of $\mathbf{u}_\mathrm{w}$ to obtain a $\boldsymbol{\xi}_*$, which we then substitute into (1.1) to find a comparable estimate for $\mathbf{y}_\mathrm{w}$.

The results are shown in Fig. 7. In this case, $\boldsymbol{\xi}_*$ was calculated using no regularization. To calculate the error, we use the following quantity:

$$\text{transmissivity error} = \frac{||\mathbf{y}_\mathrm{w} - \mathbf{y}(\boldsymbol{\xi})||}{||\mathbf{y}_\mathrm{w} + y_\mathrm{tm}\mathbf{1}||}. \tag{2.14}$$

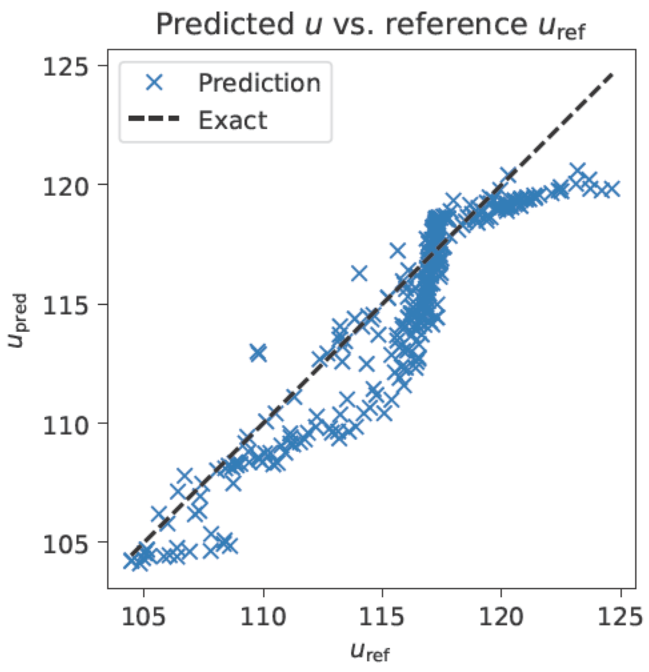


Figure 5. Scatter plot showing performance of algorithm in estimating $u$.

The numerator is the standard error we expect. The denominator is a normalization factor against the true $\log T$, which is the sum of $y$ and the total mean.

## 3 Breaking Up the Problem: Extensions

Each of the steps outlined in §2 is quite basic, and can be improved (though with increased computational time). We outline some possibilities below:

### 3.1 Finding $\eta$

Another way to calculate $\eta$ would be to add a quadratic term. We choose the simplest possible choice by replacing (2.3) with

$$u = \mathrm{a}^T \boldsymbol{\xi} + \boldsymbol{\xi}^T \Lambda \boldsymbol{\xi} + b, \tag{3.1}$$

where $\Lambda$ is a diagonal matrix. This is a quadratic fit, and perhaps there's a Python command for that. However, given the results in the next section showing a linear function is best, it probably would not improve things appreciably to add this complication, as this is related to using a quadratic as a form for $f$.

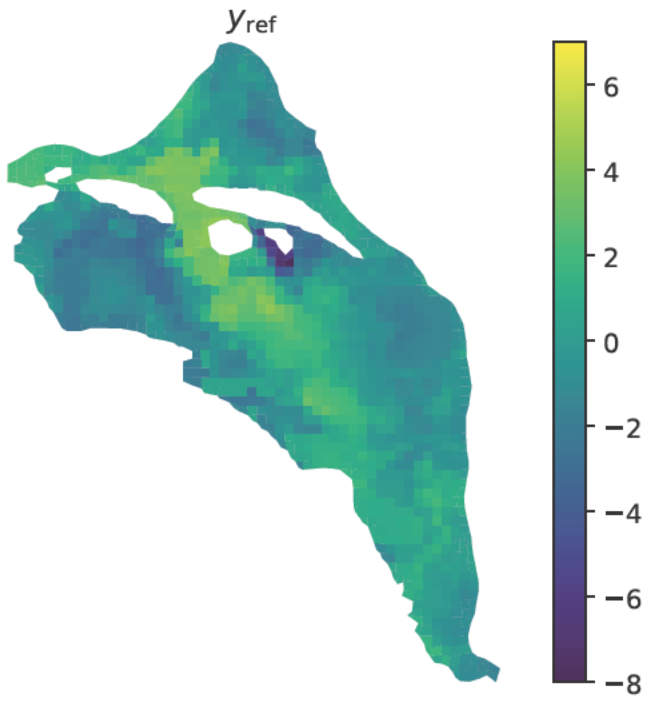


Figure 6. Reference $y$ field to be fit.

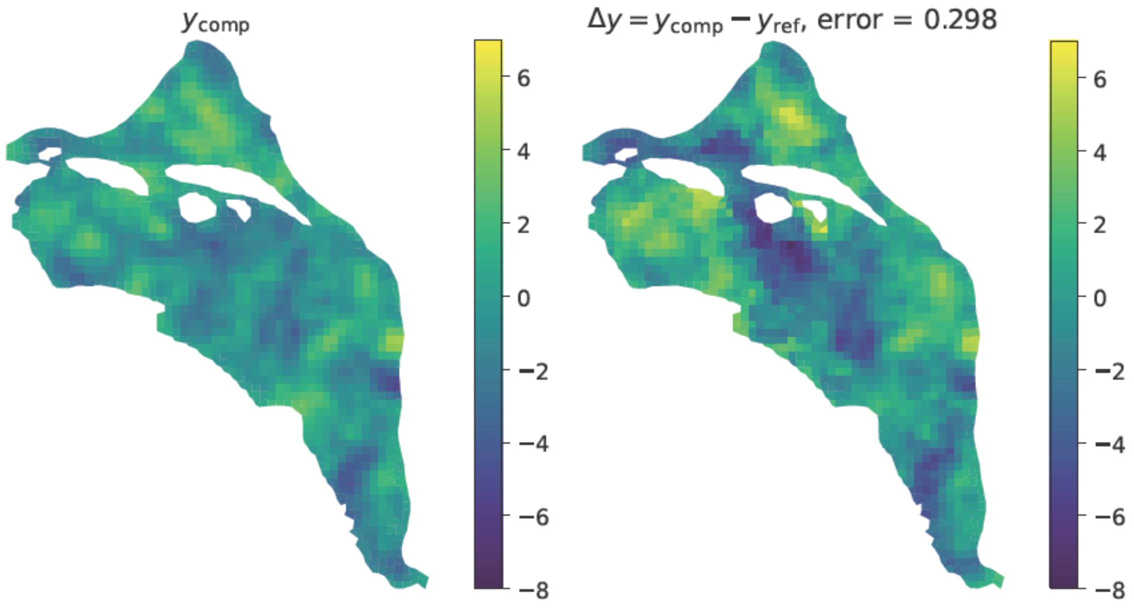


Figure 7. Left: Calculated $y$. Right: Error in calculation.

### 3.2 Finding $f$ and $\mathbf{c}$

The reason why a cubic was chosen as our first attempt for $f$ was so that it could "saturate" for high values of $\eta$ (see Fig. 4). Of course, it will also eventually decline, and for reasons related to the equation solved, $u$ should be monotonically increasing in $\eta$.

But perhaps a cubic is not even needed. Therefore, rather than using (2.9), we use a different degree polynomial. Then motivated by (2.11), we compute the following error:

$$\text{error} = \frac{1}{N_\text{w} N_\text{r}} \sum_{j=1}^{N_\text{w}} ||\mathbf{u}_j - M\mathbf{c}_j||. \tag{3.2}$$

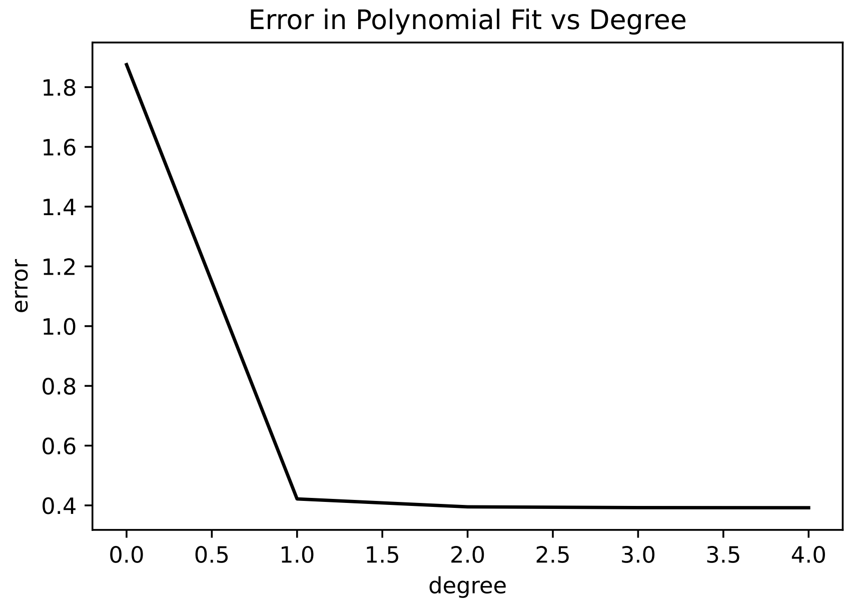


Figure 8. Error in (3.2) *vs.* degree.

This error is plotted in Fig. 8. One can clearly see that there is only minimal error reduction for degrees beyond 1. Hence a linear fit is acceptable, and using the cubic would seem to be just a waste of computational resources.

Another method we investigated was fitting the data to a sigmoid curve (also known as a hyperbolic tangent) instead, using `scipy`'s `curve_fit` function. Nonlinear fitting is more costly than linear fitting, but the sigmoid form is uniquely well-suited to capture the saturation effect because of its upper asymptote. The general form sigmoid we used is

$$f(\eta; \mathbf{c}) = c_0 + \frac{c_1}{1 + e^{-(x-c_2)/c_3}}. \tag{3.3}$$

As long as $c_3$ is positive, the sigmoid has a left asymptote at $y = c_0$ and a right asymptote at $y = c_0 + c_1$. In order for the `curve_fit` function to work, it requires initial guesses for each of the parameters—these do not have to be particularly close, but merely have to set the algorithm in the right neighborhood of parameter space. We gave it the guess that $c_0$ is the smallest $u$ value, $c_0 + c_1$ is the largest $u$ value, $c_2$ is the median value of $\eta$, and $c_3$ is half the range of the $\eta$ values.

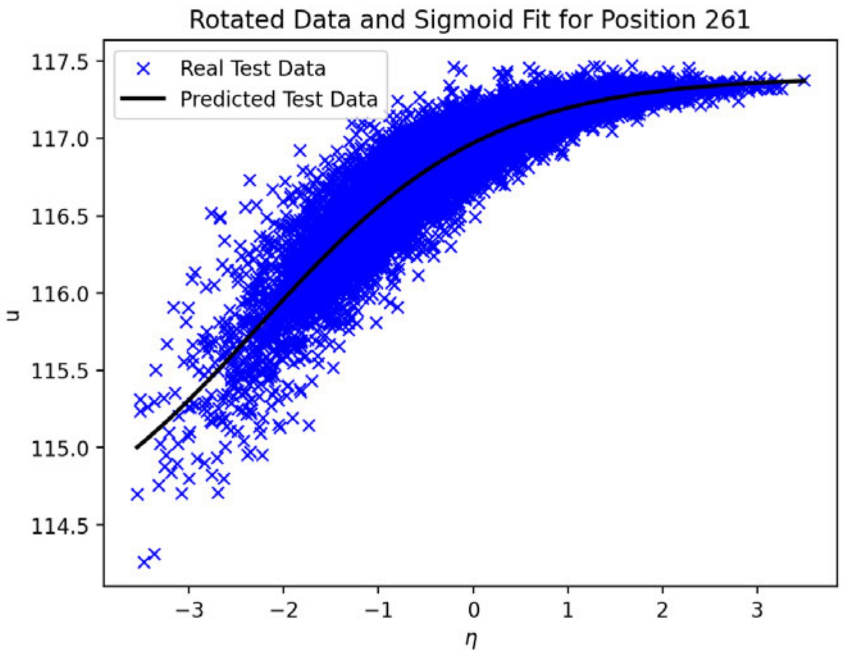


Figure 9. Sigmoid fit to data.

Using mean square error (MSE) as a metric, the sigmoid does not perform meaningfully better than a cubic, which is itself comparable to a linear fit under MSE. However, the sigmoid has the attractive properties that it is monotonic and will be bounded even if given a value of $\eta$ near the edge of the range. By comparison a different form of fitted curve may give predictions for $u$ that are too high (in the case of a line) or too low (in the case of a cubic that turns downward) in this case. We can see an example of this in Figs. 9 and 10. The sigmoid tapers off to the saturation value at the edge of the range, whereas the cubic begins to arc downward.

## 4 Fixing the Estimation

### 4.1 Regularization

In order to improve the calculation of $\boldsymbol{\xi}_*$, we choose a proper form for $\mathcal{R}$ to eliminate "undesirable" behavior. Unfortunately, there are philosophical disagreements as to what is most "undesirable". As a first step, we set $\mathcal{R}(\boldsymbol{\xi}) = ||\boldsymbol{\xi}||$, so (1.4) becomes

$$\boldsymbol{\xi}_* = \arg\min_{\boldsymbol{\xi}} ||\mathbf{u}_{\mathrm{w}} - \mathbf{u}(\boldsymbol{\xi})|| + \gamma||\boldsymbol{\xi}||. \tag{4.1}$$

In Fig. 11 we plot how the mean-squared error in $y$ varies with the regularization parameter. As one can see, when the parameter is 0, the MSE is large. However, as

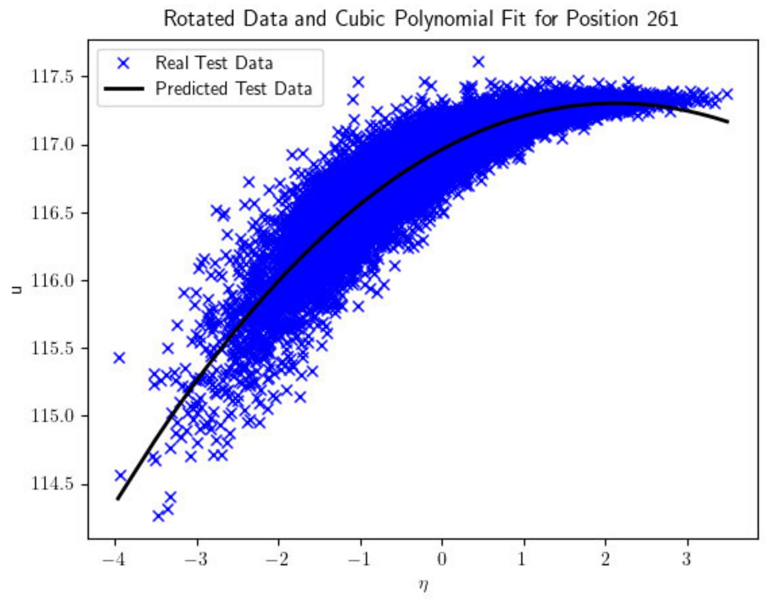


Figure 10. Cubic fit to data.

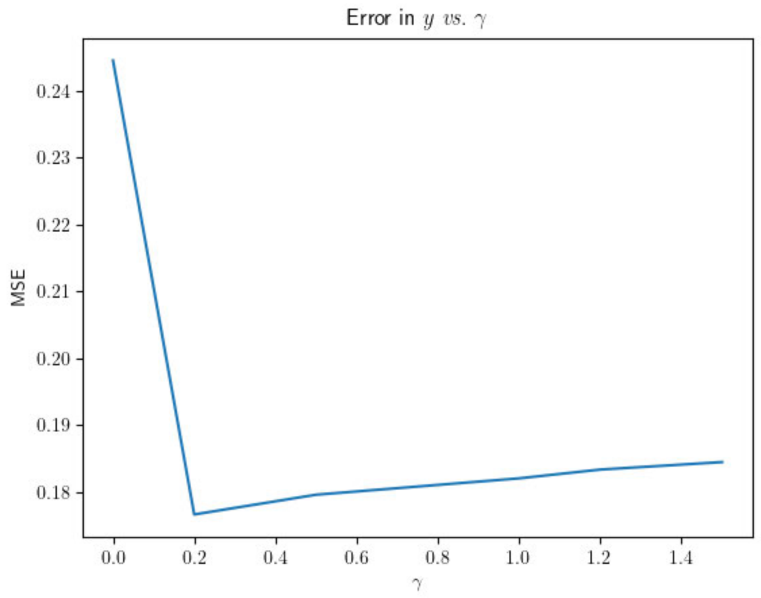


Figure 11. Error in $y$ *vs.* $\gamma$, unweighted case.

the parameter increases, the MSE also increases. Therefore, this means that we need a regularization term, but only a small one.

### 4.2 Weighting

Despite the fact that we are getting reasonably good estimates for $u$, the estimates for $y$ are still pretty bad. Another possibility would be to weight the individual well sites. Though somewhat hard to see from Fig. 1, the wells are clustered in certain locations, and spaced widely in others. It would seem natural that if we are able to match the pressure at one well, we would do a pretty good job estimating the pressure at nearby wells.

Therefore, we choose to weight the initial error calculation at each point by $\rho_j$, the distance from well $j$ to its nearest neighbor:

$$\boldsymbol{\xi}_* = \arg\min_{\boldsymbol{\xi}} \frac{\sum_{j=1}^{N_{\mathrm{w}}} \rho_j \left[u_{\mathrm{w},j} - u_j(\boldsymbol{\xi})\right]^2}{\sum_{j=1}^{N_{\mathrm{w}}} \rho_j} + \gamma \|\boldsymbol{\xi}\|. \tag{4.2}$$

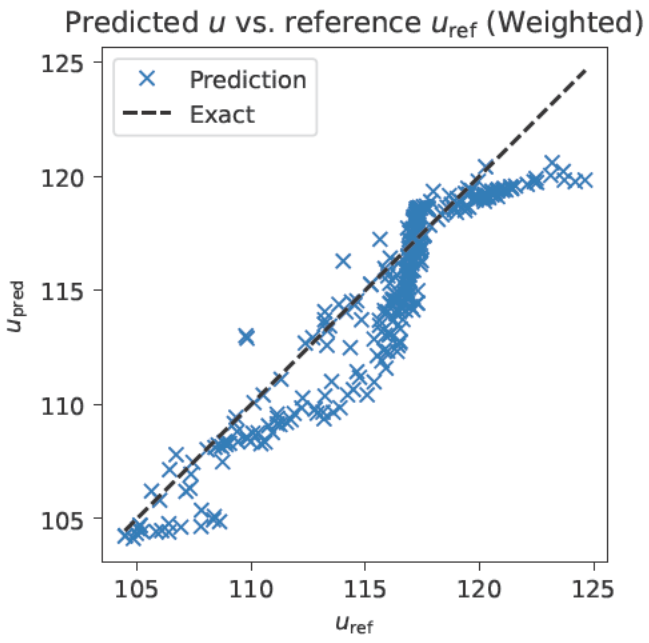


Figure 12. Scatter plot showing performance of weighted algorithm in estimating $u$.

Figure 12 is the analog of Fig. 5 for the weighted case, and Fig. 13 is the analog of Fig. 7 for the weighted case. Unfortunately, we do not see any appreciable change in the error or the calculations.

Figure 14 shows the error in $\log T$ for both the weighted and unweighted cases. The unweighted case is the analog of Fig. 11. The profile is somewhat different, as it comes from a different code. Note that the particular "nearest-neighbor" weighting chosen does not improve the results, though perhaps a different weighting would.

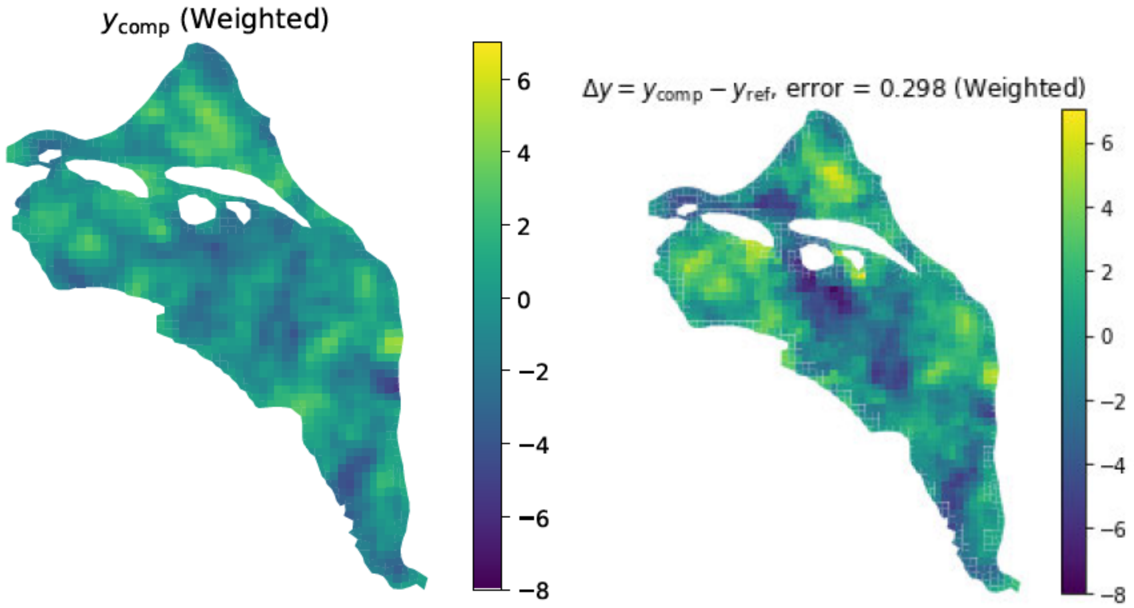

$y_{\text{comp}}$ (Weighted)
$\Delta y = y_{\text{comp}} - y_{\text{ref}}$, error = 0.298 (Weighted)
6
4
2
0
−2
−4
−6
−8

Figure 13. Left: Calculated $y$. Right: Error in calculation.

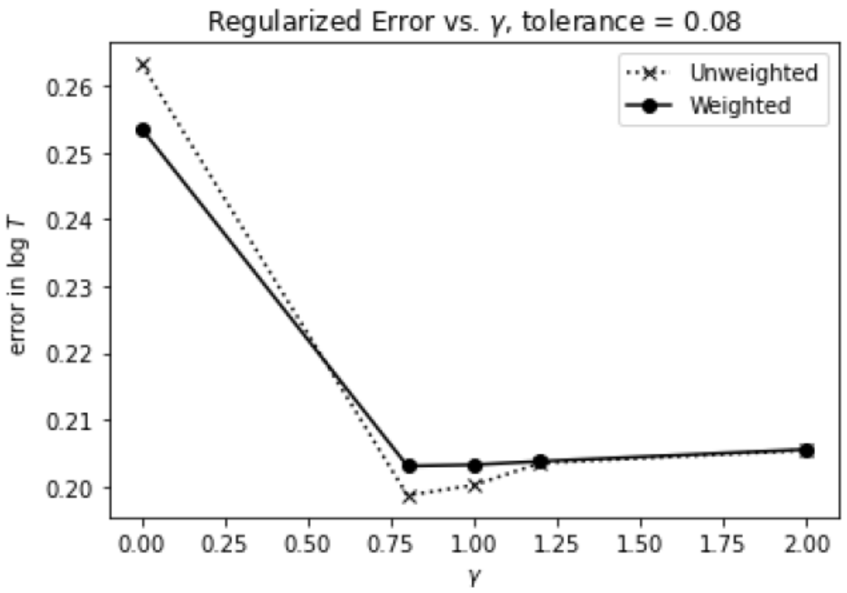


Figure 14. Error in $y$ *vs.* $\gamma$, different code.

## 4.3 Finding $\xi_*$

In order to improve the calculation of $\xi_*$, we could use a data-based approach. One example is the "one-out" method, which proceeds as follows:

(1) Construct a reduced vector $\mathbf{u}_{\mathrm{w},j}$ which contains every measurement *except* the $j$th one.

(2) Perform the optimization in (2.13) for the reduced set.

(3) Use the results to predict the value of $u_j$ and compute the error.

(4) Repeat the process for every $j$, returning a global error.

This global error will then allow us to compare different functional forms of $\mathcal{R}$ to find the best choice. Time ran out at the workshop before this subgroup could implement this algorithm; see §5.

## 5 More Regularizations and a Mapping Approach

Given the values of $N_\xi$ and $N_\mathrm{w}$ from the Appendix, we may define an *observation function* $\mathcal{F}$ which maps

$$\mathcal{F} : \mathbb{R}^{1000} \to \mathbb{R}^{323}, \tag{5.1}$$

$$(\xi_1, \ldots, \xi_{1000}) \mapsto (u(x_1), \ldots, u(x_{323})). \tag{5.2}$$

$\mathcal{F}$ can be evaluated by a numerical solver of the elliptic equation (1.2), and then evaluating the solution at the collocation points. Since this can be expensive, we instead build the surrogate model $\mathbf{u}(\boldsymbol{\xi})$ from a dataset of inputs and outputs to $\mathcal{F}$, $\mathcal{D} = \{\xi^{(i)}, u^{(i)}\}_{i=1}^{N_\mathrm{r}}$, where each $\xi^{(i)} \in \mathbb{R}^{1000}$ and each $u^{(i)} \in \mathbb{R}^{323}$. The surrogate model will then be used to solve the inverse problem of recovering the coefficients $\xi^\dagger$ from a particular set of observations of the pressure $u_{\mathrm{obs}} = \{u^\dagger(x_i)\}_i$

### 5.1 Surrogate Model

After trying different families of approaches (Gaussian process regression, polynomial regression, etc.), we observe that a linear ridge-regression model gives the best generalization error. Furthermore, its simple, analytical solution will be helpful in the inverse problem, as it will provide an analytical solution for the maximum *a posteriori* optimization problem as well.

We follow the general approach in §2.1, except that we regress with the linear map $x \mapsto A_* x$, where $A_*$ is defined as

$$A_* = \operatorname*{argmin}_A \|A\Xi - U\|_F^2 + \alpha \|A\|_F^2, \tag{5.3}$$

and $\|A\|_F^2$ is the Frobenius norm. (The work in §2.1 has $\alpha = 0$.) Equation (5.3) admits the explicit solution

$$A_* = (\Xi^T \Xi + \alpha I)^{-1} \Xi^T U. \tag{5.4}$$

Note that when $\alpha = 0$, (5.4) reduces to (2.8).

We do a 5-fold cross-validation procedure to choose $\alpha$, and observe that small $\alpha$ performs better, achieving a relative test $L^2$ loss of 0.5% (Fig. 15). The fact that small $\alpha$ performs better suggests that the linear map is a good approximation. We fix $\alpha = 1$, which improves numerical stability and the condition number of $A_*$. Training error is similar to testing error, suggesting no overfitting (as expected by a simple model).

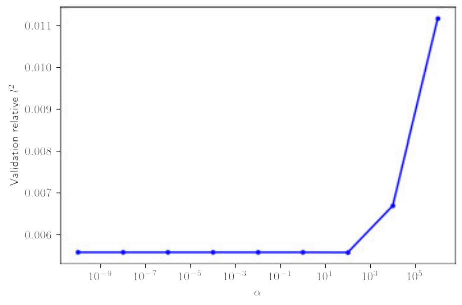


Figure 15. Validation relative error vs parameter $\alpha$ for the ridge regression surrogate model.

We fix the linear map $A_*$ as a surrogate model into the inverse problem solution step.

### 5.2 Inverse Problem

We use a maximum *a posteriori* (MAP) estimation approach, from a Bayesian perspective. We assume that the observation error is Gaussian, so that, if given access to $\mathcal{F}$, we would minimize

$$\underset{\boldsymbol{\xi}}{\operatorname{argmin}} \|\mathcal{F}(\boldsymbol{\xi}) - \mathbf{u}_{\mathrm{w}}\|_2^2 + \mathcal{R}(\boldsymbol{\xi}), \tag{5.5}$$

where $\mathcal{R}$ is a regularization term that prevents ill-posedness and can be interpreted as the prior contribution. In (5.5) we have absorbed the regularization factor $\gamma$ from (1.4) into $\mathcal{R}$; this is done so that we may consider more general forms below.

Since we aim to avoid evaluating $\mathcal{F}$, we replace it with our learned surrogate model $\hat{\mathcal{F}}(\boldsymbol{\xi}) = A_*\boldsymbol{\xi}$, and we experiment with different choices of regularization of the form $\mathcal{R}(\boldsymbol{\xi}) = \|R\boldsymbol{\xi}\|_2^2$, where $R$ is a matrix. This choice corresponds to the linear-Gaussian model inversion and provides an analytic solution to (5.5), making the two-step process equivalent to a Kalman filter. Indeed, we have

$$\begin{aligned} \boldsymbol{\xi}_* &= \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \|A_*\boldsymbol{\xi} - \mathbf{u}_{\mathrm{w}}\|_2^2 + \gamma\|R^{1/2}\boldsymbol{\xi}\|_2^2 \\ &= (A_*^T A_* + \gamma R)^{-1} A_*^T \mathbf{u}_{\mathrm{w}}. \end{aligned}$$

Natural choices of $R$ include:

(1) $I$ (the standard $L^2$ penalty used in §4.1).

(2) A diagonal matrix (weighted $L^2$).

(3) The matrix of basis functions $\Psi$, where $\psi_{ij} = \psi_j(\mathbf{x}_i)$. This roughly penalizes $L^2$ loss of $y$.

(4) $L\Psi$, where $L$ is the matrix of finite differences. Note this roughly penalizes the gradient of $y$.

(5) Regularizations of the form

$$\mathcal{R}(\boldsymbol{\xi}) = \gamma_1\|R_1^{1/2}\boldsymbol{\xi}\|_2^2 + \cdots + \gamma_s\|R_N^{1/2}\boldsymbol{\xi}\|_2^2$$

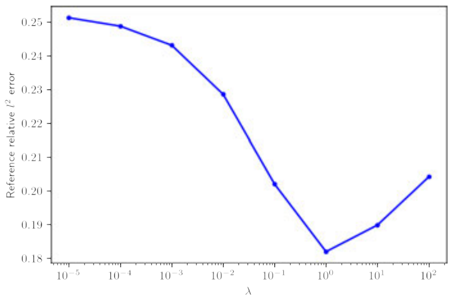


Figure 16. Predicted *vs.* true $y(\mathbf{x})$ for weighted $L^2$ regularization. Here $\gamma$ is denoted as $\lambda$.

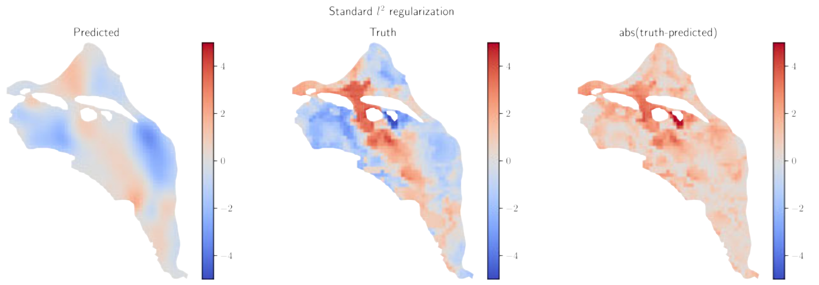


Figure 17. Predicted *vs.* true $y(\mathbf{x})$ for standard $L^2$ regularization.

are also possible, with analytic solution

$$\boldsymbol{\xi}_* = (A_*^T A_* + \gamma_1 R_1 + \cdots + \gamma_N R_N)^{-1} A_*^T \mathbf{u}_{\mathrm{w}}. \tag{5.6}$$

After recovering a predicted $\boldsymbol{\xi}$, we use the KKL expansion formula to compute $y$ in a grid and report the relative $L^2$ error with respect to the reference, as defined by (2.14).

For the first experiment, we fix $R = I$ (option #1) and choose $\gamma = 1$ by a cross-validation procedure, over the data and a reference pair of transmissivity field and pressure (Fig. 16). We observe from the predicted $y(\mathbf{x})$ (Fig. 17) and the predicted $\boldsymbol{\xi}$ (Fig. 18) that the recovered solution is too smooth and does not recover correctly the higher modes of $\boldsymbol{\xi}$. We note that the reference transmissivity field does not have $\boldsymbol{\xi}$ values as it does not come from a Gaussian process realization, but we approximate them by a least-squares procedure.

As a solution, we propose a weighted $L^2$ regularization loss (option #2), with the diagonal entries decaying as $r_{ii} \propto i^{-k}$, with $k$ also learned from cross-validation to be 0.7. We observe that the recovery improves, is less smooth (Fig. 19) and more accurately recovers the higher frequency modes (Fig. 20).

We finally experimented with a weighting of $y\boldsymbol{\xi}$, $yL\boldsymbol{\xi}$ and the first term to $\|A_*\boldsymbol{\xi} - \mathbf{u}_{\mathrm{w}}\|_\Sigma$, where $\Sigma$ is a Matérn covariance matrix of the spatial location of the $x_i$, which marginally improved performance. We present the best relative error results of different approaches in Table 1.

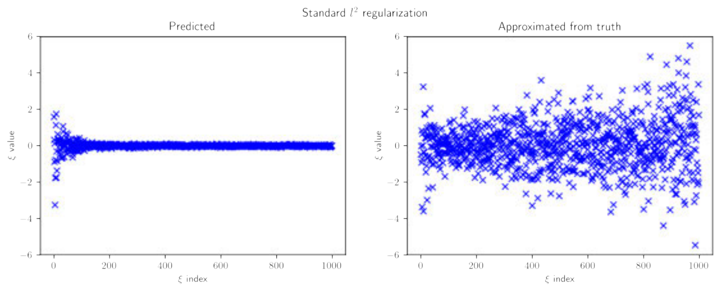


Figure 18. Predicted *vs.* approximated from truth $\xi$ for standard $L^2$ regularization.

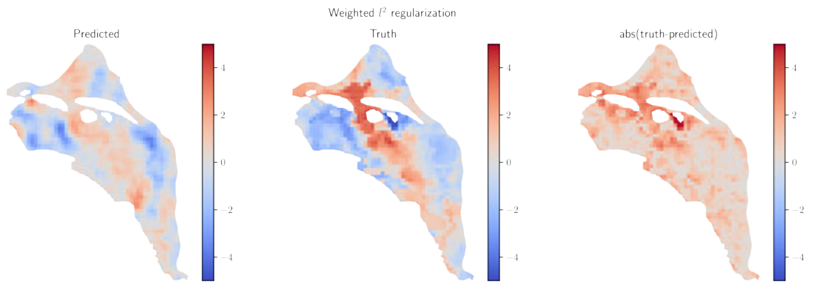


Figure 19. Predicted *vs.* true $y(\underline{x})$ for weighted $L^2$ regularization.

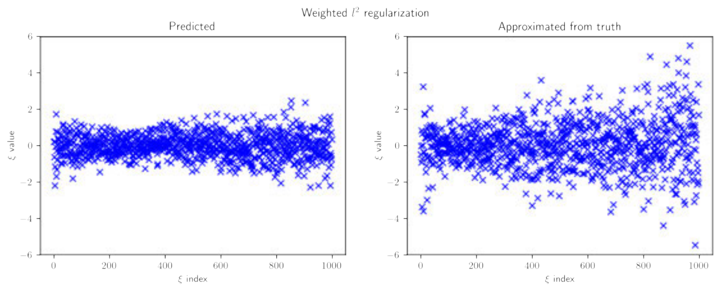


Figure 20. Predicted *vs.* approximated from truth $\xi$ for weighted $L^2$ regularization.

## 6 Neural network implementation

### 6.1 PyTorch

In this section, we explore the use of PyTorch for calculation of a surrogate model for $u(\xi)$. PyTorch is a popular package in Python whose general purpose is aligned with machine learning tasks, and built in particular to simplify constructing and fitting neural networks at a fairly high (simplified) level. It provides tensor classes for array storage, and automatic construction of computational graphs, loss functions, activation functions, and general mathematical functions, which combine with seamless automatic differentiation, to perform both forward and back-propagation calculations in feed-forward neural networks. It is valuable to know that this software is not necessarily restricted to neu-

| Approach | Best relative $L^2$ loss |
|---|---|
| Uniform $L^2$ | 0.182 |
| Weighted $L^2$ | 0.176 |
| Weighted $L^2$ + Covariance weights | 0.172 |
| Weighted $L^2 + y$ weights | 0.169 |

Table 1. Relative $L^2$ of different approaches.

ral networks, however; a wide variety of optimization tasks can be done as long as the optimizable parameters and loss function are specified.

To illustrate, let `loss` is a torch variable which calculates the value of an arbitrary loss (*e.g.*, a two-norm-based error on the prediction(s))

$$\mathcal{L}(\boldsymbol{\xi}) = ||\mathbf{u}_{\mathrm{w}} - \mathbf{u}(\boldsymbol{\xi})||, \tag{6.1}$$

as in (1.3), or some version with regularization

$$\mathcal{L}(\boldsymbol{\xi}) = ||\mathbf{u}_{\mathrm{w}} - \mathbf{u}(\boldsymbol{\xi})|| + \gamma\mathcal{R}(\boldsymbol{\xi}), \tag{6.2}$$

as in (1.4). Then calculations of gradients of the loss with respect to the loss $\nabla_{\boldsymbol{\xi}}\mathcal{L}$ and subsequent update of the collection of parameters from $\boldsymbol{\xi}_k$ to $\boldsymbol{\xi}_{k+1}$ are achieved via `loss.backward()` (for the gradient) and `loss.step()` (for an iterative update) respectively. PyTorch also exposes a wide range of optimization update schemes via `torch.optim.SGD`, whose parameters allow one to specify many gradient-based methods (not just Stochastic Gradient Descent).

Given some of the challenges specific to this problem, such as exploration of regularizer, uncertain network structure, and minimization tasks, we found the benefits of using PyTorch for black box optimization appealing. We describe the details of implementation and preliminary results in this section.

### 6.2 Implementation

For the remainder of this section, $\boldsymbol{\xi}$ refers to the collection of all parameters available to be optimized, regardless of the details of the neural network architecture.

Broadly, the question of the precise structure (number of layers and dimensionality of each layer) of a neural network, given the application problem, has little to no mathematical grounding. Rather, it is a blend of heuristics, trial and error, and incorporation of domain-specific knowledge. For example, when neural networks are applied to image processing, there is a notion of implied correlation between features (dimensions in the data). Pixels are usually located on a rectangular grid, and there is some degree of continuity in color-value space (loosely defined). The successful end result in that domain is convolutional neural networks (see, e.g. [11]). In our case, given the time constraints of the MPI workshop, we acknowledge that there are two main avenues to apply domain specific knowledge that we **have not done**.

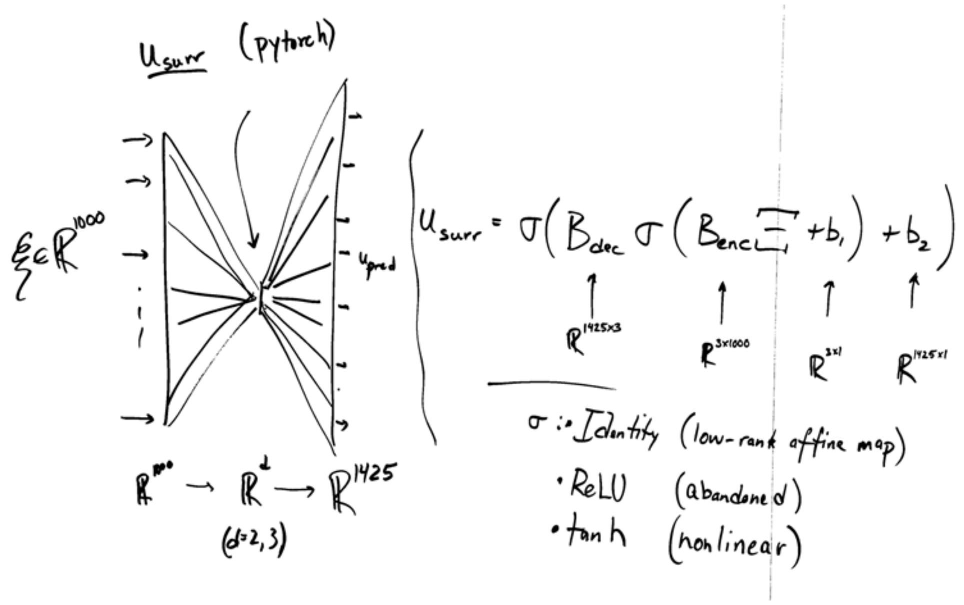

$u_{surr}$ (pytorch)
$\xi \in \mathbb{R}^{1000}$
$u_{pred}$
$u_{surr} = \sigma(B_{dec}\,\sigma(B_{enc}\Xi + b_1) + b_2)$
$\mathbb{R}^{1425\times3}$
$\mathbb{R}^{3\times1000}$
$\mathbb{R}^{3\times1}$
$\mathbb{R}^{1425\times1}$
$\sigma$ :• Identity (low-rank affine map)
• ReLU (abandoned)
• tanh (nonlinear)
$\mathbb{R}^{1000} \to \mathbb{R}^{d} \to \mathbb{R}^{1425}$
$(d=2,3)$

Figure 21. Schematic illustrating surrogate model structure in PyTorch. $u_{\text{surr}}$ is denoted $\mathbf{u}$ elsewhere.

Firstly, similar to image processing, we expect that in truth, there are strong correlations between a feature and its spatial neighbors (whether or not the domain is a rectangle). Secondly, the input/output pairs are expected to obey a physical law – transmissivity values $T$ are coupled to pressure values $u$ via (1.2). The field of *physics-informed neural networks* attempts to exploit implied knowledge to improve performance here; see for example [3].

We restrict to a shallow "bottleneck" mapping illustrated in Fig. 21. The hypothesis underlying this is that the relationship between collections of input and output variables should not depend on the resolution of the number of terms in the KKL expansion (representing the "1000" of the input dimension) nor the resolution of the mesh (representing the "1425" of the output dimension). The precise dimension of the mapping for this problem, or a general inverse problem for this type of partial differential equation, or more broadly a PDE with $n$ spatial variables, $m$ unknowns, etc., is unknown to us. However, we note that methods and heuristics exist which attempt to infer the dimension of an abstractly defined manifold rather than declaring it arbitrarily. While sophisticated methods exist, the most accessible for future work is a brute-force attempt, increasing dimensions until one sees diminishing returns in numerical performance. These are directions for future work.

## 6.2.1 *Functional form*

In our case, the function takes the form

$$\mathbf{u}(\boldsymbol{\xi}) = \boldsymbol{\sigma}\left(B_{\text{dec}}\,\boldsymbol{\sigma}(B_{\text{enc}}\boldsymbol{\xi} + \mathbf{b}_1) + \mathbf{b}_2\right), \tag{6.3}$$

where $\boldsymbol{\xi} = \{B_{\text{enc}}, B_{\text{dec}}, \mathbf{b}_1, \mathbf{b}_2\}$ are the argminimizable parameters. The dimension of the latent space is denoted $d$—understood to be small, and chosen as $d = 3$ in our numerical experiments. The pair $B_{\text{enc}} \in \mathbb{R}^{d\times 1000}$ and $\mathbf{b}_1 \in \mathbb{R}^d$ (the "encoding" stage) map input data to a low-dimensional latent space $W \subset \mathbb{R}^d$. Similarly, the "decoding" stage maps $W$ to $\mathbb{R}^{1425}$ via $B_{\text{dec}} \in \mathbb{R}^{1425\times d}$ and $\mathbf{b}_2 \in \mathbb{R}^{1425}$.

### 6.2.2 *Choice of activation function*

We will later explore the choice of "activation function" $\boldsymbol{\sigma}(\mathbf{z})$ in the neural network; in the code this is chosen by the user as one of

$$\boldsymbol{\sigma}(\mathbf{z}) = \text{Id}(\mathbf{z}) \equiv \mathbf{z}, \tag{6.4a}$$

$$\boldsymbol{\sigma}(\mathbf{z}) = \text{ReLU}(\mathbf{z}) \;=\; \text{Id}_{\mathbf{z}>0} \equiv \mathbf{z}^+ \equiv \mathbf{z}H(\mathbf{z}), \tag{6.4b}$$

$$\boldsymbol{\sigma}(\mathbf{z}) = \tanh(\mathbf{z}). \tag{6.4c}$$

Here $H(\mathbf{z})$ is the Heaviside function. When the input to $\boldsymbol{\sigma}$ is vector-valued, the operation acts entry-wise and outputs in the same space.

It is important to note that the observed values of outputs $\mathbf{u}(\boldsymbol{\xi})$ should conform to the structure of this function. For example, use of $\boldsymbol{\sigma}(\mathbf{z}) = \tanh(\mathbf{z})$ combined with the form of (6.3) requires outputs to lie in $(-1, 1)$, and $\boldsymbol{\sigma}(\mathbf{z}) = \text{ReLU}(\mathbf{z})$ necessitates outputs are nonnegative. Modification of the functional form and/or preprocessing of the original input/output pairs can address this. We did not explore other functional forms during the MPI week, but this is an important direction for future work.

### 6.2.3 *Training and evaluation of loss*

The data provided by the sponsors is of complete $(\boldsymbol{\xi}, \mathbf{u})$ pairs: $\boldsymbol{\xi} \in \mathbb{R}^{1000}$, $\mathbf{u}(\boldsymbol{\xi}) \in \mathbb{R}^{1425}$, representing the full discretization of the Hanford Basin. To address real-life applicability, measurements for transmissivity, pressure, and so on, are observable only at a few sparse locations (water wells); while the full transmissivity, pressure, etc. fields were calculated using a high-fidelity solver prior to the workshop.

We implement evaluation of loss of the pairs either across the entire domain, or only at those wells, based on metadata provided by the sponsor. Our preliminary results do not include the nuanced regularization necessary to produce globally accurate results; the distinction comes down to (6.2), where we use measurements only at the well sites (as described in §1) or over the full 1425 mesh cells.

### 6.2.4 *Details of training and parameter choices*

A custom class in Python was written extending the template `torch.nn.Module` which allows one to define the set of optimizable parameters $\boldsymbol{\xi}$ on class instantiation. We additionally define $\boldsymbol{\sigma}$ at that stage, as well as a nominal identity function (directly returning the input) to study strictly affine functional forms. The `forward()` function directly computes (6.3). A function `fit()` which we have implemented in the class follows the `scikit-learn` style methodology of taking training data ($n$ instances of input/output

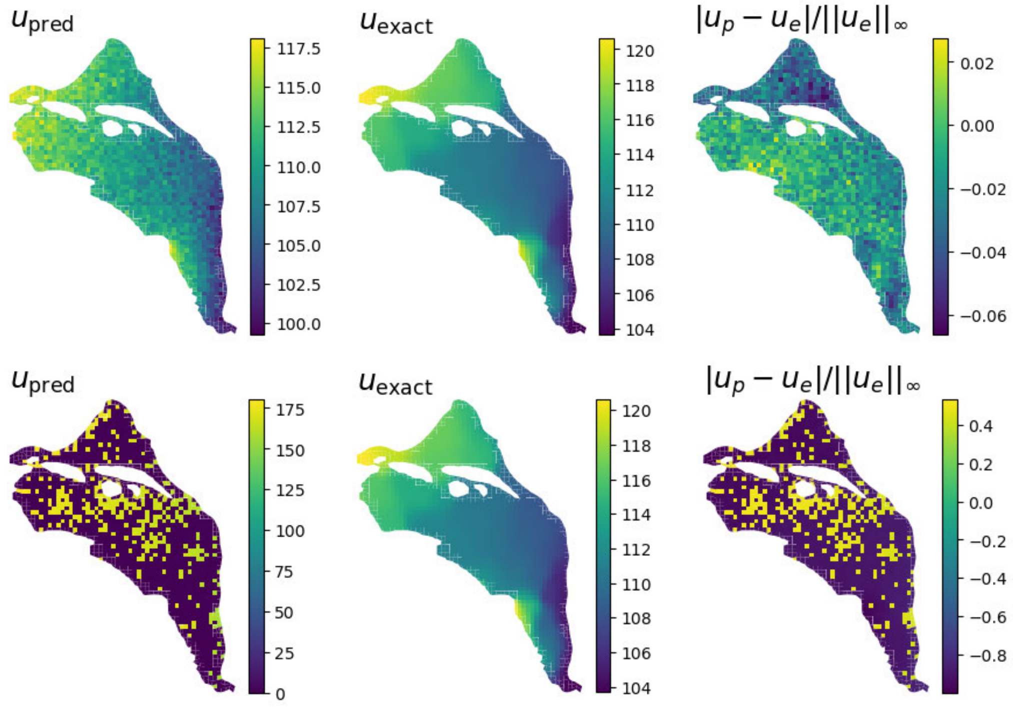


Figure 22. Top: example out-of-sample prediction based on low-rank linear mapping fit via PyTorch. Bottom: same prediction based on shallow bottleneck feed-forward neural network. ReLU activations used. Without careful choice of regularization, model appears to attempt to fit values only at wells. Here, a 2-norm regularization penalty is applied in the loss for the coefficients of the weights in the neural network.

pairs) and performing the full optimization process and storing the fitted model within the object. Descent parameters include the learning rate (default: 0.1), regularization parameter (default: $10^{-4}$), and number of training iterations (default: 100). Several user-controlled switches also exist to record information about the training (optimization) process, such as loss values, the parameter set, etc.

## 6.3 Results

The first experiment was to set $\boldsymbol{\sigma}(\mathbf{z}) = \mathrm{Id}(\mathbf{z})$, given the structure of the underlying problem. In this case, direct distribution of the matrices leads to an implied structure

$$
\begin{aligned}
\mathbf{u}(\boldsymbol{\xi}) &= B_{\text{dec}} B_{\text{enc}} \boldsymbol{\xi} + B_{\text{dec}} \mathbf{b}_1 + \mathbf{b}_2 \\
\mathbf{u}(\boldsymbol{\xi}) &= \widetilde{B} \boldsymbol{\xi} + \widetilde{\mathbf{b}},
\end{aligned}
\tag{6.5}
$$

which is a linear (strictly: affine) mapping which will explicitly be rank-$d$. When the matrices are not restricted to be a certain rank, several classes of minimization problems (6.2) have explicit solutions in terms of algebraic operations on $\boldsymbol{\xi}$ and associated $\widehat{\mathbf{u}}$.

The second experiment chooses $\boldsymbol{\sigma}(\mathbf{z}) = \mathrm{ReLU}(\mathbf{z})$, as well as evaluating pointwise error at the well sites only. The trained model is relatively "sparse" in the sense that only 933 of the $1425 \times 3 = 4275$ trained weights were numerically nonzero. Even at the well sites, where the loss function directly penalizes pointwise errors, performance was not nearly as successful; we observe a relative error in excess of 40% after the same training schedule as with the affine formulation ($\boldsymbol{\sigma} = \mathrm{Id}$) which we believe may the symptom of this sparsity in output prediction values.

## 6.4 Conclusions

A framework was implemented in PyTorch, and its basic functionality was validated when the functional form of the system is reduced to a the class of linear (strictly: affine) mappings. The flexibility in our implementation allows us to, most simply, swap out the activation function in one line and explore nonlinear approaches. As we discovered, producing a high-quality result is not quite as simple as this. Careful formulation of the loss is likely required—such as in penalizing a numerical gradient of the prediction, or penalizing total variation of the prediction, to incentivize smooth solutions.

More extensively, we understand there is an art in construction of the hyperparameters of a neural network as well as how the model is trained. Facets such as the number of layers, dimensionality of each layer, and activation function for each layer can all have significant impact on the quality of the resulting model. Similarly, aspects of how the model is trained – the choice of optimizer step, presentation of training examples, and variable learning rate, to name a few, may be the determining factor in finding the theoretical global optimal parameter set $\boldsymbol{\xi}$ for a fixed network structure.

In summary, what we have produced here is primarily a proof of concept of a pipeline which an energetic researcher may quickly explore with time and more responsive computing resources than a personal laptop. We expect higher quality solutions may be lurking in a "simpler" corner of problem space, but we leave this search for future work.

## 6.5 Code

The current version of the code exists as a "fork" of Dr. Barajas-Solano's GitHub repository provided for the workshop, located at `https://github.com/maminian/pnnl_mpi23`. Please reach out with any questions.

## 7 Alternative dimension reduction techniques

Currently, as stated in §2.1, the method proposed by the PNNL uses simple linear regression (*i.e.*, least squares) to find "effective directions," *i.e.*, vectors $\hat{\mathbf{a}}_j$ such that at each location $x_j$,

$$u(x_j) \approx \hat{\mathbf{a}}_j^T \boldsymbol{\xi} \equiv \eta_j. \tag{7.1}$$

Note the similarity between (7.1) and (2.6).

However, the scatter plots $u(x_j)$ vs $\eta_j$ presented in §2 suggest that a single effective direction per location may not be enough. Therefore, there will be a benefit in determining *several* effective directions, $\mathbf{a}_j^{(k)}$, such that at each location $j$:

$$u(x_j) \approx f_j([\hat{\mathbf{a}}_j^{(1)}]^T \boldsymbol{\xi},\ [\hat{\mathbf{a}}_j^{(2)}]^T \boldsymbol{\xi}, \ldots). \tag{7.2}$$

Here $f_j(\cdot)$ is an unknown nonlinear function, determined in a separate step. (That issue is discussed in §§2.2 and 3.2.) Note that (7.2) generalizes (2.2).

Below we describe two techniques to achieve this: Sliced Inverse Regression (SIR) [9] and Reduced-Rank Regression (RRR) [7]. As a preview, while SIR appears to be directly applicable to the problem at hand, it is not clear whether RRR has relevance to the current problem; this will be discussed in §7.3.

## 7.1 Sliced Inverse Regression

In a nutshell, the directions $\hat{\mathbf{a}}_j^{(k)}$ can be found *not* by viewing $u(\mathbf{x})$ as a function of $\boldsymbol{\xi}$ but, conversely, examining the dependence of the mean and covariance of $\boldsymbol{\xi}$ on $u(\mathbf{x})$. For brevity, in this subsection we simply write $u$ for $u(\mathbf{x})$. Thus, the algorithm presented below will find "effective directions" separately for separate locations.

The algorithm presented below is the same (or similar) to that found in Wikipedia [13] or in many existing references on SIR. The main contributions of this group are notes and some explanations.

**SIR Algorithm**

(1) For $N_\mathrm{r}$ realizations of the random vector $\boldsymbol{\xi}$, compute $N$ samples of $u$. You will get a range of those values with $\min(u) \equiv u_0$ and $\max(u) \equiv u_{\max}$. Subdivide the interval $[u_0, u_{\max}]$ into $h_{\max}$ slices. The slices do not have to be the same width, but it is probably easiest to take them uniform. Also, the exact value of $h_{\max}$ is not too important: according to [9], changing $h_{\max}$ within bounds that appear intuitively reasonable (*e.g.*, having $h_{\max} \sim 50$) changes the directions found by units of percent. However, the authors of [12] state that the results are sensitive to $h_{\max}$, although they do not present evidence for that statement. Note that if one takes $h_{\max}$ to be "too large," one will end up having too few samples per slice (defined above), which will be a problem for the next step.

(2) For each slice $h$, $1 \leq h \leq h_{\max}$, compute the average

$$\bar{\boldsymbol{\xi}}_h = \frac{1}{N_h} \sum_{i=1}^{N_h} \boldsymbol{\xi}_i, \tag{7.3}$$

where $N_h$ is the number of samples in slice $h$ and the summation is over all samples in that slice. Note that the set of $h_{\max}$ points $\bar{\boldsymbol{\xi}}_h$ form a (discretized) curve in $\mathbb{R}^{N_\xi}$. Denote this curve $\mathbf{m}(u)$. (Here '$\mathbf{m}$' stands for 'mean'.)

**Claim**: Under the assumption stated below, the curve $\mathbf{m}(u)$ lies in the subspace of $\mathbb{R}^{N_\xi}$ spanned by the effective direction vectors $\hat{\mathbf{a}}^{(k)}$, where the index $k$ enumerates these directions.

**Assumption**: For any $\mathbf{z} \in \mathbb{R}^{N_\xi}$, the conditional expectation value (on the left hand side of the equation below) satisfies the *linearity condition*:

$$\mathbb{E}\big(\mathbf{z}^T \boldsymbol{\xi} \mid [\hat{\mathbf{a}}^{(1)}]^T \boldsymbol{\xi},\ [\hat{\mathbf{a}}^{(2)}]^T \boldsymbol{\xi}, \ldots\big) = \mathbf{c}_0 + \mathbf{c}_1 [\hat{\mathbf{a}}^{(1)}]^T \boldsymbol{\xi} + \mathbf{c}_2 [\hat{\mathbf{a}}^{(2)}]^T \boldsymbol{\xi} + \cdots \tag{7.4}$$

for some constant vectors $\mathbf{c}_k \in \mathbb{R}^{N_\xi}$.

*Note SIR-1*: According to [9], property (7.4) holds for $\boldsymbol{\xi}$ whose components have a joint normal distribution or, more generally, whose joint PDFs have elliptic (*i.e.*, ellipse-shaped) level surfaces. In [1], the authors argue that the outcome of the SIR algorithm is not too sensitive to that condition being rigorously satisfied, or at least the samples can be "groomed" before being processed to have that condition satisfied approximately. A different method that bypasses the need for the linearity condition (7.4) is discussed in Note SIR-7 after the end of this algorithm.

The reader may skip the next two notes without impacting their understanding of the algorithm.

*Note SIR-2*: The proof of the Claim may be found in [9, Thm. 1], but it appears impenetrable without good knowledge of statistics. The proof for the case of only one direction $\hat{\mathbf{a}}^{(1)}$ from [5] is a little more clear but was still not fully understood by the author of this section (T.I. Lakoba).

*Note SIR-3*: Since this author could not understand the details of the proof of the Claim, he resorted to an illustrating example. Let $\boldsymbol{\xi} \in \mathbb{R}^3$ be a vector of random components with zero mean each, and consider

$$u = (\xi_1 - \xi_2) + 0.5\xi_2^2. \tag{7.5}$$

The two directions here, $\hat{\mathbf{a}}^{(1)} = [1, -1, 0]^T$ and $\hat{\mathbf{a}}^{(2)} = [0, 1, 0]^T$, span the $\xi_1\xi_2$-plane.

The slices defined in Step 1 of the Algorithm are 3D regions between parabolic cylinders defined by equations $u_h = (\xi_1 - \xi_2) + 0.5\xi_2^2$, which are perpendicular to the horizontal plane. Since $u$ does not depend on $\xi_3$ and since $\mathbb{E}(\xi_3) = 0$, then the point $[(\overline{\boldsymbol{\xi}}_1)_h, (\overline{\boldsymbol{\xi}}_2)_h, (\overline{\boldsymbol{\xi}}_3)_h]$ has $(\overline{\boldsymbol{\xi}}_3)_h = 0$, where the barred quantities are defined in (7.3). Thus, the discretized curve composed of all such points indeed lies in the $\xi_1\xi_2$-plane, spanned by $\hat{\mathbf{a}}^{(1)}$ and $\hat{\mathbf{a}}^{(2)}$.

(3) From the curve $\mathbf{m}(u)$ obtained at the previous step, the effective directions are computed in three substeps.

/S1/ Center the curve at the origin by subtracting the unconditional mean of $\boldsymbol{\xi}$:

$$\mathbf{m}_{\rm c}(u) = \mathbf{m}(u) - \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\xi}_i. \tag{7.6}$$

Recall that this is a discretized curve with $h_{\max}$ points.

/S2/ Construct the covariance matrix of this curve:

$$\Sigma_{\rm m} = \sum_{h=1}^{h_{\max}} \frac{N_h}{N} (\mathbf{m}_{\rm c})_h (\mathbf{m}_{\rm c}^T)_h, \tag{7.7}$$

where $(\mathbf{m}_{\rm c})_h$ is the point of the curve located in slice $h$ (*i.e.*, the centered—in the sense of (7.6)—version of (7.3)). In other words, the $ij$th entry of the $N_\xi \times N_\xi$

matrix $\Sigma_{\mathrm{m}}$ is:

$$\Sigma_{m,\,ij} = \sum_{h=1}^{h_{\max}} \frac{N_h}{N}\,(m_{c,i})_h (m_{c,j}^T)_h, \tag{7.8}$$

where $(m_{c,i})_h$ is the $i$th component of the $N_\xi$-dimensional vector $(\mathbf{m}_c)_h$.

*Note SIR-4*: It may be relevant to point out here in what sense matrix (7.7) is called a 'covariance' matrix. The averaging (required to make a covariance matrix) applied *over the outer product* is over different $u_h$-values. The statistical averaging over all samples leading to a set of outcomes falling into the interval $[u_{h-1}, u_h]$ has been done earlier, when obtaining the curve $\mathbf{m}_c$.

/S3/ The $k_{\max}$ dominant effective direction vectors are found as the $k_{\max}$ largest eigenvalues of the generalized eigenvalue problem

$$\Sigma_{\mathrm{m}} \hat{\mathbf{a}}^{(k)} = \lambda^{(k)} \Sigma_\xi \hat{\mathbf{a}}^{(k)}, \tag{7.9}$$

where $\Sigma_\xi$ is the simple covariance matrix of $\boldsymbol{\xi}$:

$$\Sigma_\xi = \mathbb{E}\left((\boldsymbol{\xi} - \mathbb{E}(\boldsymbol{\xi}))(\boldsymbol{\xi} - \mathbb{E}(\boldsymbol{\xi}))^T\right), \tag{7.10}$$

and $\mathbb{E}(\boldsymbol{\xi})$ is the last term in (7.6).

Similarly to what is stated in Note SIR-2 above, the proofs of substep /S3/ for $k_{\max} = 1$ [5] and for general $k_{\max}$ [9] require more knowledge of statistics than the author of this section has.

*Note SIR-5*: It may appear that Substep /S3/, which requires solving an $N_\xi \times N_\xi$ eigenvalue problem may be prohibitively expensive as it requires $O(N_\xi^3)$ operations. However, in practice one needs only the first few dominant eigenvectors, which may require only $O(N_\xi^2)$ operations.

*Note SIR-6* There are some situations, such as when the function $f$ in (7.2) has certain symmetry (e.g., $f(\eta) = \eta^2$), when the SIR method fails [2]. However, it does not seem to be a concern for the PNNL problem, where the functions are only weakly nonlinear.

*Note SIR-7* If the linearity condition (7.4) ever becomes an issue, there is a technique (MAVE) developed in [15]. Section 2 of [15], where the technique is explained, is actually quite readable. The main idea appears to be this. In the vicinity of each input vector $\boldsymbol{\xi}_i$ leading to a value $u$ in slice $h$, the linearity condition (7.4) holds (probably, by the argument that locally, joint PDFs can be approximated as normal). This is expressed by the equation preceding [15, (2.5)]. Then a certain minimization problem, (2.6) in [15], can be stated near each $\boldsymbol{\xi}_i$ point. Averaging the expressions of such problems over all points within one slice yields a minimization problem (2.7) in [15], the solution of which produces the desired effective directions. The only part that remains unclear to this author is the notation $\sigma_B$ introduced in [15, (2.2)] (to what does the subscript $B$ refer?).

[12] claims to have combined MAVE with SIR, but its presentation is such that this author was unable to understand what improvement [12] made over what has already been done in [15].

## 7.2 Kernel Sliced Inverse Regression

Kernel sliced inverse regression (KSIR) is a method first proposed by [14] and almost concurrently expanded on by [16]. KSIR builds on SIR by simply replacing the inner product, implied by the model given by (7.4), with an inner product defining a so-called reproducing kernel Hilbert space (RKHS). KSIR is a more robust method than SIR since there are examples where SIR fails to identify symmetric patterns. The implementation and more about the advantages of KSIR will be discussed in this section, but first, we will discuss the formalism precisely. Note that throughout this section we deviate from the previous subsection as we suppress the dependence on $j$ made explicit by (7.1) and (7.2) for ease of notation. **Also the usage of $x$ and $y$ are not the same as elsewhere in the report**.

Let $X$ be an arbitrary set and $H$ a Hilbert space of real-valued functions on $X$, equipped with the usual notions of pointwise addition and multiplication. We define an evaluation functional $L_x$ over $H$ as a linear functional that evaluates each function at a point $x$, that is, $L_x : f \mapsto f(x)$, for all $f \in H$. We say that $H$ is a *reproducing kernel Hilbert space* if $L_x$ is a continous operator, or equivalently if the operator is bounded [10].

In practice, since $L_x$ is a bounded, linear operator, we can apply the Riesz representation theorem, see [10, ch. 2] or any standard text on functional analysis. That is, Riesz's theorem guarantees we may represent the operator as an inner product of $f$ with a unique function $K_x \in H$ as follows:

$$f(x) = L_x(f) = \langle f, K_x \rangle_H \,, \quad \forall f \in H, \tag{7.11}$$

where $\langle \cdot, \cdot \rangle_H$ is the inner product defining the space $H$. Since $K_x$ is itself a function defined on $X$ with values in $\mathbb{R}$, we have that

$$K_x(y) = L_y\left(K_x\right) = \langle K_x, K_y \rangle_H \,.$$

Applying this pointwise in $X$ allows us to define the reproducing kernel of $H$ as a function $K : X \times X \to \mathbb{R}$ by

$$K(x, y) = \langle K_x, K_y \rangle_H \,. \tag{7.12}$$

If in addition the kernel $K$ is positive definite and we assume that the Hilbert space is separable, as is often the case in applied data science, then we can more concretely understand the reproducing kernel via its spectral decomposition

$$K(x, y) := \sum_{k=1}^{d} \lambda_k \phi_k(x) \phi_k(y), \qquad d \leq \infty. \tag{7.13}$$

The main idea of KSIR is then to first map data from some input space $\mathcal{X} \subset \mathbb{R}^p$ into the spectrum-based feature space $\mathcal{H}$ via the transformation $\Phi$:

$$\mathbf{x} \mapsto \mathbf{z} := K(\mathbf{x}) := \left(\sqrt{\lambda}_1 \phi_1(\mathbf{x}), \sqrt{\lambda}_2 \phi_2(\mathbf{x}), \dots\right)^T,$$

where $\mathbf{x} \in \mathcal{X}$. Of course, this transformation is related to the kernel function in the sense that $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle_{\mathcal{H}} = K(\mathbf{x}, \mathbf{y})$ implied by (7.13). Now, the regression model analogous to SIR in the feature space $\mathcal{H}$ is given by

$$y = f\left(\mathbf{a}_1^T \Phi(\mathbf{x}), \dots, \mathbf{a}_{\max}^T \Phi(\mathbf{x})\right) = f\left(\mathbf{a}_1^T \mathbf{z}, \dots, \mathbf{a}_{\max}^T \mathbf{z}\right),$$

where $\mathbf{a}_k \in \mathbb{R}^d, d \leq \infty$.

The following observation is key: the reproducing kernel function $K(\mathbf{x}, \mathbf{y})$ is defined directly by (7.12) and so the possibility to apply the SIR methodology without the explicit computation of the spectral transformation $\Phi$ may exist. Indeed, the KSIR methodology leverages this in such a way that allows the SIR framework to be applied rather directly. The remainder of this discussion then is to make clear how the computation of the effective directions $\mathbf{a}_k^T \Phi(\mathbf{x})$ can be made while circumventing an explicit computation of the mapping $\Phi$.

To begin, the strategy mimics classical SIR by seeking the dominant directions of the associated covariance matrix. To this end, we let

$$\Sigma_{\mathbf{zz}} = \frac{1}{n} \sum_{i=1}^{n} \Phi\left(\mathbf{x}_i\right) \Phi\left(\mathbf{x}_i\right)^T$$

be the sample covariance matrix of $\mathbf{z} := \Phi(\mathbf{x})$, assuming for sake of convenience that the spectral function $\Phi$ has mean zero. Define the following

$$p_h = \frac{\sum_{i=1}^{n} \delta_h(y_i)}{n} = \frac{n_h}{n}, \qquad \delta_h(y_i) = \begin{cases} 1 & y_i \in h\text{th slice}, \\ 0 & \text{otherwise}, \end{cases} \tag{7.14}$$

so $p_h$ is the proportion of all observed $y_i$'s that fall into the $h$th slice. Also, let $\Sigma_{E(\mathbf{z}|\tilde{y})}$ be the sample between-slice covariance matrix given by

$$\Sigma_{E(\mathbf{z}|\tilde{y})} = \sum_{h=1}^{h_{\max}} p_h \bar{\Phi}_h \bar{\Phi}_h^T, \tag{7.15}$$

where

$$\bar{\Phi}_h = \frac{1}{n p_h} \sum_{i=1}^{n} \Phi\left(\mathbf{x}_i\right)$$

denotes the sample mean of the $h$th slice. As before, we compute the eigenvalues $\lambda \geq 0$ and eigenvectors $\mathbf{v} \in \mathcal{H}$ satisfying

$$\Sigma_{E(\mathbf{z}|\tilde{y})} \mathbf{v} = \lambda \Sigma_{\mathbf{zz}} \mathbf{v}; \tag{7.16}$$

compare this with (7.9). By projection with respect to the RKHS inner product, we have

$$\left\langle \Phi\left(\mathbf{x}_i\right), \Sigma_{E(\mathbf{z}|\tilde{y})} \mathbf{v} \right\rangle_{\mathcal{H}} = \lambda \left\langle \Phi\left(\mathbf{x}_i\right), \Sigma_{\mathbf{zz}} \mathbf{v} \right\rangle_{\mathcal{H}}, \qquad i = 1, \ldots, n.$$

We now seek solutions of the form $\mathbf{v} = \sum_{i=1}^{n} \alpha_i \Phi\left(\mathbf{x}_i\right)$ for some $\alpha_1, \ldots, \alpha_n$. To clean up the consequent computation, we define the kernel data by

$$\mathbf{K} := \left\{ K_{ij} = \left\langle \Phi\left(\mathbf{x}_i\right), \Phi\left(\mathbf{x}_j\right) \right\rangle_{\mathcal{H}} \right\}_{n \times n}, \tag{7.17}$$

with the $ij$th element defining the Gram matrix entry $K_{ij}$. Consider the eigenvalue problem

$$\mathbf{E}_H \mathbf{K} \boldsymbol{\alpha} = \lambda \mathbf{K} \boldsymbol{\alpha}, \qquad \mathbf{E}_H = \sum_{h=1}^{H} n_h^{-1} \mathbf{1}_h \mathbf{1}_h^T, \qquad \mathbf{1}_h = \left[\delta_h(y_1) \ldots \delta_h(y_n)\right]^T, \tag{7.18}$$

where $\boldsymbol{\alpha}$ is an $n$-dimensional vector whose $j$th element is the coefficient $\alpha_j$. A straightforward, yet tedious, calculation, shown in the Appendix of [14], proves that (7.18) is equivalent to the problem given by (7.16). The importance of reformulating the eigenvalue problem (7.16) in this way will become evident soon.

Of course, it is best to work with an orthonormal basis. To this end, KSIR assumes that each generalized eigenfunction $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n$ is normalized so that $\left\langle \mathbf{v}^k, \mathbf{v}^k \right\rangle_{\mathcal{H}} = 1$ for all $k = 1, \ldots, n$. Let $\lambda_1 \geq \cdots \geq \lambda_n$ denote the eigenvalues, and $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n$ be the corresponding complete set of eigenvectors. Since each $\mathbf{v}^k$ is of the form $\mathbf{v}^k = \sum_{i=1}^n \alpha_i^k \Phi\left(\boldsymbol{x}_i\right)$, the normalization implies that

$$1 = \sum_{i=1}^n \sum_{j=1}^n \alpha_i^k \alpha_j^k \left\langle \Phi\left(\mathbf{x}_i\right), \Phi\left(\mathbf{x}_j\right) \right\rangle_{\mathcal{H}} = \left\langle \boldsymbol{\alpha}^k, \mathbf{K} \boldsymbol{\alpha}^k \right\rangle_{\mathbb{R}^n} = \lambda_k \left\langle \boldsymbol{\alpha}^k, \boldsymbol{\alpha}^k \right\rangle_{\mathbb{R}^n}, \qquad k = 1, \ldots, n. \tag{7.19}$$

Tying it all together, we observe how the effective directions are computed. Let $\mathbf{x}$ be a test point, with an image $\Phi(\mathbf{x})$ in $\mathcal{H}$ and let $\tilde{\mathbf{v}}^k$ be the vector $\mathbf{v}^k$ with corresponding normalized $\boldsymbol{\alpha}^k$. Projecting $\Phi(\mathbf{x})$ along the vectors $\mathbf{a}^k := \tilde{\mathbf{v}}^k$ for each $k$ gives

$$\left\langle \mathbf{a}^k, \Phi(\mathbf{x}) \right\rangle_{\mathcal{H}} = \sum_{i=1}^n \alpha_i^k \left\langle \Phi\left(\mathbf{x}_i\right), \Phi(\mathbf{x}) \right\rangle_{\mathcal{H}} = \sum_{i=1}^n \alpha_i^k K\left(\mathbf{x}_i, \mathbf{x}\right). \tag{7.20}$$

We now see that (7.18)–(7.20) determine effective directions and do not require knowledge of the spectral mapping $\Phi$ in explicit form. Therefore, computing the effective directions in KSIR has computational complexity comparable to SIR, *mutatis mutandis*. Moreover, the computational methodology is almost identical to classical SIR discussed in the previous subsection.

To be clear, the procedure is as follows:

(1) Solve the eigenvalue problem (7.18) for each $\mathbf{a}^k$.
(2) Normalize the resulting spectral basis via (7.19).
(3) Compute each effective direction $\left\langle \mathbf{a}^k, \Phi(\mathbf{x}) \right\rangle_{\mathcal{H}}$ by computing the $n$ kernel evaluations, multiplications, and additions required by $\sum_{i=1}^n \alpha_i^k K\left(\mathbf{x}_i, \mathbf{x}\right)$.

In practice, the choice of the kernel function remains. [14] makes the claim that choosing an elliptically symmetric kernel is sufficient to meet the linear design condition for KSIR, which is analogous to (7.4) for SIR. Also, it is discussed that Gaussian kernels with parameters chosen *ad hoc*, work well enough in practice. A further discussion in [16] provides opportunities for computational speed up of KSIR via typical compressions (dimensionality reduction) afforded by the singular value decomposition; see for example [8]. That is, it is often not the case that all the eigenvectors $\mathbf{a}^k$ must be known; just the first few dominant modes would suffice with how many to keep dependent on the desired fidelity in the computational problem.

**Testing KSIR:**

A suggested test for multi-response regression problems is made clear in [16]. There, the so-called Friedman system is used to generate synthetic data with 10 response variables and 10 predictor variables. The predictor variables $x_1, \ldots, x_{10}$ are iid uniformly over

$[0, 1]$, and the 10 response variables are generated via the Friedman system:

$$\begin{aligned} y_1 &= 10\sin(\pi x_1 x_2) + 5(x_3 - 0.5)^2 + x_4 + x_5 + \epsilon, \\ y_2 &= 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon, \\ y_3 &= 5y_1 + \epsilon \\ y_4 &= y_1 + y_2 + \epsilon, \\ y_5, \ldots y_{10} &\sim \mathcal{N}(0, 0.1), \qquad \epsilon \sim \mathcal{N}(0, 1), \end{aligned}$$

where $\mathcal{N}(0, \cdot)$ is a normal distribution with a mean of zero and standard deviation of $\cdot$. In [16], a random sample of size 2000 is generated.

This example is illustrative since, although the responses are in a ten-dimensional space, the effective dimensionality is actually only two. Moreover, it is demonstrated by [16] that KSIR is able to identify the even symmetry in $x_3$ exhibited by the response variables $y_1$ and $y_2$, while SIR does not. Regrettably, the workshop finished before an implementation of KSIR could be undertaken and tested on this example. We leave this for future consideration should one consider a straightforward comparison between SIR and KSIR on identifying the two effective directions in this relatively low- (ten-) dimensional problem.

## 7.3 Reduced-rank regression (RRR)

The summary of the RRR technique found below is based on [7] and largely uses the notation of that reference. We also give the correspondence between those notations and the ones used in the rest of this report. As announced in the preamble to §7, we will point out that RRR, unlike SIR, is not relevant to the problem considered in §2 of this report.

Consider a regression problem

$$\mathbf{Y} = \boldsymbol{\mu} + \mathbf{C}\mathbf{X} + \boldsymbol{\varepsilon}, \tag{7.21}$$

where $\mathbf{Y}, \boldsymbol{\mu}, \boldsymbol{\varepsilon} \in \mathbb{R}^{s\times 1}$, $\mathbf{X} \in \mathbb{R}^{q\times 1}$, $\mathbf{C} \in \mathbb{R}^{s\times q}$, with $\boldsymbol{\mu}$ being the mean of $\mathbf{Y}$ over realizations and $\boldsymbol{\varepsilon}$ is the regression error. The goal is: given $\mathbf{X}$ and $\mathbf{Y}$, find $\mathbf{C}$ and $\boldsymbol{\mu}$ that minimize $\boldsymbol{\varepsilon}$.

To compare (7.21) with the problem considered in §2 of this report, we will write the transpose of the former equation and omit the $\boldsymbol{\varepsilon}$-term for simplicity:

$$\mathbf{Y}^T = \boldsymbol{\mu}^T + \mathbf{X}^T\mathbf{C}^T. \tag{7.21$^T$}$$

This equation corresponds to a single row (*i.e.*, one realization, $N_\mathrm{r} = 1$, of the random vector $\boldsymbol{\xi}$) of (2.7). In other words, $\mathbf{Y}^T$ corresponds to (one row of) $U$, $\mathbf{X}^T$ corresponds to $\boldsymbol{\xi}^T$, and $\mathbf{C}^T$ corresponds to one row of $A$ in (2.7). Thus, $s = N_\mathrm{w}$ and $q = N_\xi$.

If, instead of considering a single realization of $\mathbf{X}$ one considers $N_\mathrm{r} > N_\mathrm{w}$ random realizations (so that now $\mathbf{X}^T \in \mathbb{R}^{N_\mathrm{r}\times N_\xi}$ and $\mathbf{Y}^T \in \mathbb{R}^{N_\mathrm{r}\times N_\mathrm{w}}$), the problem of finding $\mathbf{C}$ reduces to the standard least-squares problem and has the solution:

$$\mathbf{C}^T = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{Y}^T, \tag{7.22}$$

where we have omitted $\boldsymbol{\mu}$ for simplicity. Generically (and for $N_\mathrm{r} > N_\mathrm{w}$), matrices $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}\mathbf{Y}^T$ have full rank, and then so does $\mathbf{C}$. The solution (7.22) will then yield the matrix $A$ in (2.7). We will now explain what new perspective the approach presented in [7][1] brings to the solution of (7.21). We will also explain that *this approach is* **not** *relevant to the problem of finding multiple effective directions* considered in §7.1.

Reference [7] discusses how one can find a matrix $\mathbf{C}$ of rank $t$ that is *lower* than $s = N_{\rm w}$:

$$\operatorname{rank}(\mathbf{C}) = t < s = N_{\rm w}. \tag{7.23}$$

(Note that for the setup of §2, one always has $N_{\rm w} < N_\xi$, so that the rank of $\mathbf{C}$ cannot exceed $N_{\rm w}$.)

To provide an interpretation for the significance of such a lower-rank $\mathbf{C}$, consider an example where in (7.21), $s = N_{\rm w} = 10$, $q = N_\xi$ is arbitrary, and $N_{\rm r} = 1$. Suppose that $\operatorname{rank}(\mathbf{C}) = 6$; then 4 of the rows of $\mathbf{C}$ are linearly dependent on the other 6 rows. As one illustrating situation, suppose that each of the rows 7–10 depends on all of rows 1–6. Then entries 7–10 of each vector (column of) $\mathbf{Y}$ are dependent (a.k.a. "are fully explained by") entries 1–6. This says that the pressures at wells 1–6 fully predict the pressures at wells 7–10. In other words, wells 1–6 form a cluster whose measurements fully predict the results of measurements elsewhere. As a second illustrating situation, suppose that row 7 depends on rows 1–2 and rows 8–10 depend on rows 3–6. Then the pressure measurements in the cluster formed by wells 1–2 predict the measurement in well 7, and the measurements in the cluster formed by wells 3–6 predict the measurements in wells 8–10.

As one can see, the above "clustering" of wells is unrelated to the problem of finding clusters (combinations $\eta$ in (2.6)) of entries of $\boldsymbol{\xi}$ that would provide an approximate description of the pressure $u_j$ at a given well $j$. However it may be similar to the geographic clustering discussed in §4.2.

Nonetheless, for completeness, we will present the algorithm of finding $\mathbf{C}$, assuming that its rank $t < s$ has been previously estimated. Estimation of $t$ appears to be discussed in [4], where, for some reason, the roles of $s$ and $t$ are reversed compared to that used here (and in [7]). This estimation appears to be based on the singular value decomposition of the matrix on the right-hand side of (7.22).

**RRR Algorithm**

0a. Note that one can write $\mathbf{C} = \mathbf{AB}$, where $\mathbf{A}$ and $\mathbf{B}$ are both rank $t$. Matrices $\mathbf{A}$ and $\mathbf{B}$ are not unique, but once one chooses them by the step shown below, one then knows $\mathbf{C}$.

0b. Assume that the mean vectors $\boldsymbol{\mu}_X$ and $\boldsymbol{\mu}_Y$, as well as the covariance matrices $\boldsymbol{\Sigma}_{XX}$, $\boldsymbol{\Sigma}_{XY} \in \mathbb{R}^{q\times s}$, $\boldsymbol{\Sigma}_{YX} \in \mathbb{R}^{s\times q}$, $\boldsymbol{\Sigma}_{YY}$ are known. Note that

$$\boldsymbol{\Sigma}_{YX} = \boldsymbol{\Sigma}_{XY}^T. \tag{7.24}$$

1. The choices

$$\mathbf{A} = \boldsymbol{\Gamma}^{-1/2}\,[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t], \tag{7.25}$$

[1] The RRR theory presented in [7] had been developed in earlier papers; see the second paragraph on p. 249 there for a brief review.

$$\mathbf{B} = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_t^T \end{bmatrix} \boldsymbol{\Gamma}^{1/2}\, \boldsymbol{\Sigma}_{YX}\, \boldsymbol{\Sigma}_{XX}^{-1}, \tag{7.26}$$

$$\boldsymbol{\mu} = \boldsymbol{\mu}_Y - \mathbf{AB}\, \boldsymbol{\mu}_X \tag{7.27}$$

minimize the covariance matrix

$$\mathbb{E}\left(\boldsymbol{\Gamma}^{1/2}\left(\mathbf{Y} - \boldsymbol{\mu} - \mathbf{ABX}\right)\left(\mathbf{Y} - \boldsymbol{\mu} - \mathbf{ABX}\right)^T \boldsymbol{\Gamma}^{1/2}\right) \tag{7.28}$$

(*i.e.*, minimizes all of its eigenvalues simultaneously). Here $\boldsymbol{\Gamma}$ is a symmetric positive definite weight matrix: by choosing it in specific ways, one can make the problem at hand coincide with the principal component or canonical correlates analyses. One can also choose it differently if needed. The vectors $\mathbf{v}_j$ are eigenvectors of the matrix $\boldsymbol{\Gamma}^{1/2}\, \boldsymbol{\Sigma}_{YX}\, \boldsymbol{\Sigma}_{XX}^{-1}\, \boldsymbol{\Sigma}_{XY}\, \boldsymbol{\Gamma}^{1/2}$, which is symmetric due to (7.24) and positive definite. Hence $\mathbf{v}_j$ (can be chosen to) form an orthonormal set and, in particular,

$$\sum_{j=1}^{s} \mathbf{v}_j \mathbf{v}_j^T = I, \tag{7.29}$$

where $I \in \mathbb{R}^{s\times s}$ is the identity matrix. Finally, the matrix $\mathbf{C} = \mathbf{AB}$ is given by:

$$\mathbf{C} = \boldsymbol{\Gamma}^{-1/2} \sum_{j=1}^{t} \mathbf{v}_j \mathbf{v}_j^T \boldsymbol{\Gamma}^{1/2} \boldsymbol{\Sigma}_{YX}\, \boldsymbol{\Sigma}_{XX}^{-1}\, . \tag{7.30}$$

Note that when $t = s$, (7.30) reduces to the usual full-rank least-squares formula (7.22) *via* the use of (7.29).

## 8 Conclusions and Further Research

In this one-week endeavor, we investigated the inverse problem of identifying the transmission field given the sparse observation of the hydraulic head field. A three-step algorithm was developed to represent the problem in a reduced dimension to diminish the computational cost. First, a simplified weight vector $\boldsymbol{\eta}$ was constructed to reduce the number of input parameters to the model from $N_\xi = 10^3$ down to just one. Next we replaced the map from $\eta$ to $u$ with a linear function to make the computation much faster. Last, we implemented standard optimization procedures to identify the optimal $\boldsymbol{\xi}$. Results from this most simple of models compared favorably with the test data set we were given.

Once this proof of concept had been established, we implemented more complicated models. The linear map was replaced by slightly more complicated ones (quadratic, cubic, or sigmoid). Regularization of several different types were introduced and tested. The algorithm was also implemented using a neural network. Again, our results compared favorably with the test data.

Lastly, we provided detailed summaries of more complicated dimension reduction techniques found in the literature, such as SIR, KSIR, and RRR. We also provided descriptions of how these techniques could be applied to the transmission field problem. Another possible direction to pursue is to adopt the physics-informed deep neural network (DNN), in which the transmission field can be captured by a Gaussian random field. The coefficient for this representation can be determined via the DNN, constrained by the physical law or the PDE.

Modeling contaminant transport at the Hanford waste site is a crucial problem due to the high toxicity of the material stored there. This work can provide researchers with a simpler, faster way to characterize the underlying transmissivity field while still maintaining accuracy.

## Nomenclature

If a symbol appears in bold face, it is a vector whose components (typically at wells) are in regular type. Equation numbers where a variable is first defined is listed, if appropriate.

$\mathbf{A}$: matrix in RRR algorithm (7.25).
$A$: matrix whose columns are $\mathbf{a}_j$ (2.7).
$\mathbf{a}$: near-gradient vector (2.2).
$\mathbf{B}$: matrix in RRR algorithm (7.26).
$B$: encoding/decoding matrix (6.3).
$b$: constant in linear expression of $u$ (2.3).
$\mathbf{C}$: matrix in RRR regression problem (7.21).
$C$: matrix whose columns are $\mathbf{c}_j$ (2.12).
$\mathbf{c}$: vector of parameters characterizing $f$ (2.1) or vector in linearity condition (7.4).
$\mathcal{D}$: set of inputs and outputs to $\mathcal{F}$.
$d$: dimension, variously defined.
$\mathbf{E}_H$: weighting matrix in SIR eigenvalue problem (7.18).
$\mathcal{F}$: observation function (5.1).
$\mathbf{f}$: simplified approximation function (2.1).
$\mathcal{H}$: feature space in KSIR problem.
$H$: reproducing kernel Hilbert space (7.11).
$h$: index of slices in SIR algorithm (7.3).
$i$: index variable for realizations (2.4).
$j$: index variable for cells (2.5).
$\mathbf{K}$: Gram matrix (7.17).
$K$: reproducing kernel in KSIR problem (7.11).
$k$: index variable for weights (1.1) or directions.
$\mathcal{L}$: loss function (6.1).
$L$: matrix of differences (§5.2) or evaluation functional in KSIR problem (7.11).
$M$: matrix used in polynomial fitting (2.10).
$\mathbf{m}(u)$: discretized mean curve in SIR algorithm (7.6)
$m(\mathbf{x})$: centering function of $\log T$ (1.1).
$(m_{c,\cdot})_h$: component of $(\mathbf{m}_c)_h$ (7.8).
$N$: maximal index (1.1).
$n$: number of observations in KSIR problem (7.14).
$p_h$: proportion of observations in slice (7.14).
$\mathbf{q}(\mathbf{x})$: groundwater velocity at position $\mathbf{x}$.
$q$: dimension of matrix in RRR regression problem.
$\mathcal{R}(\boldsymbol{\xi})$: regularization function (1.4).
$R$: regularization matrix.
$s$: dimension of matrix in RRR regression problem.
$T(\mathbf{x})$: transmissivity field at position $\mathbf{x}$ (1.1).
$t$: rank($\mathbf{C}$) in RRR regression problem (7.23).
$U$: matrix whose columns are $\mathbf{a}_j$ (2.7).
$\mathbf{u}$: pressure field measurements (1.2).
$\mathbf{v}$: one of an orthonormal set of eigenvectors in the RRR algorithm (7.16).
$W$: latent space.

$\mathbf{X}$: vector of independent variables in RRR regression problem (7.21).
$\mathcal{X}$: input space in KSIR problem.
$X$: arbitrary set.
$\mathbf{x}$: position in flow field (1.1).
$\mathbf{Y}$: vector of measurements in RRR regression problem (7.21).
$\mathbf{y}$: $\log T$ (1.1).
$\mathbf{z}$: arbitrary vector, variously defined.
$\boldsymbol{\alpha}$: eigenvector in SIR eigenvalue problem (7.18).
$\alpha$: regularization parameter for fitting problem for $A$ (5.3).
$\boldsymbol{\Gamma}$: positive definite symmetric matrix in RRR algorithm (7.25).
$\gamma$: regularization parameter (1.4).
$\delta_h(\cdot)$: indicator function (7.14).
$\boldsymbol{\varepsilon}$: vector of regression errors in RRR regression problem (7.21).
$\boldsymbol{\eta}$: simplfied weight vector (2.1).
$\Lambda$: diagonal matrix (3.1).
$\lambda$: eigenvalue (7.9).
$\boldsymbol{\mu}$: mean of $\mathbf{Y}$ over realizations in RRR regression problem (7.21).
$\Xi$: matrix whose rows are $\boldsymbol{\xi}^{(i)}$ (2.5).
$\boldsymbol{\xi}$: weight vector (1.1).
$\rho$: nearest-neighbor inter-well distance (4.2).
$\Sigma$: covariance matrix (7.7).
$\boldsymbol{\sigma}$: function in neural network implementation (6.3).
$\Phi$: spectral function in KSIR problem (7.13).
$\Psi$: matrix of basis vectors.
$\psi(\mathbf{x})$: basis vector (1.1).
$\Omega$: computational domain (1.2).

## Other Notation

c: as a subscript on $N$, refers to the number of computational cells; as a subscript on $\mathbf{m}$, refers to the centered mean curve (7.6).
dec: as a subscript, used to represent the decoding phase (6.3).
enc: as a subscript, used to represent the encoding phase (6.3).
$h$: as a subscript, refers to a slice (7.3).
$(k)$: as a superscript on $\mathbf{a}$, used to refer to effective direction (7.2).
m: as a subscript on $\Sigma$, refers to the mean curve covariance matrix (7.7).
max: as a subscript, used to represent a maximum value.
obs: as a subscript, used to refer to observations.
ref: as a subscript, used to refer to a reference value.
r: as a subscript, refers to the number of random realizations (2.4).
tm: as a subscript on $y$, refers to the total mean.
w: as a subscript, refers to wells (where observations are taken).
$\eta$: as a subscript on $N$, indicates the number of weights in vector $\boldsymbol{\eta}$.
$\xi$: as a subscript on $N$, indicates the number of weights in vector $\boldsymbol{\xi}$ (1.1); as a subscript on $\Sigma$, indicates the simple covariance matrix (7.9).
$*$: as a subscript, used to refer to an optimizer (1.3).
$\bar{}$: used to refer to an average (7.3).
$\dagger$ : used to refer to an estimate from a single set of data.
$\hat{}$: used to refer to a unit vector (2.2) or estimate.
$\tilde{}$: used to refer to a matrix that has been expanded (2.8), a matrix or vector that has been modified (6.5), or to arguments of a between-slice covariance matrix (7.15).
$||\cdot||_F$: Frobenius norm (5.3).

## References

[1] Chen, Chun-Houh, & Li, Ker-Chau. (1998). Can SIR be as popular as multiple linear regression? *Statistica sinica*, **8**, 289–316.

[2] Cook, Dennis, & Weisberg, Sanford. (1991). Sliced inverse regression for dimension reduction: Comment. *Journal of the american statistical association*, **86**, 328–332.

[3] Cuomo, Salvatore, Di Cola, Vincenzo Schiano, Giampaolo, Fabio, Rozza, Gianluigi, Raissi, Maziar, & Piccialli, Francesco. (2022). Scientific machine learning through physics–informed neural networks: where we are and what's next. *Journal of scientific computing*, **92**(3), 88.

[4] Davies, P.T., & Tso, M.K.-S. (1982). Procedures for reduced-rank regression. *Applied statistics*, **31**, 244–255.

[5] Duan, Naihua, & Li, Ker-Chau. (1991). Slicing regression: A link-free regression method. *The annals of statistics*, **19**, 505–530.

[6] Hristopulos, D.T. (2020). *Random fields for spatial data modeling: A primer for scientists and engineers*. Advances in Geographic Information Science. Springer Netherlands.

[7] Izenman, Alan Julian. (1975). Reduced-rank regression for the multivariate linear model. *Journal of multivariate analysis*, **5**, 248–264.

[8] Kutz, J. Nathan, & Brunton, Steven L. (2019). *Data-driven science and engineering.* Control-Cambridge University Press.

[9] Li, Ker-Chau. (1991). Sliced inverse regression for dimension reduction. *Journal of the american statistical association*, **86**, 316–327.

[10] Lieb, E., & Loss, M. (2010). *Analysis.* Second edn. American Mathematical Society.

[11] O'Shea, Keiron, & Nash, Ryan. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

[12] Wang, Hansheng, & Xia, Yingcun. (2008). Sliced regression for dimension reduction. *Journal of the american statistical association*, **103**, 811–821.

[13] Wikipedia contributors. (2023). *Sliced inverse regression — Wikipedia, the free encyclopedia.*

[14] Wu, Han-Ming. (2008). Kernel sliced inverse regression with applications to classification. *Journal of computational and graphical statistics*, **17**(3), 590–610.

[15] Xia, Yingcun, Tong, Howell, Li, W.K., & Zhu, Li-Xing. (2002). An adaptive estimation of dimension reduction space. *Journal of royal statistical society, series B*, **64**, 363–410.

[16] Yeh, Yi-Ren, Huang, Su-Yun, & Lee, Yuh-Jye. (2008). Nonlinear dimension reduction with kernel sliced inverse regression. *IEEE transactions on knowledge and data engineering*, **21**(11), 1590–1603.

## Appendix A Parameter Values

We were given values of

$$N_\xi = 10^3, \quad N_\mathrm{r} = 2\times 10^5, \quad N_\mathrm{w} = 323, \quad N_\mathrm{c} = 1475. \tag{A 1}$$