On Obtaining Sparse Semantic Solutions for Inverse Problems, Control, and Neural Network Training

David A. B. Hyde[a,*], Michael Bao[b,**], Ronald Fedkiw[b,c,**]

[a] *UCLA Mathematics Department, Box 951555, Los Angeles, CA 90095-1555, United States*
[b] *Stanford University, 353 Jane Stanford Way, Gates Computer Science Room 207, Stanford, CA 94305, United States*
[c] *Epic Games, Inc., 620 Crossroads Boulevard, Cary, NC 27518, United States*

Abstract

Modern-day techniques for designing neural network architectures are highly reliant on trial and error, heuristics, and so-called best practices, without much rigorous justification. After choosing a network architecture, an energy function (or loss) is minimized, choosing from a wide variety of optimization and regularization methods. Given the ad-hoc nature of network architecture design, it would be useful if the optimization led to a sparse solution so that one could ascertain the importance or unimportance of various parts of the network architecture. Of course, historically, sparsity has always been a useful notion for inverse problems where researchers often prefer the L_1 norm over L_2. Similarly for control, one often includes the control variables in the objective function in order to minimize their efforts. Motivated by the design and training of neural networks, we propose a novel column space search approach that emphasizes the data over the model, as well as a novel iterative Levenberg-Marquardt algorithm that smoothly converges to a regularized SVD as opposed to the abrupt truncation inherent to PCA. In the case of our iterative Levenberg-Marquardt algorithm, it suffices to consider only the linearized subproblem in order to verify our claims. However, the claims we make about our novel column space search approach require examining the impact of the solution method for the linearized subproblem on the fully nonlinear original problem; thus, we consider a complex real-world inverse problem (determining facial expressions from RGB images).

Keywords: Machine learning, Levenberg-Marquardt, principal component analysis, column space search, coordinate descent

1. Introduction

The current age of deep learning began (at least according to the Turing Award committee[1]) with works addressing problems such as object classification [73, 82], reading handwritten digits and documents [80, 81, 79], and speech recognition and natural language tasks [11, 100]. Although models based on traditional scientific first principles do not exist for these sorts of problems, the underlying machine learning methods have been permeating into various scientific communities, including computational physics [69, 51, 49, 123, 48, 89, 113, 115, 132]. Perhaps the main difference between the use of machine learning for customizing advertisements [19, 55], online dating [36, 99], or self-driving cars [16, 67] and its use in computational physics is that our community has developed a fairly reasonable scientific and mathematical understanding of many of the problems of interest via a combination of theoretical, experimental, and computational approaches, especially as opposed to the ad-hoc data-driven nature of popular machine learning application areas. Unfortunately, ad-hoc approaches leave neural networks wide open to adversarial attacks [65, 3, 125], which does not bode well for predictive numerical capabilities. Therefore, one goal of our community (and perhaps contribution) would be to better understand neural network architectures in order to provide a more thorough and rigorous approach to designing them, similar to the contributions that the applied mathematics

*dabh@math.ucla.edu, UCLA
**mikebao@stanford.edu, fedkiw@cs.stanford.edu, Stanford University
[1] https://awards.acm.org/about/2018-turing

community made to finite element simulation, e.g. reformulating spring and beam elements as basis functions [141, 142].

Techniques used in modeling and training neural networks are highly related to well-studied approaches for inverse problems and control. To understand some of the differences between inverse problems, control, and training neural networks, consider $Y = f(X; C)$, with input X, output Y, and function parameters C. In a typical inverse problem, one is given Y and aims to find an X that produces Y. Poor conditioning of the function f or noise in the given/desired output Y can lead to spurious information contained in X. Thus, various regularization approaches may be used to ascertain an X with a high signal-to-noise ratio, see for example [26, 92, 131, 140] and the more general references [42, 39, 12]. In the control problem, X and Y are both given, and the goal is to ascertain some subset of the function parameters C that allows one to coerce X toward Y. Typically, most of f is a well-known function, such as the Navier-Stokes equations, and thus the added controls should have a light/minimal touch; therefore, they are often included in the objective function so that their magnitude/effort is minimized. This too is regularization, and needs to be done wisely so that minimizing controls does not prevent one from hitting the target (while still considering signal-to-noise ratio, etc.), see e.g. [70, 2, 118]. When considering neural networks, the function f is almost entirely ad-hoc, and one does not know which parameters might have physicality and which are more arbitrary. Thus, it becomes even more important to consider careful regularization with the hope that some of the coefficients will dominate others, providing some insight into which portions of the network architecture may have some basis in first principles as opposed to which may be considered for removal/replacement, see e.g. [137, 95, 138, 133, 54, 90, 60, 93, 102, 134, 56, 109, 85, 4, 57, 58]. Because so little is known about f, neural networks cannot proceed with one input X and one output Y as can a control problem. Instead, one requires a family of given $(X; Y)$ pairs called training data, before an attempt to identify the function coefficients C can be made. Methods for formulating and optimizing neural networks are typically significantly more rudimentary and ad-hoc than those designed for inverse and control problems, relying on simple methods such as gradient descent and stochastic gradient descent (SGD) or ordinary differential equation discretizations of gradient flow, such as Adam [72], AdaGrad [37], Nesterov [103], momentum methods [114, 126], etc. [117, 17, 53].

The process of network architecture design is often motivated by heuristics that hinder the ability to subsequently train the network and find suitable coefficients. For example, the "all or none" property of biological neurons leads to discontinuous functions with identically zero derivatives almost everywhere, which is disastrous for optimization/training [98]. The idea that biological neurons fire with increased frequency for stronger signals leads to piecewise linear functions with discontinuous derivatives, also problematic for optimization. These Heaviside and rectifier/ReLU [53] models require smoothing before they can subsequently be used with numerical optimization. It seems quite dubious to design and analyze non-smooth network architectures that are later smoothed in the first significant digit when deployed in practice, especially given the nuances exposed by the numerical analysis community regarding the differences between continuous and discrete formulations (e.g. [61]) even when such occurs only at the level of machine precision (the 7^{th} or 15^{th} decimal place). This motivates our aim to better utilize various approaches to regularization and sparsity to ascertain the importance of various components of the network architecture.

In Section 2, we introduce a suitably complex model problem for demonstrating the numerical methods presented in the paper: determining facial expression from RGB images. This problem is both algorithmically challenging and grounded in physics, meaning that we can attempt to develop algorithms which find semantically meaningful solutions in terms of known physical and anatomical properties. In Section 3, we outline a general framework for optimization, showing how neural network training is recast as a nonlinear optimization problem. We highlight various approximations made in practice during this process, such as (sometimes drastic) approximations to the Hessian and Jacobian. As is typical, rank-one updates are discussed, which motivates the singular value decomposition (SVD) and principal component analysis (PCA), both of which are used in subsequent sections. Section 4 presents a novel iterative Levenberg-Marquardt [83, 97] scheme that is shown by proof and experiment to converge *smoothly* (and monotonically) to a regularized SVD, unlike the truncation typical of a PCA approach. Section 5 presents a novel column space search technique that focuses more on the data term than the model, again an improvement over PCA. Moreover, we explain how column space search enables the discovery of sparse and semantically meaningful solutions to fully nonlinear optimization problems. To demonstrate this experimentally, we compare to alternative strategies such as Dogleg [112, 94] and BFGS with L_2 or soft L_1 regularizers.

2. Tackling Complex Real-World Inverse Problems for Faces

We chose a fairly complex model problem which is still cutting-edge in order to illustrate the need for robust and efficient approaches. Specifically, we consider an inverse problem where a two-dimensional RGB image of a human face is processed to determine facial expressions in terms of a three-dimensional parameterized model with a semantic, anatomical basis. This inverse problem is useful throughout industries such as medicine, surveillance, intelligence gathering, entertainment, etc. Similar to many other complex processes, one can understand the problem via a pipeline with various function layers, see Figure 1.

$$X_1 \xrightarrow{\text{input}} f_1\left(X_1, C_1\right) \xrightarrow{\text{output}} X_2 \xrightarrow{\text{input}} f_2\left(X_2, C_2\right) \xrightarrow{\text{output}} X_3 \xrightarrow{\text{input}} f_3\left(X_3, C_3\right) \xrightarrow{\text{output}} X_4$$

$$\hat{f}\left(X_4\right) = ||X_4 - X_{\text{target}}||$$

Figure 1: Multiple layers of functions f_i map an initial vector of inputs X_1 to a final output X_4, which is evaluated with an objective function $\hat{f}$. Vectors of parameters C_i may either be prescribed or be determined via experimentation or neural network training.

The inverse problem seeks to find an X_1 that outputs an X_4 as close to X_{target} as possible, i.e. minimizing $\hat{f}\left(X_4\right)$, using regularization to combat noise and overfitting when necessary. Using classical optimization, this requires differentiation that can be expressed as

$$\frac{\partial \hat{f}}{\partial X_1} = \frac{\partial \hat{f}}{\partial X_4} \frac{\partial f_3\left(X_3, C_3\right)}{\partial X_3} \frac{\partial f_2\left(X_2, C_2\right)}{\partial X_2} \frac{\partial f_1\left(X_1, C_1\right)}{\partial X_1}, \tag{1}$$

implying that every function layer requires differentiability with respect to its inputs. Now suppose that $f_2\left(X_2; C_2\right)$ represented a neural network layer that needs to be trained in order to ascertain reasonable parameters C_2. In order to do this, one would consider a large number K of known training pairs of the form $\left(X_1^k; X_{\text{target}}^k\right)$; however, notationally, one may stack all the training pairs into a single X_1 and X_{target}, at least conceptually (for the sake of exposition). Then the required differentiation is

$$\frac{\partial \hat{f}\left(X_4\right)}{\partial C_2} = \frac{\partial \hat{f}}{\partial X_4} \frac{\partial f_3\left(X_3, C_3\right)}{\partial X_3} \frac{\partial f_2\left(X_2, C_2\right)}{\partial C_2}, \tag{2}$$

highlighting the notable differences as compared to an inverse problem. Firstly, any pre-process, such as f_1 here, does not require differentiability and can utilize any known procedural methods. In fact, one might use an f_1 based on first principles aiming to solve the problem outright, and then supplement the results with the composition of f_2 and f_3 in order to better match real-world data. This means that data-driven neural network approaches may be added on top of any existing codebase, whether it is differentiable or not. Secondly, any post-process for the neural network, such as f_3, only requires as much differentiability as would be required for f_3 if it were included in a typical inverse problem. Thirdly, the neural network itself, f_2, does not require the usual differentiability inherent to inverse problems, but only requires differentiability with respect to its parameters C_2.

Most facial pipelines take as input a set of parameters that govern the shape/geometry of a three-dimensional face, as given by triangle vertex positions. For example, a blendshape facial rig (see e.g. [84]) describes how a face is deformed from a neutral rest state n in terms of a linear combination of basis facial shapes, e.g. semantic basis vectors which represent particular expressions such as "smile" or "yawn." The basis facial shapes are often acquired using dense performance capture (see e.g. [9, 10, 20, 50]) or via sculpting by an artist/modeler [30, 75]. A typical high-quality blendshape rig contains hundreds of basis shapes corresponding to different expressions between which one can interpolate (see e.g. [29]). Once a blendshape model is obtained, stacking each blendshape into a column of a matrix B allows one to define the facial geometry as $n + Bb(w)$, where the vector b contains a degree of freedom for each blendshape and w represents a set of meaningful controls ($b(w)$ may be nonlinear, but should be smooth). In order to

avoid linearized rotation artifacts due to rotational jaw motion [32, 121, 143], one typically hybridizes the linear blendshape system with skinning/enveloping (see e.g. [96, 76]), which blends together the nonlinear six-degree-of-freedom rigid body transformation from the skull and jaw. Each triangle vertex is assigned weights that dictate the relative influence of the skull and jaw such that vertices far from the jaw move with the skull, vertices far from the skull move with the jaw, and vertices in between move in a blended fashion. This can be written compactly as a matrix $T(j(w))$, where the controls w drive the six-degree-of-freedom rigid body offset j of the jaw from the skull and T assembles all the transformations and weights so that one may write

$$x(w) = T(j(w))(n + Bb(w)), \tag{3}$$

where $x(w)$ are the triangle vertex positions of the face surface. Importantly, as long as the dependencies in Equation 3 are chosen carefully (in a smooth enough manner), then x is differentiable with respect to w.

As an alternative to blendshape approaches, one can construct an anatomically motivated finite element facial model based on soft tissue, musculature, and underlying skeletal structures (see e.g. [121, 122]). In [121], the vertex positions are differentiable as a function of the muscle activations and jaw parameters, and the authors used this differentiability to solve inverse problems. However, since anatomical facial models rely on MRI, CT scans, etc., it is difficult to make an accurate model; therefore, [121, 122] struggled to express the wide variety of shapes possible with a blendshape system. Thus, [31] augmented the results of [121] using a three-dimensional morphing process in order to derive target locations for muscles. Although the method proposed in [31] regains the expressivity of a blendshape system, their morphing process lacked differentiability. Later, [7] noted that the morphing process could be made differentiable, but that this would require a mapping from each surface vertex to all other affected vertices in the simulation mesh, which is quadratic complexity and thus impractical. Instead, [7] parameterized the morph with a standard blendshape system, so that the parameters b drive the morph, resulting in linear complexity. This was implemented (in [7]) by simulating the anatomical mesh for each blendshape (using the morphing from [31]) in order to create the muscle shapes needed in order to define a blendshape system for the muscles themselves; then, the three-dimensional target shape of each muscle can be specified by the parameters w, with each muscle tetrahedron vertex $x(w)$ defined along the lines of Equation 3. Manipulating w determines a blendshape for each muscle, which is then targeted with the anatomical finite element simulation from [31] and [121], see Figure 2. Notably, the resulting scheme is fully differentiable and hence can be used to solve inverse problems.

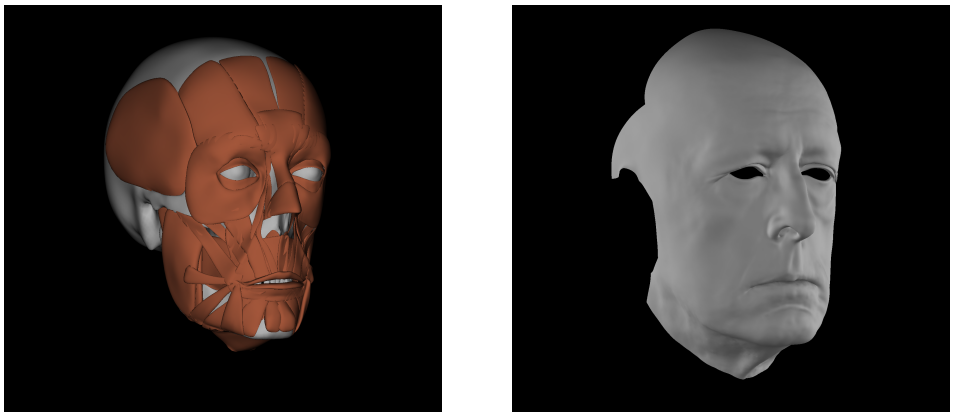


Figure 2: *(Left)* Skull and jaw (gray) with anatomical muscle shapes (red). *(Right)* Corresponding surface of the tetrahedral finite element mesh simulated from the targeted muscle shapes.

Given target geometry for the three-dimensional face surface, one can specify an energy that minimizes the distance between the target and the parameterized model, and then solve an inverse problem for the parameters w that drive the b and j for the muscle blendshape system, which in turn drives the quasistatic finite element simulation of [121] augmented by [31] in order to match the target (see [7] for details). In order to match a two-dimensional RGB image, one needs to render the resulting geometry with a differentiable renderer along the lines of [88, 91] and utilize an energy that considers the difference in pixel colors. Then, one can solve an inverse problem for the controls w that drive the blendshape muscles which in turn drive the

finite element simulation which results in the surface mesh that is rendered into pixel colors that minimize the energy. Unfortunately, as shown in Figure 3 Left, differentiable renderers don't typically have the same quality as a photorealistic renderer or photograph, so aiming to match pixel colors is overly optimistic. To overcome this limitation, [6] proposed processing both the image and the differentiable render with a pre-existing/widespread face landmark detector neural network, such as 2D/3D-FAN [24] (see Figure 3). These networks were trained with vast amounts of hand-labeled data so that they could find keypoint/landmark positions from images regardless of texture, geometry, shading, lighting, shadows, etc. As such, the poor rendering quality of a differentiable renderer is also serendipitously ignored by these neural networks. In summary, [6] utilizes an energy that computes the difference between keypoints/landmarks, and solving the inverse problem requires differentiating through the keypoint detector neural network (2D/3D-FAN), the differentiable renderer, and the quasistatic finite element muscle simulation.

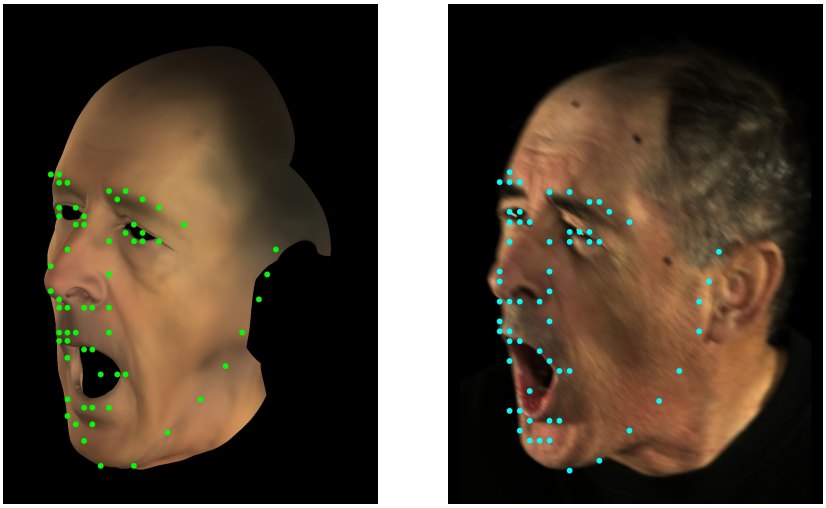


Figure 3: Results of a machine-learning based facial keypoint detector such as 2D/3D-FAN [24] on a synthetic render (left) as well as the corresponding photograph (right).

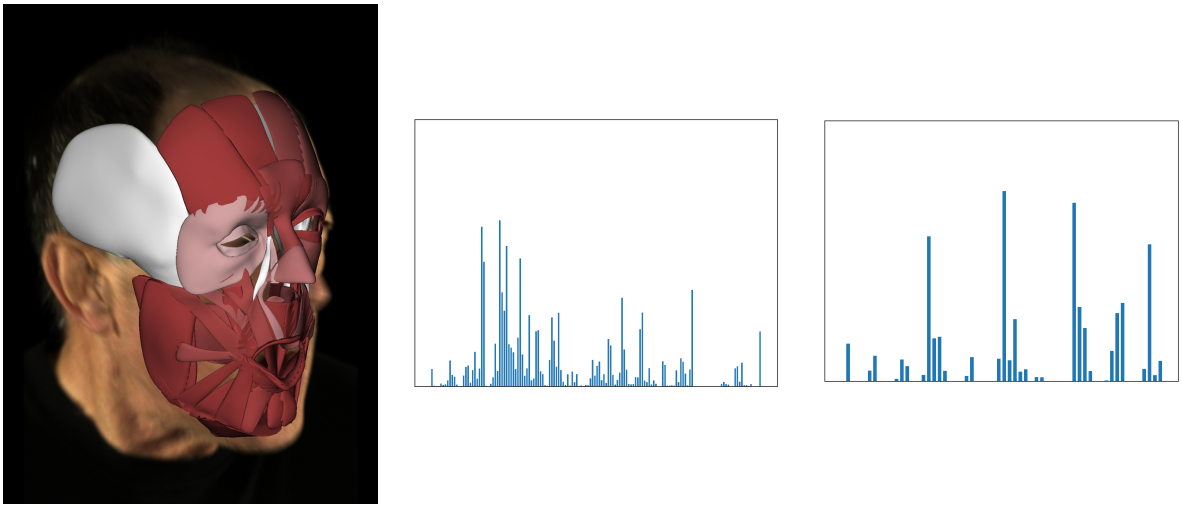


Figure 4: *(Left)* A pose of the hybridized muscle system of [7] depicted on top of the corresponding target RGB photograph. Whiter muscle shapes correspond to more activated muscles. *(Middle)* The corresponding blendshape weights. *(Right)* The corresponding muscle activations.

Figure 4 illustrates results typical of this process. Figure 4 (Middle) shows the value of b along the vertical axis for each blendshape on the horizontal axis. In spite of the expression in Figure 4 (Left) being not that

complex, many blendshapes have non-zero values; worse yet, successive frames in a video produce noisy uncorrelated blendshape values that are hard to interpret as meaningful semantic information. In contrast, Figure 4 (Right) illustrates that the muscle activations are sparser and more indicative of the image; in fact, successive frames tend to be highly correlated, allowing one to separate semantic information from noise. Generally speaking, sparse semantic solutions to inverse problems are obviously preferred over dense, noisy, temporally uncorrelated results. This goal of ascertaining sparser semantic information motivates our considerations throughout the rest of the paper.

3. Optimization Framework

Whether it be the search for viable inputs for an inverse problem, minimizing some measure of effort for a control problem, or the determination of network architecture parameters that allow a neural network to well-match training data, these problems all take the form of an optimization minimizing a cost function $\hat{f}(c)$ over parameters c. Importantly, one typically has certain conditions in mind to which c should be subject. For example, one might want c close to a prior/initial guess, one might desire the norm of c to be small, and/or one might want c sparse so that it carries interpretable semantic meaning. In particular, as noted above, it would be useful if neural network training resulted in a sparse c in order to identify unnecessary/unimportant components of the network architecture.

Either in the absence of constraints or with constraints and suitable Lagrange multipliers, the minima of $\hat{f}$ occur at critical points where the (column vector) Jacobian $F(c) = J_{\hat{f}}^T(c) = \nabla \hat{f}(c) = 0$. Since $F(c) = 0$ is (generally) a nonlinear system of equations, one typically linearizes the system by taking the first two terms of the Taylor expansion about a point c^*, $F(c) \approx F(c^*) + F'(c)(c - c^*)$ where $F'(c) = J_F(c) = H_{\hat{f}}^T(c)$ is the transpose of the Hessian of $\hat{f}$. Newton's method uses this relationship to write $F\left(c^{q+1}\right) - F\left(c^q\right) = F'\left(c^q\right)\Delta c^q$, where $\Delta c^q = c^{q+1} - c^q$ and q represents the current iteration. Then, one solves the linear system $F'\left(c^q\right)\Delta c^q = \beta F\left(c^q\right) - F\left(c^q\right)$ to update $c^{q+1} = c^q + \Delta c^q$ where $\beta \in [0, 1)$, and using $\beta \neq 0$ more slowly shrinks $F\left(c^q\right)$ towards 0. Alternatively, one can utilize Δc^q merely as a search direction and subsequently employ a number of one-dimensional approaches, e.g. bisection search, golden section search, etc.

While Newton's method and similar techniques are reasonably well-justified and often converge well in practice, they depend on access to various derivatives of the cost function $\hat{f}(c)$. To compute these derivatives, one may utilize symbolic/analytic differentiation, finite differences, or automatic differentiation (e.g. backpropagation). Automatic differentiation is often preferred in the context of training neural networks both because of its ease of implementation as well as its availability via various software packages (e.g. Tensorflow [1], Caffe [71], PyTorch [110], Theano [127], etc.); however, roundoff and other errors generally accumulate proportional to the size of the network, which can turn out to be numerically catastrophic. Moreover, high dimensionality makes the computation and storage of $H_{\hat{f}}^T$ impractical, and thus practitioners typically resort to quasi-Newton methods that aim to avoid direct consideration of second derivatives.

Broyden's method [21] for solving nonlinear systems, in the context of optimization, first approximates $\left(H_{\hat{f}}^T\right)^0 = I$, and then iteratively uses rank-one updates aiming for successively better estimates. Each iteration, one solves $\left(H_{\hat{f}}^T\right)^q \Delta c^q = -J_{\hat{f}}^T\left(c^q\right)$ to find a search direction Δc^q, and then uses line search to find c^{q+1}; subsequently, Δc^q is updated via $\Delta c^q = c^{q+1} - c^q$. Given $\left(\Delta J_{\hat{f}}^T\right)^q = J_{\hat{f}}^T\left(c^{q+1}\right) - J_{\hat{f}}^T\left(c^q\right)$, the rank-one update is

$$\left(H_{\hat{f}}^T\right)^{q+1} = \left(H_{\hat{f}}^T\right)^q + \frac{1}{\left(\Delta c^q\right)^T \Delta c^q}\left(\left(\Delta J_{\hat{f}}^T\right)^q - \left(H_{\hat{f}}^T\right)^q \Delta c^q\right)\left(\Delta c^q\right)^T \tag{4}$$

so that $\left(H_{\hat{f}}^T\right)^{q+1} \Delta c^q = \left(\Delta J_{\hat{f}}^T\right)^q$. When c is of large dimension, forming and inverting the dense $O\left(n^2\right)$ $H_{\hat{f}}^T$ is undesirable, especially considering that the approximation is built from rank-one updates, and thus a matrix-free approach to the action of $H_{\hat{f}}^{-T}$ on a vector is preferred. That is, $\Delta c^q = -\left(H_{\hat{f}}^{-T}\right)^q J_{\hat{f}}^T\left(c^q\right)$ is used to find the search direction for the line search used to determine c^{q+1}, which is used to update Δc^q and

$\left(\Delta J_{\hat{f}}^T\right)^q$; then, rank-one update for $H_{\hat{f}}^{-T}$ is

$$\left(H_{\hat{f}}^{-T}\right)^{q+1} = \left(H_{\hat{f}}^{-T}\right)^q + \frac{\left(\Delta c^q - \left(H_{\hat{f}}^{-T}\right)^q \left(\Delta J_{\hat{f}}^T\right)^q\right) (\Delta c^q)^T \left(H_{\hat{f}}^{-T}\right)^q}{(\Delta c^q)^T \left(H_{\hat{f}}^{-T}\right)^q \left(\Delta J_{\hat{f}}^T\right)^q}, \tag{5}$$

so that $\left(H_{\hat{f}}^{-T}\right)^{q+1} \left(\Delta J_{\hat{f}}^T\right)^q = \Delta c^q$. Other low-rank update methods such as SR1 [33, 22], DFP [33, 46], and BFGS [23, 44, 52, 119] are similar in spirit. In particular, the limited-memory L-BFGS [106] only stores the past several low-rank updates making it quite efficient, see e.g. [78, 34].

Instead of performing rank-one updates to improve upon $\left(H_{\hat{f}}^T\right)^0 = I$ as in Broyden-style methods, gradient descent methods simply use $H_{\hat{f}}^T = I$ so that the search direction is obtained trivially via $\Delta c^q = -J_{\hat{f}}^T (c^q) = -\nabla \hat{f} (c^q)$. When problems have high dimensionality, practitioners often make further simplifications such as evaluating only a subset of the right-hand side (mini-batch gradient descent) or even just one or a few randomly-selected entries at a time (SGD), see e.g. [117, 18]. One can even ignore the search direction equation entirely by choosing Δc^q to be various basis vectors, i.e. coordinate descent [120]. Furthermore, gradient descent methods can be envisioned as forward Euler approximations of gradient flow, i.e. of $\frac{dc(t)}{dt} = -\nabla \hat{f} (c(t))$, which allows for the wealth of knowledge in designing and solving ordinary differential equations to be utilized. For instance, adaptive time stepping leads to such techniques as AdaGrad [37], which utilizes separate learning rates (time steps) for each parameter, or AdaDelta [139] and RMSprop [129], both of which lessen the effects of history terms in AdaGrad in order to maintain a sufficiently positive learning rate to avoid stalling. Incorporating the effects of prior search directions and state updates can be seen as utilizing momentum, which rewrites gradient flow using Newton's Second Law [114]. The Adam method [72] combines the notion of using a moving average of gradients as in momentum methods with an adaptive learning rate for each parameter. The 52,000+ citations[2] of [72] indicate the success practitioners have enjoyed with Adam, often finding that it converges faster than SGD.

4. Iterative Levenberg-Marquardt

When training a neural network on data (x_i, y_i), one seeks to find the parameters c of a generally vector-valued function $f(x, y, c)$ that minimize error over the training data, i.e. one desires $||f(x_i, y_i, c)||$ to be close to zero for all i. Choosing the L_2 norm leads to minimizing $\hat{f}(c) = \frac{1}{2} \sum_i f(x_i, y_i, c)^T f(x_i, y_i, c) = \frac{1}{2} \tilde{f}^T(c) \tilde{f}(c)$, which is a nonlinear least squares problem [14]. Critical points have $J_{\tilde{f}}^T (c) = J_{\tilde{f}}^T (c)\, \tilde{f}(c) = 0$, which can be rewritten using the Taylor expansion of $\tilde{f}(c)$ about c^q as $J_{\tilde{f}}^T (c) \left(\tilde{f}(c^q) + J_{\tilde{f}}(c^q)\, \Delta c^q + \cdots\right) = 0$, where $\Delta c^q = c - c^q$. Dropping high-order terms and evaluating $J_{\tilde{f}}^T$ at c^q leads to the Gauss-Newton equations $J_{\tilde{f}}^T (c^q) J_{\tilde{f}}(c^q) \Delta c^q \approx -J_{\tilde{f}}^T (c^q) \tilde{f}(c^q)$, which imply an estimate of $H_{\tilde{f}}^T (c^q) \approx J_{\tilde{f}}^T (c^q) J_{\tilde{f}}(c^q)$, see e.g. [107]. Notably, the Gauss-Newton approximation to the Hessian only requires first derivatives. Moreover, since the Gauss-Newton equations are the normal equations for $J_{\tilde{f}}(c^q) \Delta c^q = -\tilde{f}(c^q)$, one can obtain Δc^q via any least squares and minimum norm approach for solving this much better conditioned set of equations.

When $J_{\tilde{f}}$ is poorly-conditioned or rank-deficient, one can regularize the Gauss-Newton equations via $\left(J_{\tilde{f}}^T (c^q) J_{\tilde{f}}(c^q) + \epsilon^2 I\right) \Delta c^q = -J_{\tilde{f}}^T (c^q) \tilde{f}(c^q)$ with $\epsilon > 0$, which is referred to as Levenberg-Marquardt or damped nonlinear least squares, see e.g. [83, 97, 107, 14]. This makes a tradeoff between solvability and accuracy, since the unique and least squares components of the solution will be perturbed away from their correct values. To illuminate this, consider stacking a general linear system $Ac = b$ with the full-rank $\epsilon I c = 0$ to obtain

$$\begin{pmatrix} A \\ \epsilon I \end{pmatrix} c = \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{6}$$

[2] as of September 2020, according to Google Scholar

which has equivalent normal equations of $\left(A^TA + \epsilon^2 I\right) c = A^T b$. Using the SVD, $A = U\Sigma V^T$, this becomes $\left(\Sigma^T\Sigma + \epsilon^2 I\right) \hat{c} = \Sigma^T \hat{b}$ where $\hat{c} = V^T c$ and $\hat{b} = U^T b$. For a general A, Σ has the form

$$\Sigma = \begin{pmatrix} \hat{\Sigma} & 0 \\ 0 & 0 \end{pmatrix}, \tag{7}$$

where $\hat{\Sigma}$ is diagonal and full-rank. This leads to

$$\left(\begin{pmatrix} \hat{\Sigma}^T & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \hat{\Sigma} & 0 \\ 0 & 0 \end{pmatrix} + \epsilon^2 I \right) \begin{pmatrix} \hat{c}_r \\ \hat{c}_z \end{pmatrix} = \begin{pmatrix} \hat{\Sigma}^T & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \hat{b}_r \\ \hat{b}_z \end{pmatrix}, \tag{8}$$

where $\hat{c}$ and $\hat{b}$ have been decomposed to separate out the portions that correspond to identically-zero submatrices of Σ^T. Equation 8 sets $\hat{c}_z$ identically equal to zero as desired (i.e. minimum norm solution), but the entries in $\hat{c}_r$ are determined via

$$\hat{c}_k = \frac{\sigma_k}{\sigma_k^2 + \epsilon^2} \hat{b}_k = \left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2} \right) \frac{\hat{b}_k}{\sigma_k}, \tag{9}$$

perturbing them away from their correct unique or least squares solution $\hat{c}_k = \hat{b}_k / \sigma_k$. This perturbation is negligible for $\sigma_k \gg \epsilon$, but smaller σ_k have their associated $\hat{c}_k$ more significantly incorrectly perturbed toward zero. One typically chooses ϵ so that it does not interfere too much with the larger (more important) singular values, while still being large enough to regularize numerical issues associated with smaller σ_k (as well as identically zero singular values). As a side note for weighted least squares, one adds the full-rank $\epsilon D c = 0$ (with diagonal D) instead of $\epsilon I c = 0$ to obtain a modified version of Equation 6, which after column scaling becomes

$$\begin{pmatrix} AD^{-1} \\ \epsilon I \end{pmatrix} Dc = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{10}$$

Then, a simple renaming of variables results in the original Equation 6, and the above analysis applies without modification.

Motivated by the Broyden-style iterative methods (discussed in the previous section) which began with a simple guess for the Hessian and then corrected it after each iteration, we propose a similar strategy for Levenberg-Marquardt. That is, we start with $\epsilon I c = 0$ but subsequently update the right-hand side as the iteration proceeds, progressively removing the erroneous perturbation of the least squares solution shown in Equation 9. Our approach converges to the exact solution for larger singular values, as for example would also be achieved using PCA; however, unlike the all-or-nothing approach of PCA, our approach *smoothly* tapers between the exact solution for larger σ_k and robust regularization for smaller σ_k.[3]

We start with a guess c^* for c and stack $Ac = b$ with $\epsilon I c = \epsilon c^*$ leading to the normal equations

$$\left(A^TA + \epsilon^2 I\right) c = A^T b + \epsilon^2 c^*. \tag{11}$$

Substituting the SVD of A leads to

$$\left(\Sigma^T\Sigma + \epsilon^2 I\right) \hat{c} = \Sigma^T \hat{b} + \epsilon^2 V^T c^* = \Sigma^T \hat{b} + \epsilon^2 \hat{c}^*. \tag{12}$$

where $\hat{c}^* = V^T c^*$. This modified version of Equation 8 sets $\hat{c}_z$ equal to $\hat{c}_z^*$, while the entries in $\hat{c}_r$ are determined via

$$\hat{c}_k = \left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2} \right) \frac{\hat{b}_k}{\sigma_k} + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2} \right) \hat{c}_k^* \tag{13}$$

illustrating that $\hat{c}_k$ is a convex combination of the exact solution $\hat{b}_k / \sigma_k$ and the initial guess $\hat{c}_k^*$. When $\sigma_k \gg \epsilon$, the associated $\hat{c}_k$ tend toward the correct solution as usual. When $\sigma_k \ll \epsilon$, the associated $\hat{c}_k$ tend toward $\hat{c}_k^*$.

[3]This method/proof was derived for a CS205L lecture at Stanford in Winter quarter 2019 [41].

Starting with a guess of $c^* = 0$, one obtains $\hat{c}^* = 0$ and thus $\hat{c}_z^* = 0$ and $\hat{c}_z = 0$ as desired. In addition, $\hat{c}_r^* = 0$ and Equation 13 is identical to Equation 9. Multiplying by V transforms $\hat{c}$ back to the c that would result from solving Equation 11. Setting c^* equal to this newly obtained value of c and repeating the above analysis maintains $\hat{c}_z = 0$ (as desired), while

$$\hat{c}_k^* = \left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k} \tag{14}$$

so that Equation 13 becomes

$$\hat{c}_k = \left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k} + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)\left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k} = \left(1 + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)\right)\left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k}. \tag{15}$$

Repeating the entire process again results in

$$\begin{aligned}\hat{c}_k &= \left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k} + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)\left(1 + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)\right)\left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k} \\ &= \left(1 + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right) + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)^2\right)\left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k},\end{aligned} \tag{16}$$

and further iterations give

$$\hat{c}_k = \left(1 + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right) + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)^2 + \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)^3 + \cdots\right)\left(\frac{\sigma_k^2}{\sigma_k^2 + \epsilon^2}\right)\frac{\hat{b}_k}{\sigma_k}, \tag{17}$$

where the term in parentheses is a geometric series with $r = \frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}$.

Since the geometric series in Equation 17 converges to $\frac{1}{1-r} = \frac{\sigma_k^2 + \epsilon^2}{\sigma_k^2}$, Equation 17 converges to the exact solution $\hat{c}_k = \hat{b}_k/\sigma_k$. Any practical numerical method will only take q steps, leading to the partial sum

$$\frac{1 - r^q}{1 - r} = \frac{\sigma_k^2 + \epsilon^2}{\sigma_k^2}\left(1 - \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)^q\right), \tag{18}$$

which yields

$$\hat{c}_k = \left(1 - \left(\frac{\epsilon^2}{\sigma_k^2 + \epsilon^2}\right)^q\right)\frac{\hat{b}_k}{\sigma_k}. \tag{19}$$

The scalar term premultiplying $\hat{b}_k/\sigma_k$ monotonically approaches 1 as the iteration proceeds, and thus each $\hat{c}_k$ converges monotonically to the exact solution and converges more quickly for larger σ_k as desired.

4.1. Examples

Typical inverse, control, and learning problems involve numerically challenging data, where linear subproblems may have coefficient matrices with both small and identically zero singular values and the right-hand side may not be in the range of the coefficient matrix. Accordingly, we evaluate our approach against these types of problems. Here, we compare our iterative Levenberg-Marquardt (iLM) approach to PCA because PCA is a widely applied and well-understood algorithm. Since our goal is merely to demonstrate the feasibility of iLM, we utilize straightforward Matlab implementations of both methods. For iLM, we solve Equation 11 using Matlab's `pcg` routine with no preconditioner, i.e. conjugate gradients. For PCA, we compute the largest singular values and corresponding singular vectors using Matlab's `svds` function, which finds these quantities via either Lanczos bidiagonalization [5, 77] or a computation of the full SVD depending on the number of singular vectors desired. All experiments were run on a workstation equipped with Matlab R2020a, 128GB RAM, and a 24-core Intel CPU running at 3.00GHz.

First we consider rather large dense matrices and compare PCA and iLM for a varying number of singular values, noting that an increased number of iLM iterations is required for increased accuracy. We generate

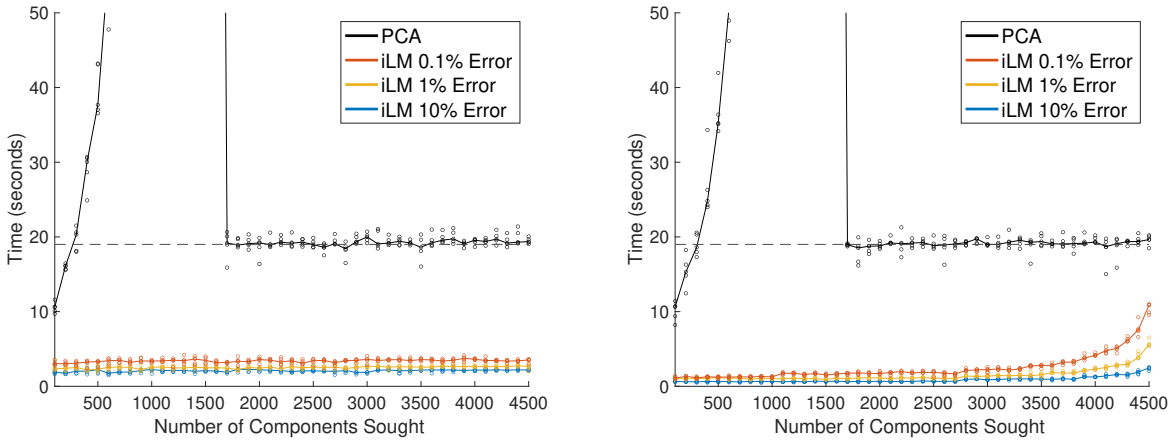


Figure 5: Performance of iLM and PCA for estimating an increasing number of $\hat{c}_k$ given a dense 5,000×5,000 square matrix with 25 million randomly-generated entries (post-processed to have 100 zero singular values) and a randomly-generated right-hand side not in its range. We observed deleterious behavior of the Matlab software for an intermediate range of sought components[4]; however, a better-devised approach would obviously not rise above the hashed line drawn in the figures. Five experiments were run for each number of components tested, and solid lines are drawn through the median results. *(Left)* $\epsilon = 0.1$. *(Right)* $\epsilon = 5.0$. Increased regularization slows convergence for singular values that are very small compared to ϵ, as expected (see Figure 6).

five random dense 5,000×5,000 matrices (each post-processed to have 100 zero singular values) as well as random right-hand sides outside the range of the coefficient matrix. Then, we estimate various numbers of components of $\hat{c}$, noting that in typical applications one seeks only a small number of components. While PCA estimates these components "exactly" up to numerical precision, the accuracy of iLM is limited by regularization (i.e. ϵ) and the tolerance of the CG solver. The number of iLM iterations is chosen so that the relevant $\hat{c}_k$ are within 10%, 1%, or 0.1% of the $\hat{c}_k$ obtained via PCA in the L_∞ norm; this required CG solver tolerances of 1e-7, 1e-8, and 1e-10, respectively. Results are shown in Figure 5. Data from each of the five trials are plotted as circles, and the median results across the five trials are connected by solid lines. The hashed line represents the approximate time taken to compute the SVD of the coefficient matrix. The number of iLM iterations required for the median results for each level of accuracy are plotted in Figure 6. To help clarify the required CG tolerance for iLM, we plot in Figure 7 the tolerance required to obtain each level of accuracy for one of the five trials.

We also consider how iLM and PCA perform for a fixed number of desired $\hat{c}_k$ as the size of the coefficient matrix increases. We let size vary from 1,000×1,000 to 10,000×10,000 and seek 500 components, creating a random dense matrix (post-processed to have 100 zero singular values) and random right-hand side not in the range of the coefficient matrix. For iLM, we iterate (as before) until the L_∞ norm of the vector of $\hat{c}_k$ is within 10%, 1%, or 0.1% of that obtained via PCA, using a CG tolerance of 1e-7, 1e-8, or 1e-10, respectively. The results are shown in Figure 8.

The aforementioned tests are unfair to iLM because they stringently require iLM to do as well as PCA on the values PCA estimates nearly exactly while ignoring the fact that PCA obtains totally inaccurate (identically zero) solutions for all the other $\hat{c}_k$. To illustrate the added benefit of smooth convergence obtained via iLM, we construct a small (to make the graphs easier to read) 100×100 random matrix with ten of its singular values set to zero (see Figure 9). A random right-hand side b outside the range of A is then formed. Figures 10 and 11 show the results of iLM and PCA, illustrating how well the obtained $\sigma_k \hat{c}_k$ reconstruct the projected right-hand side $\hat{b}_k$. iLM leverages rich information about the structure of A even when ϵ is larger than the largest singular value of A (substantial regularization). In these experiments we used a CG tolerance of 1e-6 and a maximum of 1000 CG iterations.

[4]We observed that, by default, Matlab seems to wait too long to switch from Lanczos bidiagonalization to computing the full SVD.

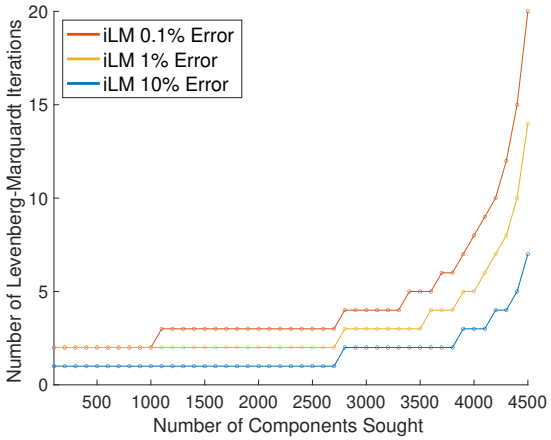


Figure 6: Number of iterations required for iLM for the medians of the trials in Figure 5 (Right) with $\epsilon = 5.0$. (When $\epsilon = 0.1$, iLM mostly converges to the desired tolerance in one iteration.)

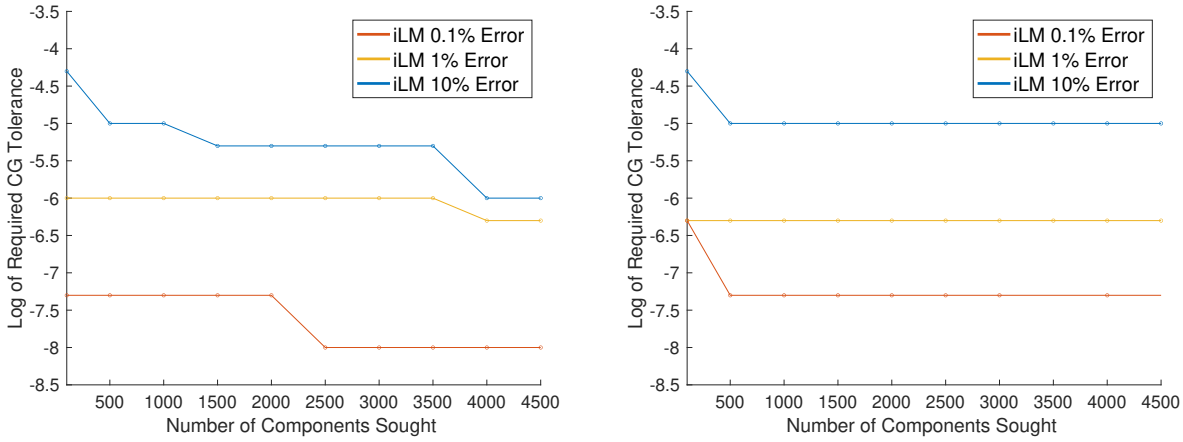


Figure 7: Using iLM, as more $\hat{c}_k$ are sought or as more accuracy is desired, a tighter CG tolerance needs to be used to prevent convergence from stalling. Plotted are the experimentally-determined maximum CG tolerances which yielded convergent results, ranging from 100 to 4500 $\hat{c}_k$ sought. *(Left)* $\epsilon = 0.1$. *(Right)* $\epsilon = 5.0$.

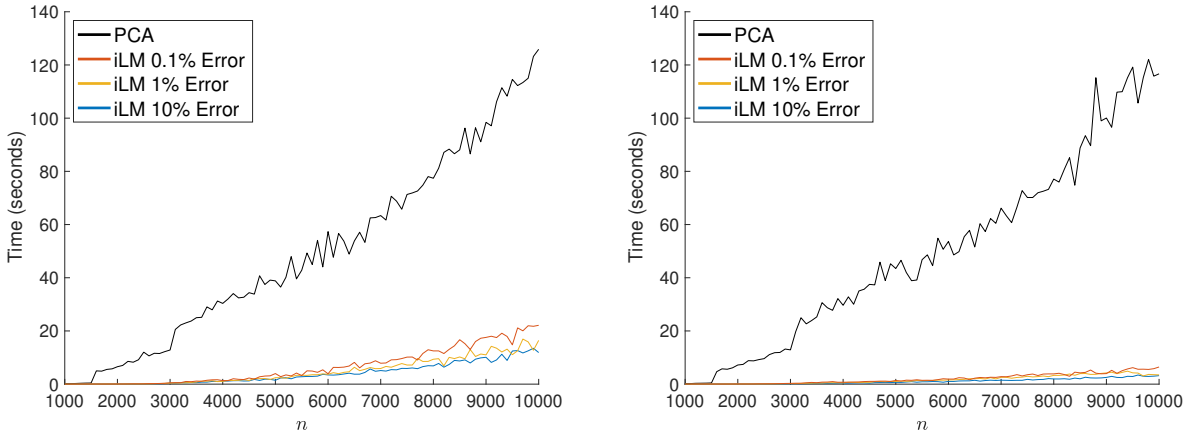


Figure 8: Performance of iLM and PCA for estimating a fixed number of $\hat{c}_k$ (500 of them) given a dense $n \times n$ square matrix with n^2 randomly-generated entries (post-processed to have 100 zero singular values) and a randomly-generated right-hand side not in its range. *(Left)* $\epsilon = 0.1$. Only one iLM iteration is required for these trials, and hence the growing cost is a combination of the increased cost per CG iteration and the number of CG iterations required. *(Right)* $\epsilon = 5.0$. For large n, the added regularization appears to aid in the convergence of CG.

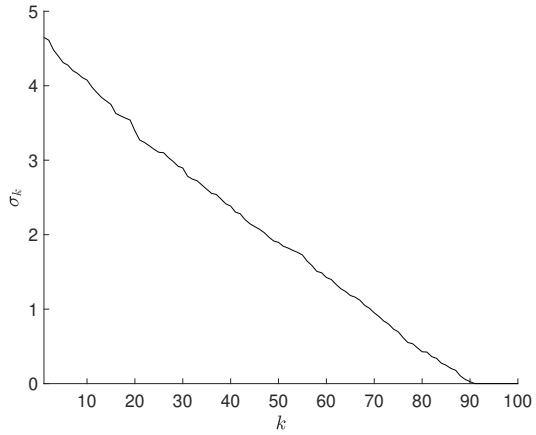


Figure 9: The singular values of the matrix used for the experiments shown in Figures 10 and 11. Ten of the singular values are identically zero.

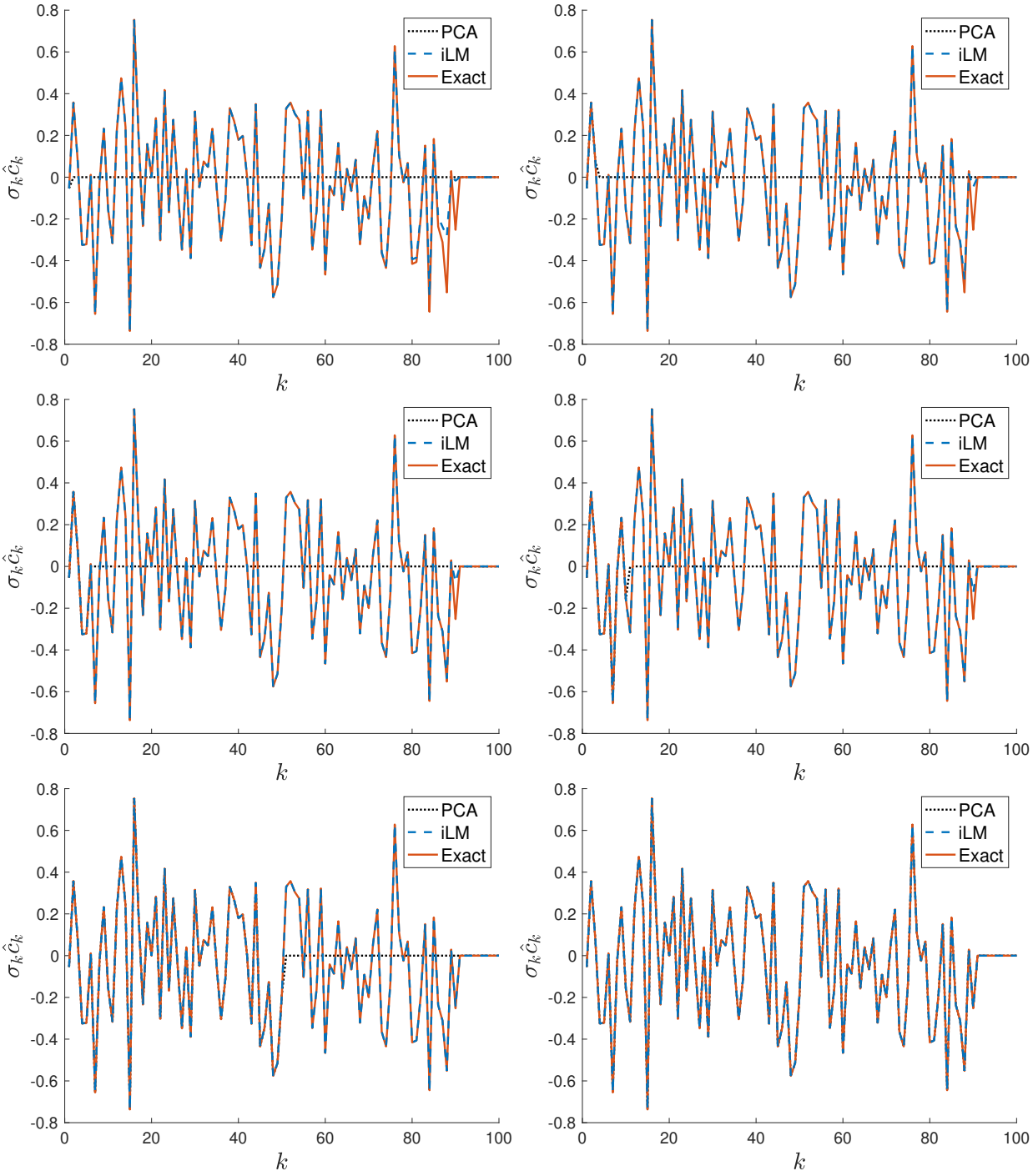


Figure 10: Convergence of iLM and PCA to the exact solutions using 1, 3, 5, 10, 50, and 100 iterations with $\epsilon = 0.1$ for iLM and 1, 3, 5, 10, 50, and 100 components for PCA, respectively. iLM converges quickly for $\hat{c}_k$ associated with larger singular values but takes additional iterations to converge for the smallest singular values due to the regularization. iLM and PCA are both exact to numerical precision for the $\hat{c}_k$ that should be identically zero (i.e. those in $\hat{c}_z$).

4.1.1. Comparisons for Nonlinear Optimization Problems

The above examples demonstrate the utility of iLM when solving linear problems such as those that arise on each iteration of a standard nonlinear optimization algorithm. We now explicitly consider solving nonlinear optimization problems using iLM and related approaches.

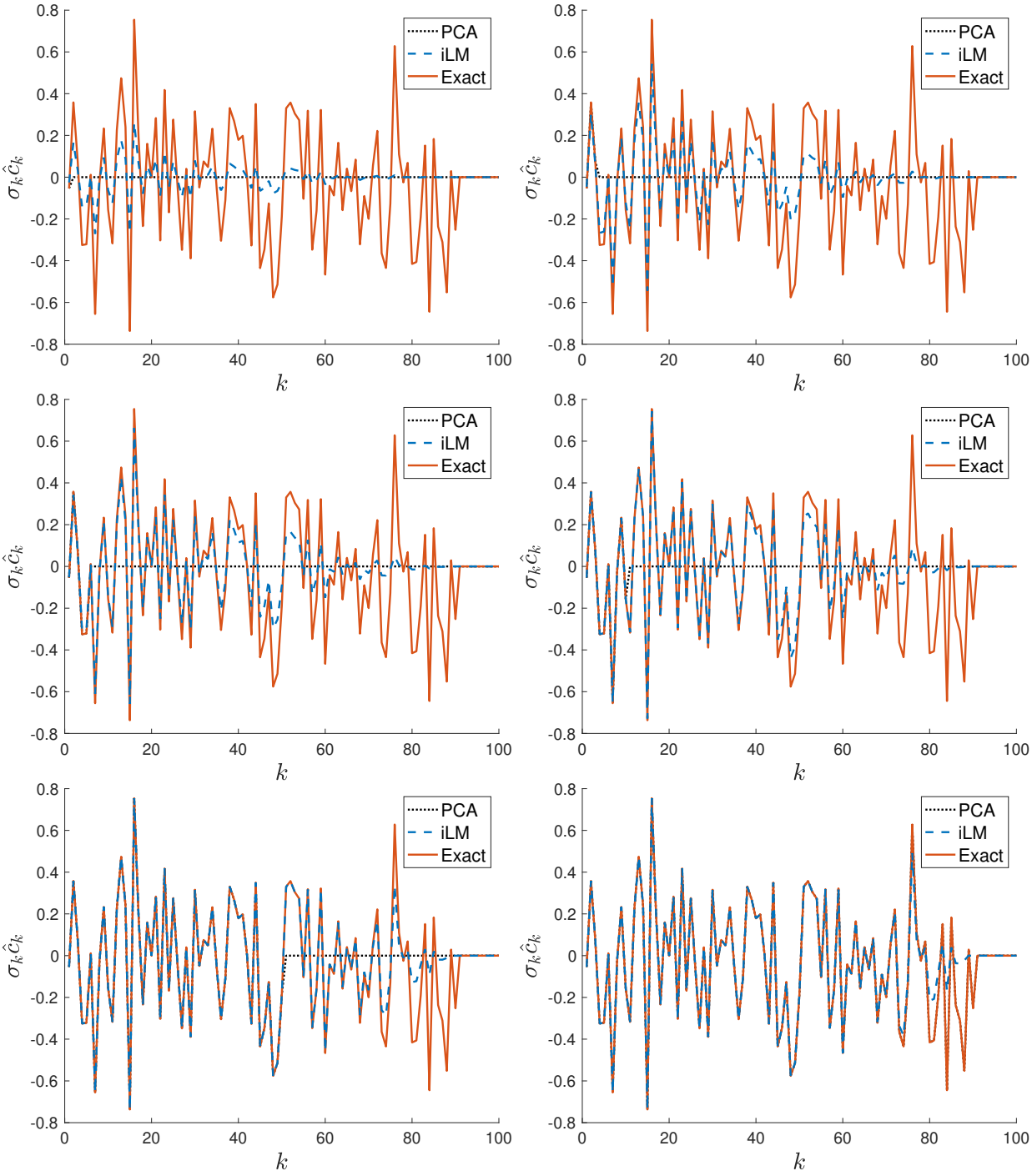


Figure 11: Same as Figure 10 except with $\epsilon = 5.0$. The increased regularization slows down iLM convergence, as expected.

We first consider an objective $f(x_1, x_2) = x_1^2 + 5x_2^2 - 4$ with an initial guess of $x^0 = \begin{pmatrix} -3 & -4 \end{pmatrix}^T$. Figure 12 shows the results of using Newton's method, gradient descent, Levenberg-Marquardt, Fan's modified Levenberg-Marquardt [40], and iLM to solve this problem. Each method is allowed to run until either the objective at the current iterate is within 10^{-6} of the analytic minimum value or until the L_2 norm of the iterate changes less than 10^{-6} between iterations. All CG solves use a tolerance of 10^{-6} and a maximum of 1,000 iterations. Since the objective is quadratic, Newton's method converges to the unique, global minimum in one iteration. Gradient descent, which lacks second-derivative information, oscillates around

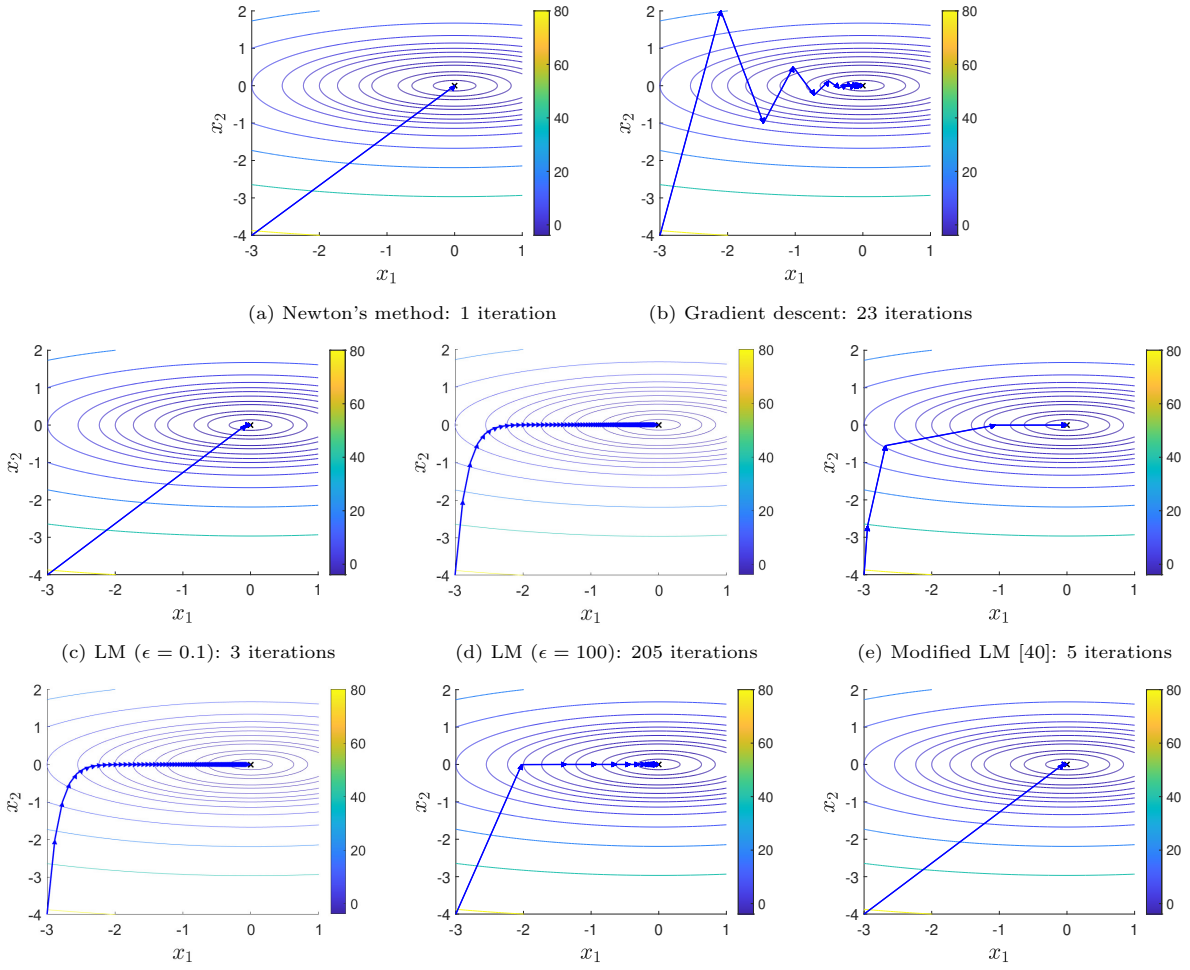
the (non-uniformly-scaled) energy landscape before eventually reaching the minimum. A learning rate of 0.15 was used. Levenberg-Marquardt can be seen as blending between the Newton and gradient descent iterates. With little regularization ($\epsilon = 0.1$), Levenberg-Marquardt looks similar to Newton's method, while with a large regularization parameter ($\epsilon = 100.0$), the number of iterations required for convergence significantly increases. The modified Levenberg-Marquardt of [40] can offer cubic convergence rates under suitable conditions by performing essentially two Levenberg-Marquardt steps on each iteration (a standard step and a forward-looking step based on the standard step). We implemented this method using the same parameters as in Section 4 of [40], except we used an initial μ_0 of 10 in order to be similar to our regularization of the other Levenberg-Marquardt variants. Finally, we consider iLM using 1, 10, or 100 iterations, all with $\epsilon = 10.0$ and using an initial guess of $c^* = 0$. Note that iLM uses an ϵ^2 rather than an ϵ scaling of the identity term, so $\epsilon = 10.0$ is equivalent to $\epsilon = 100.0$ with Levenberg-Marquardt. iLM converges to the Newton step (when the Newton step is defined) as the number of iterations increases, so iLM has a quadratic order of convergence in the best case; though of course, like Levenberg-Marquardt, gradient descent, etc., it is possible to design parameters and scenarios which make iLM converge poorly or not at all. Moreover, we stress that various strategies for adaptive learning rates and adaptive regularization terms may improve the performance of these methods. In particular, the adaptive parameter values used for our implementation of [40] are quite useful for aiding the convergence of the method, and in practice one would want to utilize adaptive regularization schemes for Levenberg-Marquardt and iLM as well (which would likely remove many of the small steps those algorithms take as they approach the solution).

As a potentially greater challenge, we consider adding a third coordinate to our objective. We alter our initial guess to have a value of 1 along this direction. Since the objective function does not depend on this third component, it is possible for the solution iterate to drift along this additional axis, e.g. when regularization is perturbing the solution away from the true minimum. Newton's method is not applicable in this case since the Hessian becomes singular, although iLM appears to converge to what Newton's method would compute using the Hessian's pseudoinverse. Interestingly, all methods appear to converge to the solution $\begin{pmatrix}0 & 0 & 1\end{pmatrix}^T$, rather than e.g. the minimum norm solution at the origin. We also consider rotating the objective and initial guess by 30 degrees about the x_1 and x_3 axes in order to make the null space of the Hessian less obvious. However, the optimization methods we tested still reach the minimum in the same number of iterations as reported in Figure 12, except for Newton's method, which remains undefined.

Further differences in the behavior and convergence of these optimization algorithms can be elucidated by considering the slightly modified objective $f(x) = \min\left(x_1^2 + 5x_2^2 - 4, (x_1 + .1)^2 + 5(x_2 - .1)^2 - 4\right)$, which has minima at $\begin{pmatrix}0 & 0\end{pmatrix}^T$ and $\begin{pmatrix}-.1 & .1\end{pmatrix}^T$. With the same initial guess of $x^0 = \begin{pmatrix}-3 & -4\end{pmatrix}^T$, the nearest minimum in the L_2 norm is the minimum-norm solution $\begin{pmatrix}0 & 0\end{pmatrix}^T$. However, if an algorithm does not proceed directly towards this solution, it may instead converge towards the other minimum with greater norm and less sparsity. This is demonstrated in Figure 13. Newton's method converges in one step to $\begin{pmatrix}0 & 0\end{pmatrix}^T$. With enough iterations, iLM approximates the Newton step and also selects the minimum-norm solution. With fewer iterations, though, iLM behaves more like gradient descent and Fan's modified Levenberg-Marquardt, which select the non-zero minimum. Standard Levenberg-Marquardt can be driven to select different minima by tuning the regularization parameter. In general, one must consider the types of solutions one seeks to an optimization problem (e.g., minimum-norm or sparse solutions) when selecting an algorithm and its parameters. Practical considerations like this and real-world performance tradeoffs can often overshadow theoretical convergence guarantees, as seen for example with the continued ubiquity of (stochastic) gradient descent.

5. Column Space Search

For the sake of motivation, consider the 2×2 linear subproblem $Ac = b$ where $A = \begin{bmatrix}1 & -1 \\ .1 & 1 \times 10^{-6}\end{bmatrix}$ and $b = \begin{bmatrix}0 & 1\end{bmatrix}^T$. Although the right-hand side is in the range of A, it is not "easily" in the range of A; in other words, the columns of A are mostly orthogonal to b leading to a solution that utilizes large multipliers $c_1 = 1/(1 \times 10^{-6} + .1)$ and $c_2 = 1/(1 \times 10^{-6} + .1)$ on the columns of A. See Figure 14. Even though this is the exact solution to the linear subproblem, it misleadingly heavily weights columns of A that do not correlate well with the desired b. Large values of c_1 and c_2 seemingly indicate that those columns of A are



(a) Newton's method: 1 iteration

(b) Gradient descent: 23 iterations

(c) LM ($\epsilon = 0.1$): 3 iterations

(d) LM ($\epsilon = 100$): 205 iterations

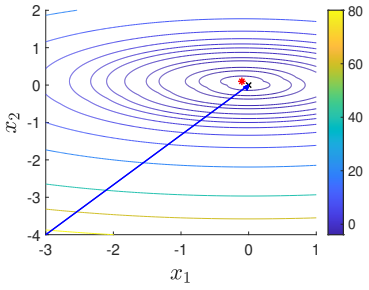
(e) Modified LM [40]: 5 iterations

(f) iLM (1 iteration): 205 iterations

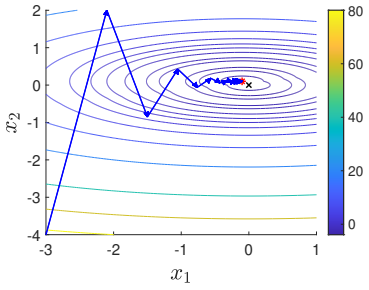
(g) iLM (10 iterations): 21 iterations

(h) iLM (100 iterations): 3 iterations

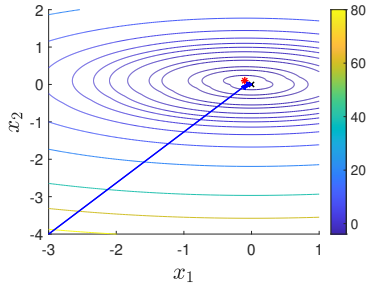
Figure 12: Different algorithms applied to minimizing $f(x_1, x_2) = x_1^2 + 5x_2^2 - 4$ with an initial guess of $x^0 = \begin{pmatrix} -3 & -4 \end{pmatrix}^T$. Contours of the function are drawn and shaded by contour value. Arrows indicate steps taken on each iteration of the optimization as the algorithm is allowed to converge to $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ (the black x). The number of iterations required for each method to converge to a tolerance of 10^{-6} is reported. We emphasize that these methods have different computational requirements; for instance, a Levenberg-Marquardt (LM) step requires one linear solve, while an iteration of [40] requires two.



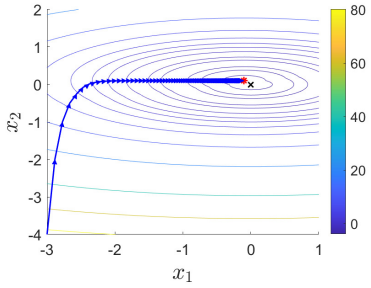
(a) Newton's method: 1 iteration



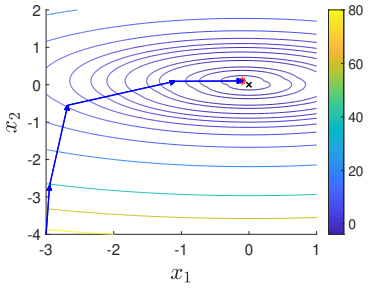
(b) Gradient descent: 23 iterations



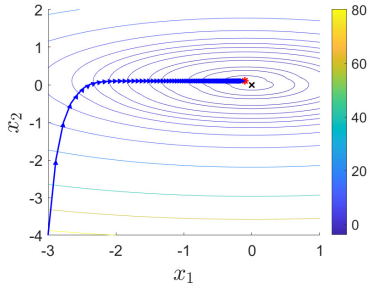
(c) Levenberg-Marquardt ($\epsilon = 0.1$): 3 iterations



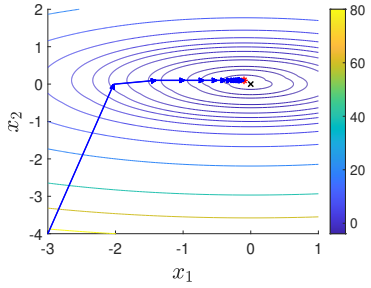
(d) Levenberg-Marquardt ($\epsilon = 100$): 204 iterations



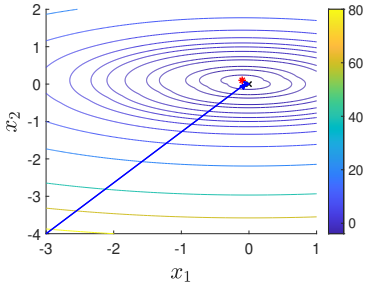
(e) Modified LM [40]: 5 iterations



(f) iLM (1 iteration): 204 iterations



(g) iLM (10 iterations): 21 iterations



(h) iLM (100 iterations): 3 iterations

Figure 13: Repeating the experiment of Figure 12 with an objective of $f(x_1, x_2) = \min\left(x_1^2 + 5x_2^2 - 4, (x_1 + .1)^2 + 5(x_2 - .1)^2 - 4\right)$. Algorithms converge to either $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ (the black x) or to $\begin{pmatrix} -.1 & .1 \end{pmatrix}^T$ (the red star).

important, even though they mostly cancel each other out being nearly orthogonal to the right-hand side. The regularized least squares problem $\min_c ||b - Ac||_2^2 + \lambda ||c||_2^2$ reduces the values of c_1 and c_2, although it does not alleviate the fact that these columns mostly work to cancel each other out, making minimal progress towards b. At best, heavy regularization could drive c_1 and c_2 even further towards zero.

As previously discussed, the columns of the linear subproblem are often quite erroneous approximations to the Hessian, which itself is only a linearization of the nonlinear problem; yet the linear subproblem is often solved and used to increment the solution vector (i.e. via $c^{q+1} = c^q + \Delta c^q$). The original nonlinear problem may include significant noise and heavy regularization, and thus it seems more important to focus on controls that make direct progress towards energy/loss minimization than those that make only incidental progress while competing with and largely cancelling each other. Thus, we advocate dropping parameters from consideration when the gains made toward the solution by some combination of those parameters are incidental compared to the parameters' main actions. As discussed previously, in regard to neural networks, this allows one to identify and differentiate which building blocks of the neural network are more or less important than others. In order to identify the more important parameters, we make note of two common misconceptions/flaws in the pursuit of solving linear subproblems. First, solving the linear subproblem exactly is not necessarily desirable since the columns of A may be terrible approximations to those of the Hessian, which itself is a linearization. Second, the largest singular values of A do not necessarily represent the most important features of the problem (as is assumed by typical PCA approaches); oftentimes, the more important notion is which columns of A are well-correlated with the right-hand side b, allowing one to make clean, non-competitive progress toward the solution.

Next, consider the right-hand side $b = \begin{bmatrix} 5 & 1 \end{bmatrix}^T$, which is better correlated with at least one of the columns of A. See Figure 15. In this case, the exact solution in Figure 15b is an improvement over Figure 14 (Right), but still contains problematic cancellation. The regularized solution shown in Figure 15c makes more progress towards the solution as compared to Figure 14 (Right), except it uses a lot more of a_2 and a lot less of a_1 than one might expect given how much better correlated a_1 is with b. Regularization damps the use of a_1 hindering its progress towards the solution; as such, a_2 ends up being utilized significantly. One could obtain a better solution for this example by changing the regularization in the least squares problem to have the form $\min_c ||b - Ac||_2^2 + \lambda_1 c_1^2 + \lambda_2 c_2^2$ with $\lambda_1 = 0$. Figure 15d shows the result for $\lambda_2 = 1$ which is highly improved. One could do even better using only a_1 as shown in Figure 15e, obtained using $\min_{c_1} ||b - a_1 c_1||_2^2$. For more discussion on various regularization strategies, especially pertaining to the facial expression inverse problem described in Section 2, see [15, 25, 86, 128, 13, 64, 136, 63, 87, 20, 68, 105].

The aforementioned discussion motivates the notion of choosing only the columns of A which are most correlated with b. Such an approach can be implemented one column at a time using a basic coordinate descent algorithm [111]. Importantly, this allows one to circumvent null spaces without adding regularization, making coordinate descent an attractive option for use on ill-posed, poorly-conditioned problems. At each iteration, the column can be chosen stochastically [104] or deterministically. Popular deterministic methods for choosing the next search direction include cyclic coordinate descent [74], the Gauss-Southwell (GS) and Gauss-Southwell-Lipschitz rule [108], and the maximum block improvement (MBI) rule [28]. Instead of looking at a single column at a time, block coordinate descent can be used to update multiple columns simultaneously [130]; however, regularization may still be needed when the block of columns is poorly-conditioned or does not have full rank. See [120, 135] for more discussion. Typical coordinate descent algorithms may choose a large number of poorly correlated coordinates in place of a smaller number of more strongly correlated coordinates. Using correlation to choose the next coordinate to add to the model can alleviate this problem and is the central idea behind MBI [28], forward and backward stepwise regression [35], and LARS [38]. The latter statistical regression methods are often used to gain better prediction accuracy and interpretability of the model [59]. However, LARS converges to the least squares solution of the linear subproblem [38] because it eventually uses uncorrelated coordinates.

In order to facilitate our goal of obtaining sparse, semantic solutions to optimization problems, particularly without adding unnecessary heuristic regularization which can lead to overfitting and error, we propose[5] solving linear subproblems by first pruning away any coordinates that are geometrically uncorrelated with the right-hand side as motivated by least angle regression (LARS) [38]; then, we estimate the remaining

[5]This approach was first proposed in the following preprint: [8].

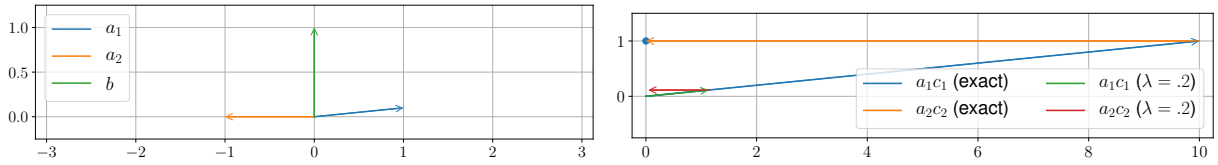


Figure 14: *(Left)* A visualization of the columns of A as well as b for the linear subproblem $Ac = b$ from Section 5 when $b = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Note how the columns of A are mostly orthogonal to b. *(Right)* The exact solution utilizes quite large values of c_1 and c_2, over-scaling largely competing columns of A in order to make progress towards b. Since the columns of A are often poor approximations to the Hessian, and the Hessian itself is only a linearization of the nonlinear problem, it seems imprudent to over-utilize controls c_1 and c_2 in order to make progress towards b. A regularized solution (with $\lambda = .2$) is also shown in the figure. It does reduce the magnitudes of c_1 and c_2 but still demonstrates the same non-desirable competitive behavior between the columns.

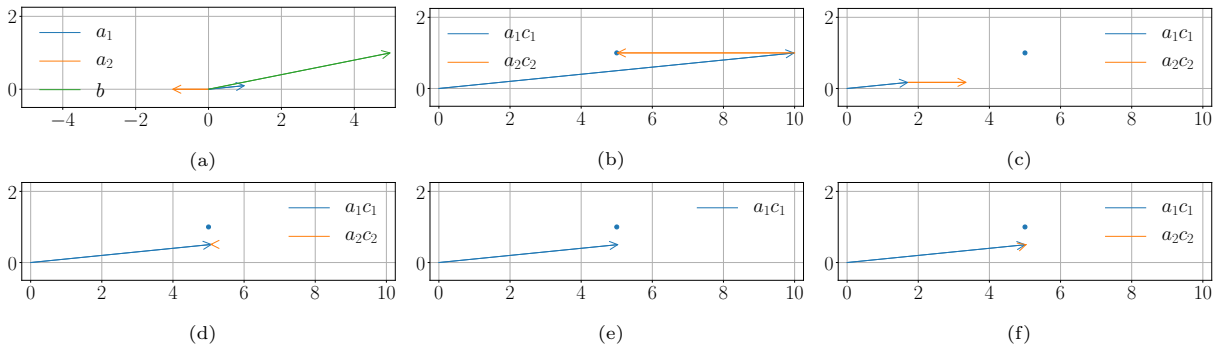


Figure 15: (a) A visualization of A's columns and b for the linear problem $Ac = b$ from Section 5 when $b = \begin{bmatrix} 5 & 1 \end{bmatrix}^T$. (b) The exact solution depicted by a_1c_1 and a_2c_2. (c) The regularized solution with $\lambda = 1$. (d) The regularized solution with $\lambda_1 = 0$ and $\lambda_2 = 1$. (e) The solution when solving for c_1 only. (f) The solution obtained after a few iterations of coordinate descent using the MBI selection rule.

coordinates via coordinate descent, eliminating the need to regularize for solvability.

5.1. Pruning Geometrically Uncorrelated Directions

We illustrate our approach, hereafter referred to as Column Space Search (CSS), by again considering solving a generic nonlinear least squares optimization problem of the form $\min_c ||f(x, y, c)||_2^2$. Using a Gauss-Newton based method, every iteration of the optimization requires solving the linear subproblem $J_{\tilde{f}}^T(c^q) J_{\tilde{f}}(c^q) \Delta c^q = -J_{\tilde{f}}^T \tilde{f}(c^q)$ to find the Δc^q subsequently used to make progress towards the solution. Again, one may equivalently consider $J_{\tilde{f}}(c^q) \Delta c^q = -\tilde{f}(c^q)$.

We first compute the geometric correlation between each column j_i of $J_{\tilde{f}}(c^q)$ and the right-hand side $-\tilde{f}(c^q)$. Similar to LARS [38] and MBI [28], we use $|\hat{j}_i \cdot \tilde{f}(c^q)|$, where $\hat{j}_i = j_i / \|j_i\|_2$. Poorly geometrically correlated columns can only make significant progress towards the solution either when partially cancelled by other poorly geometrically correlated columns (as in Figure 14 (Right)) or as corrections to better geometrically correlated columns (as in Figure 15b). However, this so-called progress, while valid for the linear subproblem, may pollute the sparsity and semantics of the solution to the original nonlinear problem. See Figure 16. Thus, we prune poorly geometrically correlated columns from $J_{\tilde{f}}(c^q)$ resulting in a lower-rank J_S. Motivated by the Gauss-Southwell rule, one might instead prune using gain correlation $|j_i \cdot \tilde{f}(c^q)|$, which considers large residual decreases with smaller variable values; however, we instead prefer removing poorly geometrically correlated columns even when they may have large gains as it seems to lead to better semantic interpretation. Additionally, one could drop the absolute value and consider $\hat{j}_i \cdot \tilde{f}(c^q)$ in order to prune columns that are only semantically sensible in one direction.

Pruning columns of $J_{\tilde{f}}(c^q)$ to get a reduced J_S has the additional benefit of potentially eliminating portions of the null space of $J_{\tilde{f}}(c^q)$, as the pruned out columns or a combination of them with the non-pruned columns may have linear dependencies; this pruning may also improve the condition number. This is especially prudent when working with a large number of dimensions, in which case the dimension of the

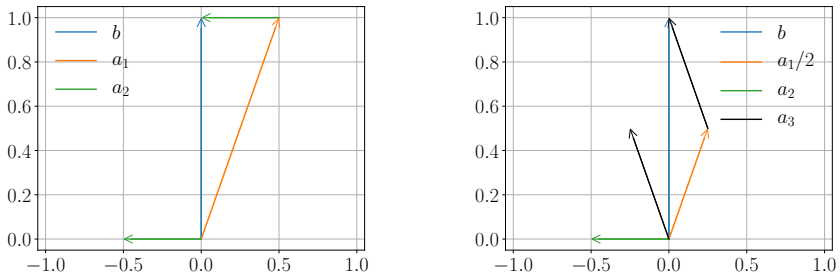


Figure 16: *(Left)* Here, $b = a_1 + a_2$, but a_2 is only valid for making progress towards b in conjunction with a_1 (and is otherwise orthogonal). While using a_2 may be desirable when trying to solve a truly linear system of equations, it makes less sense when solving a linearization of a high-dimensional nonlinear problem. *(Right)* It may be desirable to only progress in the direction of a_1 until other directions become better correlated. At that point, it would be better to find a new direction (in this case, a_3) that leads back towards b.

null space of $J_{\tilde{f}}(c^q)$ and the condition number of $J_{\tilde{f}}(c^q)$ may be quite large. Moreover, these difficulties are exacerbated when regularization is not used.

5.2. Solving the Pruned System

We avoid regularization entirely in order to avoid changing the solution to the problem; thus, we pursue a coordinate descent strategy to solve $J_S \Delta c_S^q = -\tilde{f}(c^q)$ where Δc_S^q is a subset of Δc^q. At each iteration, a single column j_i of J_S is used to make progress towards $-\tilde{f}(c^q)$. We generally only execute a few iterations to mimic the regularization effects of early stopping [53] and truncated-Newton methods [101], as it helps to prevent overfitting to the linearized subproblem or reaching the undesirable least squares solution as in LARS [38]. Furthermore, we also terminate early if the decrease in L_2 error is low.

Choosing the most geometrically correlated column j_i (as in MBI [28]) allows one to best minimize the remaining residual; however, small-magnitude columns may require large, undesirable step sizes $\alpha(j_i)$ to make progress. Instead, motivated by the Gauss-Southwell rule [108], we choose the column j_i that maximizes a discretized ratio of residual reduction to step size, i.e.

$$\frac{\Delta(r^T r)}{\Delta \alpha} = \frac{||r(\Delta c_S^q)||_2^2 - ||r(\Delta c_S^q) - \alpha(j_i) j_i||_2^2}{|\alpha(j_i)|}, \tag{20}$$

where $r(\Delta c_S^q)$ is the current residual as a function of the current estimate for Δc_S^q. In addition, $\alpha(j_i)$ is the step size obtained when choosing column j_i. Flipping all j_i so that $r(\Delta c_S^q)^T j_i > 0$ leads to $\alpha(j_i) > 0$, which allows one to equivalently maximize

$$M = \frac{r(\Delta c_S^q)^T r(\Delta c_S^q) - \left(r(\Delta c_S^q)^T r(\Delta c_S^q) - 2\alpha(j_i) r(\Delta c_S^q)^T j_i + (\alpha(j_i))^2 j_i^T j_i \right)}{\alpha(j_i)} \tag{21a}$$

$$= 2 r(\Delta c_S^q)^T j_i - \alpha(j_i) ||j_i||_2^2 \tag{21b}$$

$$= r(\Delta c_S^q)^T j_i + (r(\Delta c_S^q) - \alpha(j_i) j_i)^T j_i. \tag{21c}$$

The greedy choice of $\alpha(j_i)$ removes as much of the residual as possible, setting $\alpha(j_i) j_i = (r(\Delta c_S^q) \cdot \hat{j}_i) \hat{j}_i$ or

$$\alpha(j_i) = (r(\Delta c_S^q) \cdot j_i) / ||j_i||_2^2, \tag{22}$$

which zeros out the second term in Equation 21c leaving only $r(\Delta c_S^q)^T j_i$, i.e. gain correlation.

There are two subtleties to consider regarding Equations 20–22. First, we do not necessarily use columns with the largest gains because, as discussed in Section 5.1, we prune away poorly geometrically correlated columns before considering Equations 20–22. Second, one typically limits the size of $\alpha(j_i)$ when training neural networks and/or solving optimization/control problems, see e.g. trust region methods [124, 45, 107], adaptive step sizes for temporal numerical integration [43, 47], and adaptive learning rate techniques such as Adam [72], ADADELTA [139], etc. Thus, shorter j_i will not necessarily yield the greedy $\alpha(j_i)$ shown in Equation 22, leaving the second term in the last line of Equation 21c non-zero. See Figure 17.

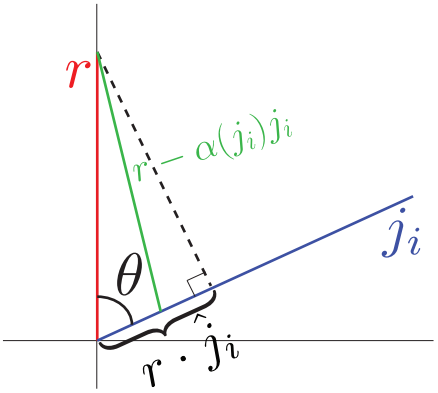


Figure 17: For longer j_i, the $\alpha(j_i)$ required to take the greedy step (Equation 22) will be small enough such that it is not clamped via various safe set or trust region considerations, resulting in the second term in Equation 21c being identically zero. However, for shorter j_i, $\alpha(j_i)$ may be clamped, resulting in an $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ which is not perpendicular to j_i (shown in green in the figure). In this case, the second term in Equation 21c is non-zero, and the gain correlation of the remaining residual $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ with the search direction j_i penalizes search directions that become poorly gain correlated after using them. With regard to Figure 16, this prefers a scenario using $a_1/2$ and a_3 as in Figure 16 (Right) as opposed to using a_1 and a_2 in Figure 16 (Left).

Consider bounding the step size $\alpha(j_i)$ from above with some $\alpha_{\max}$. One can choose a reference frame such that $r\left(\Delta c_S^q\right)$ is a unit vector along the y-axis and j_i is in the first quadrant of the xy-plane, as shown in Figure 18. Referring to the greedy $\alpha(j_i)$ in Equation 22, we plot curves representing vectors j_i where the greedy $\alpha(j_i)$ is equal to 1/1.5, 1, and 1/.75 in the figure. When bounding $\alpha(j_i)$ from above by some $\alpha_{\max}$, the $\alpha_G = \alpha_{\max}$ curve represents the boundary between the tips of longer vectors that can use the greedy $\alpha(j_i)$ and the tips of shorter vectors where $\alpha(j_i)$ would be clamped. In particular, for the $\alpha_{\max} = 1$ case, the yellow region in the figure represents the tips of longer vectors and the green region represents the tips of shorter vectors. In the green region, the second term in Equation 21c is added to the usual $r\left(\Delta c_S^q\right)^T j_i$ gain correlation, increasing preference for search directions that remain well-correlated after using them. Figure 18 (Right) shows the magnitude of the second term in Equation 21c. Additionally, one could multiply the second term in Equation 21c by an arbitrary scaling constant and increase its influence.

We also consider the case of clamping based on total progress. Figure 19 uses the same reference frame as Figure 18 but draws the boundary where $||j_i||_2 = ||r\left(\Delta c_S^q\right)||_2/2$. Figure 19 (Left) draws three search directions taking the greedy step. Regardless of the length of j_i, the greedy $\alpha(j_i)$ rescales such that $\alpha(j_i)j_i$ ends at the boundary between the green and yellow regions where $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ is orthogonal to j_i. Figure 19 (Right) shows how clamping the progress limits the ability of a search direction to take the greedy step, resulting in the second term in Equation 21c being non-zero. Another way of choosing $\alpha(j_i)$ is based on the observation that $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ is always less geometrically correlated with j_i than $r\left(\Delta c_S^q\right)$ is, since it points to the left instead of upwards. Hence, one could choose α in order to bound how much worse the gain/geometric correlation of $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ is allowed to have compared to that of $r\left(\Delta c_S^q\right)$. In general, there are many potential strategies, but in all such cases, our methodology is to first prune so that large gain correlation does not introduce poorly geometrically correlated vectors, and then to consider correlation of the new residual $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ in addition to correlation of the current residual in order to favor scenarios like Figure 16 (Right) over Figure 16 (Left).

5.3. Examples

We consider the problem of determining parameters w that best match a three-dimensional synthetic face model to a real image, as discussed in Section 2. Although we used CSS to generate Figure 4 and for related efforts, here we consider a slightly modified situation in order to better isolate and demonstrate the behavior of CSS. Let w represent the controls for the face blendshapes, jaw angles, and jaw translation, and $x(w)$ represent the synthetic three-dimensional face surface obtained from w. We replace the inference-based neural network keypoint detector with a more deterministic artist-drawn rotoscoping of curves for the eyes and mouth, as shown in Figure 20 (Left). In order to generate comparable keypoints on the synthetic face

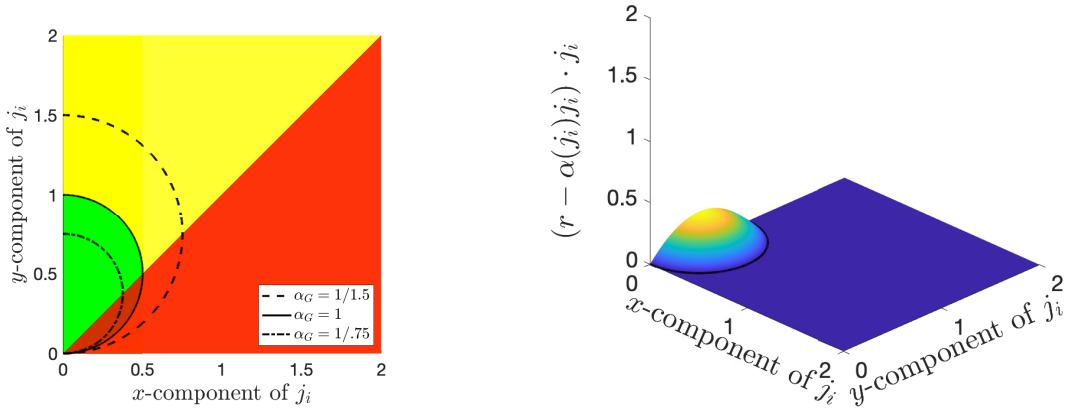


Figure 18: We choose a reference frame where $r\left(\Delta c_S^q\right)$ is a unit vector along the y-axis and j_i is in the first quadrant of the xy-plane. *(Left)* Poorly geometrically correlated vectors (those with tips in the red region) are pruned as in Section 5.1. Referring to the greedy $\alpha(j_i)$ in Equation 22, we plot curves representing the tips of vectors j_i where the greedy $\alpha(j_i)$ is equal to 1/1.5, 1, and 1/.75. When bounding $\alpha(j_i)$ from above by $\alpha_{\max}$, the $\alpha_G = \alpha_{\max}$ curve represents the boundary between the tips of longer vectors that can use the greedy $\alpha(j_i)$ and the tips of shorter vectors where $\alpha(j_i)$ would be clamped. In particular, for the $\alpha_{\max} = 1$ case, the yellow region represents the tips of longer vectors and the green region represents the tips of shorter vectors. *(Right)* The magnitude of the second term in Equation 21c for the $\alpha_{\max} = 1$ case. Note that it is non-zero only for shorter vectors with tips inside the $\alpha_G = 1$ curve.

model, we draw corresponding curves barycentrically embedded on the three-dimensional face geometry. Then, $x(w)$ determines the three-dimensional location of these barycentrically embedded curves, which subsequently are projected into the image plane using calibrated camera intrinsic and extrinsic parameters [62]; this simple projection replaces the differentiable renderer. See Figure 20 (Right). In order to obtain comparable keypoints, we label easily-identifiable locations on both sets of curves (i.e. those drawn on the synthetic model and those drawn on the image), e.g. corners of the mouth and eyes, middles of the lips, etc. To increase the number of comparable keypoints, we uniformly sample between the projected (into the image plane) locations of the easily-identifiable keypoints. Letting C^* be the two-dimensional keypoints on the real image and $C(x(w))$ be the corresponding projected keypoints determined by the parameters w of the synthetic model, we then solve

$$\min_w \|C^* - C(x(w))\|_2^2 \tag{23}$$

in order to recover the parameters w that best match the two sets of keypoints together.

For comparison against CSS, we consider solving Equation 23 using Dogleg [112, 94] with no prior, Dogleg with a prior weight of $\lambda = 3600$, and BFGS [107] with a soft-L_1 prior with a weight of 3600 (i.e. with an extra term $3600 \sum_i 2(\sqrt{1 + w_i^2} - 1)$ [27]). When solving with CSS, we first prune all columns whose angle to the residual has an absolute cosine less than 0.3. Then, $\alpha(j_i)$ is set to a fixed size of 0.01 and coordinate descent is run until the linear L_2 error no longer sufficiently decreases or when over 10 coordinates are used. We limit all four methods to at most 10 Gauss-Newton linearization iterations. Figure 21 shows the results. CSS and methods using regularization give the most reasonable geometric results. A major benefit of CSS is the resulting sparsity of the weights: while Dogleg with and without regularization sets nearly all the parameters to a non-zero value, CSS generally uses only a small number of non-zero weights. The soft L_1 regularized solution is sparser than the L_2 regularized solution; however, due to approximations in the chosen optimization approach (BFGS, Soft L_1), it produces many small (i.e. $< 1 \times 10^{-3}$) weights instead of identically zero values. While one could clamp small values to zero, care must be taken to not accidentally clamp weights that contribute significantly to the overall solution.

To further elucidate the performance of these approaches, we construct a known exact solution and subsequently add an increasing amount of noise. First, an exact set of keypoints is determined by subsampling the contours on the three-dimensional face geometry. Then, a smile expression is created by setting two specific components of w to 1 while the other components remain set to zero. This known w^* determines

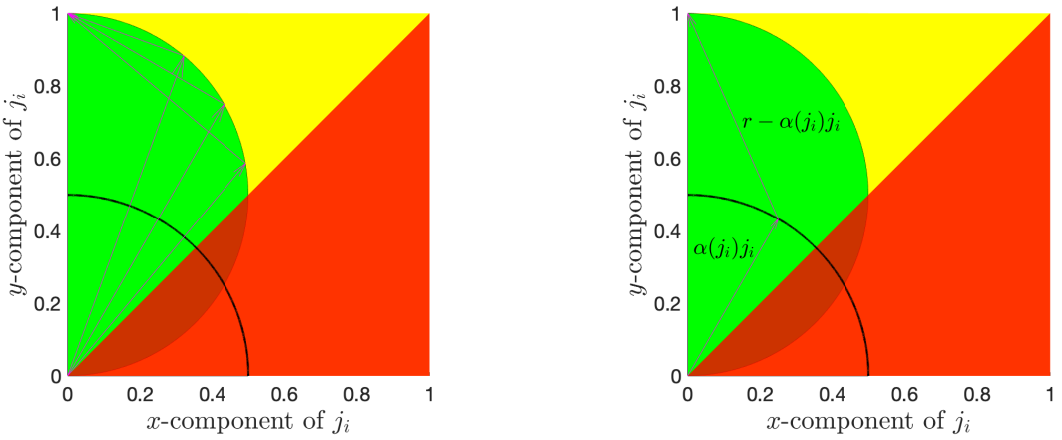


Figure 19: We use the same reference frame as in Figure 18 but draw the boundary where $||j_i||_2 = ||r\left(\Delta c_S^q\right)||_2/2$. The yellow and green regions are shaded as in the $\alpha_{\max} = 1$ case from Figure 18. *(Left)* Three search directions are drawn taking the greedy step. Regardless of the length of j_i, the greedy $\alpha(j_i)$ rescales such that $\alpha(j_i)j_i$ ends at the boundary between the green and yellow regions where $r\left(\Delta c_S^q\right) - \alpha(j_i)j_i$ is orthogonal to j_i. *(Right)* Clamping progress limits the ability of a search direction to take the greedy step, resulting in the second term in Equation 21c being non-zero.

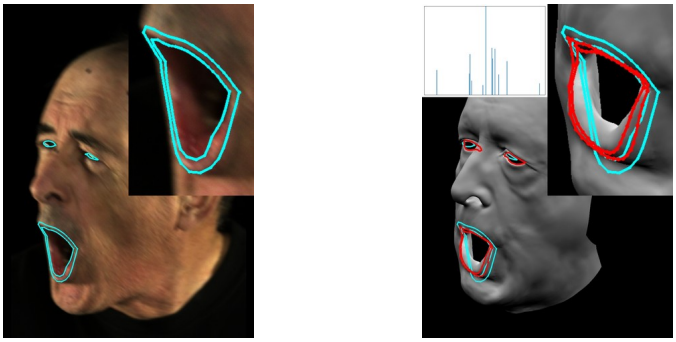


Figure 20: *(Left)* Hand-drawn rotoscope curves on a real image. *(Right)* Barycentrically embedded curves on the three-dimensional geometric facial model are projected into the image plane (red) and compared to curves drawn/rotoscoped on the real image (blue). The inset shows the facial parameters w for this pose.

face geometry $x(w^*)$ along with barycentrically embedded keypoints that can be projected into the image plane to determine C^*. The results obtained are shown in Figures 22a and 22d and are similar to those obtained using the real image data in Figure 21. Next, we add an increasing amount of uniformly distributed noise to C^* in the image plane. As expected, Dogleg with no regularization produces reasonable results when the noise is low but begins to overfit to the erroneous data as the amount of noise increases, see Figure 22 (first row). Both of the regularized approaches as well as CSS are able to target the noisy curves without overfitting, producing more reasonable geometry. The right half of Figure 22 shows that CSS yields sparser, more semantic solutions even with the added noise. Tables 1 and 2 demonstrate quantitative results for these examples. As seen in Table 1, CSS performs the best in all cases as measured by various metrics. Table 2 uses two sparsity measures: the l^0 metric counts how many facial parameters are strictly 0, and the Gini metric is $1 - 2\sum_{i=1}^{\hat{w}_i/||\hat{w}||_1}(\frac{N-i+0.5}{N})$, where $\hat{w}$ are the sorted parameters with $\hat{w}_i$ the i^{th} largest [66]. Note how regularization improves the Gini metric, but does not necessarily improve the l^0 metric.

5.4. Parameter Study

Column Choice. Returning to the solution of Equation 23 for the real image data, we compare our approach for choosing the next coordinate descent column (Section 5.2) to using Gauss-Southwell and MBI. For each approach, we linearize and solve with no thresholding for the relative decrease in L_2 error, an upper limit of 10 unique coordinates used, and a fixed step size of 0.01; in these examples, we remove the eye rotoscope

Table 1: Comparing the accuracy of estimating the facial parameters in the synthetic tests under various metrics. CSS produces the best results regardless of noise and metric.

	L_2 Error			L_1 Error			EMD [116] Error		
Method	No Noise	0.005	0.01	No Noise	0.005	0.01	No Noise	0.005	0.01
Dogleg	2.578	8.815	20.325	19.227	70.852	157.22	0.128	0.485	1.07
Dogleg+L_2	0.972	0.952	1.209	4.954	5.324	5.704	0.034	0.036	0.039
BFGS+Soft L_1	0.923	0.91	1.023	3.139	3.057	3.359	0.0215	0.021	0.023
CSS	**0.741**	**0.392**	**0.509**	**2.208**	**0.99**	**1.08**	**0.015**	**0.007**	**0.007**

Table 2: The sparsity of the results of the synthetic tests using common sparsity metrics (a larger number is better).

	l^0 Metric			Gini Metric		
Method	No Noise	0.005	0.01	No Noise	0.005	0.01
Dogleg	21	21	21	0.628	0.580	0.607
Dogleg+L_2	21	21	21	0.807	0.745	0.739
BFGS+Soft L_1	21	21	21	0.913	0.905	0.916
CSS	**128**	**137**	**140**	**0.949**	**0.974**	**0.978**

curves from the energy function and only consider curves drawn around the mouths on the image and model. Results are shown in Figure 23. MBI overfits and overuses mouth blendshapes, e.g. the two most heavily weighted shapes have magnitudes of 85.78 and 63.12. On the other hand, Gauss-Southwell and CSS keep the parameters within a reasonable range while maintaining the sparsity of the solution. We note that with coordinate descent it is generally a matter of when, not if, the algorithm chooses a coordinate that will be overused/overweighted; our examples demonstrate that MBI chooses those coordinates more quickly than Gauss-Southwell and CSS.

Step Size & Convergence. Since the problem has been normalized so that the $\alpha(j_i)$ generally make most sense between 0 and 1, here we compare fixed step sizes of $\alpha(j_i) = 0.01, 0.02, 0.1, 0.5$ and 1.0 to the full, greedy step in Equation 22. Without pruning, we run 10 Gauss-Newton iterations with no thresholding for the relative decrease in L_2 error and an upper limit of 10 unique coordinates used. We find that smaller step sizes achieve better overall facial shapes and less overused parameters (see Figure 24). In particular, the greedy step sets 7 parameters to be greater than 1 while step sizes of 0.02 and 0.01 only set 4. Removing the eye rotoscope curves causes the overused parameters to disappear; however, as seen in Figure 25, the greedy step causes the mouth to move unnaturally. This would seem to indicate that always taking the greedy step will result in some overfitting.

We also compare the effect of using fixed step sizes in Equation 20 versus the full, greedy step size equivalent to Gauss-Southwell without pruning. To isolate this variable, we run 10 Gauss-Newton iterations with no thresholding for the relative decrease in L_2 error and an upper limit of 10 unique coordinates used. We vary $\alpha(j_i)$ in Equation 20 but set the actual step size taken to be fixed at 0.01. As shown in Figure 26, while the resulting geometry and weights are all similar, our approach of allowing the step size to influence the chosen coordinate allows the optimization to more quickly reduce the error in earlier Gauss-Newton iterations than when using Gauss-Southwell (see Figure 27). Therefore, it may be beneficial to use CSS with a fixed size step when only a few Gauss-Newton iterations are desired.

Pruning. We rescale r to $\hat{r} = r/||r||$ and then compare different threshold values for pruning: 0.0 (no pruning), 0.2, 0.3, and 0.5. We run 10 Gauss-Newton iterations with a step size of 0.01 with no thresholding for the relative decrease in L_2 error. To emphasize the effect of pruning, we allow up to 50 unique coordinates per linearization, and focus only on the rotoscope curves around the mouth. With little to no pruning the model overfits and the geometry around the mouth deforms unreasonably. As the pruning threshold increases,

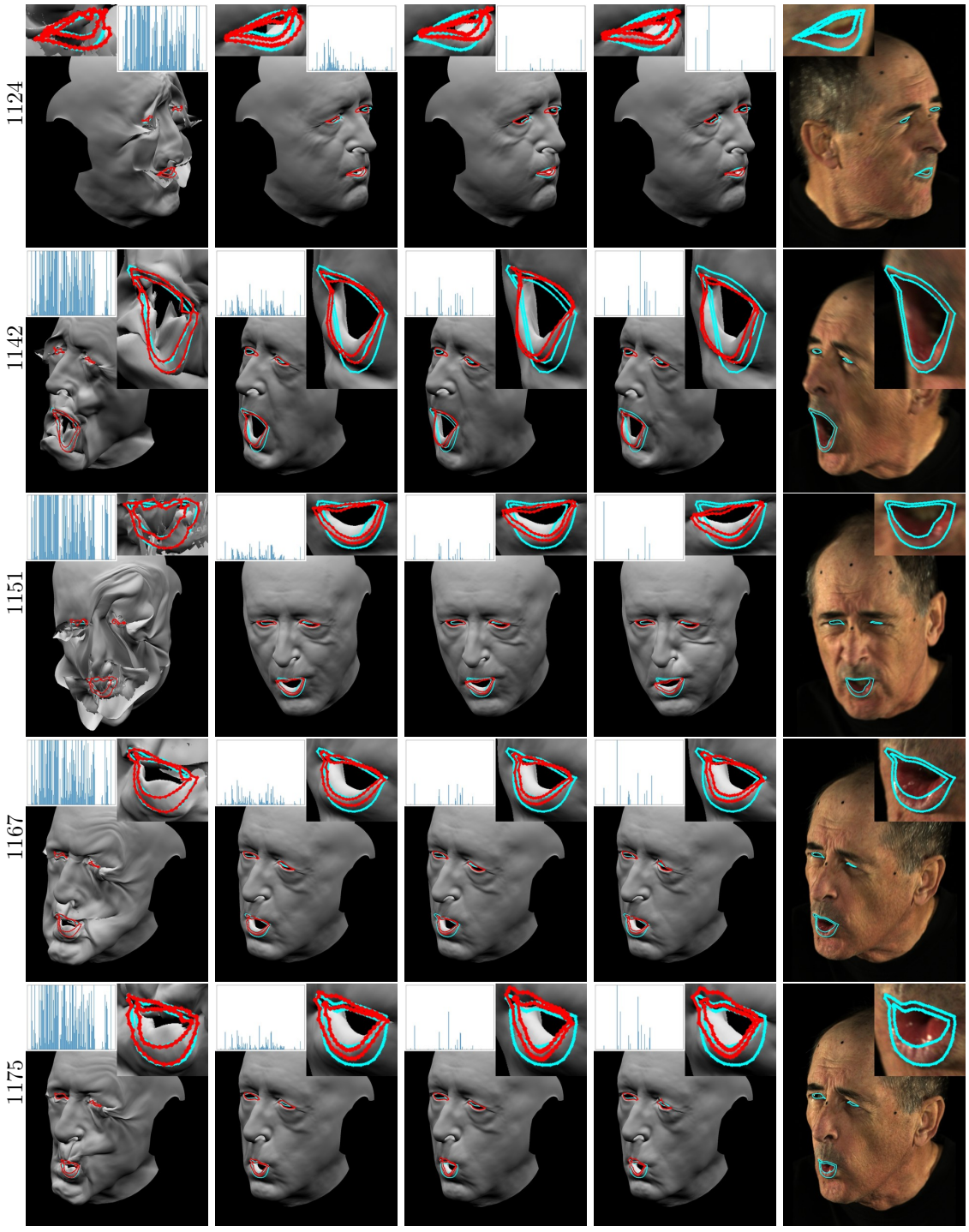
the geometry becomes more regularized and the facial parameters are sparser, as the optimization is forced to use only the most correlated directions. See Figure 28. However, we caution that too much pruning causes MBI style column choices.

6. Conclusions

In difficult nonlinear problems such as the one described in Section 2, one often solves linear subproblems to make progress. Although PCA is quite popular for solving such problems, especially when there are issues with null spaces and the right-hand side not being in the range of the linearized system, we showed that our iLM method not only efficiently monotonically converges to the exact solution of the linearized subproblem, but does so more smoothly. We subsequently pointed out that the larger singular values of the coefficient matrix can be less important than considering which controls are optimal for obtaining the right-hand side. These considerations motivated our column space search (CSS) approach. We chose a complex real-world problem, estimating three-dimensional facial expressions from a mere eight contours drawn on a single two-dimensional RGB image, that allows even non-experts to simply glance at an image and comprehend the effects of noise, overfitting, and regularization. We were able to robustly estimate clean sparse parameter values with good semantic meaning in a highly underconstrained situation where one would typically need significant regularization. In fact, the standard approach without regularization was wildly inaccurate, and although regularization helped to moderate the overall face shape, it excited almost every parameter in the model, clouding semantic interpretation.

Acknowledgements

Research supported in part by ONR N00014-13-1-0346, ONR N00014-17-1-2174, ARL AHPCRC W911NF-07-002, and generous gifts from Amazon and Toyota. In addition, we would like to thank both Reza and Behzad at ONR for supporting our efforts into computer vision and machine learning, as well as Cary Phillips and Industrial Light & Magic for supporting our efforts into facial performance capture. M.B. was supported in part by The VMWare Fellowship in Honor of Ole Agesen. We would also like to thank Paul Huston for his acting.



(a) Dogleg (b) Dogleg+L_2 (c) BFGS+Soft L_1 (d) CSS (e) Target

Figure 21: Dogleg without regularization clearly overfits to the curves, producing highly unrealistic face shapes. Dogleg with regularization performs better but sometimes overfits as well. This could be tuned by increasing the regularization weight at the cost of potentially damping out the performance. Our approach produces facial expressions that are reasonably representative of the captured image. The inset bar plots demonstrate the sparsity of the weights for each of the methods. Our method generally produces the sparsest set of weights; e.g. in frame 1142, our method has 12 non-zero parameter values while L_2 regularization produces fully dense results and soft L_1 regularization has 49 significant parameter values (i.e. $> 1 \times 10^{-3}$).

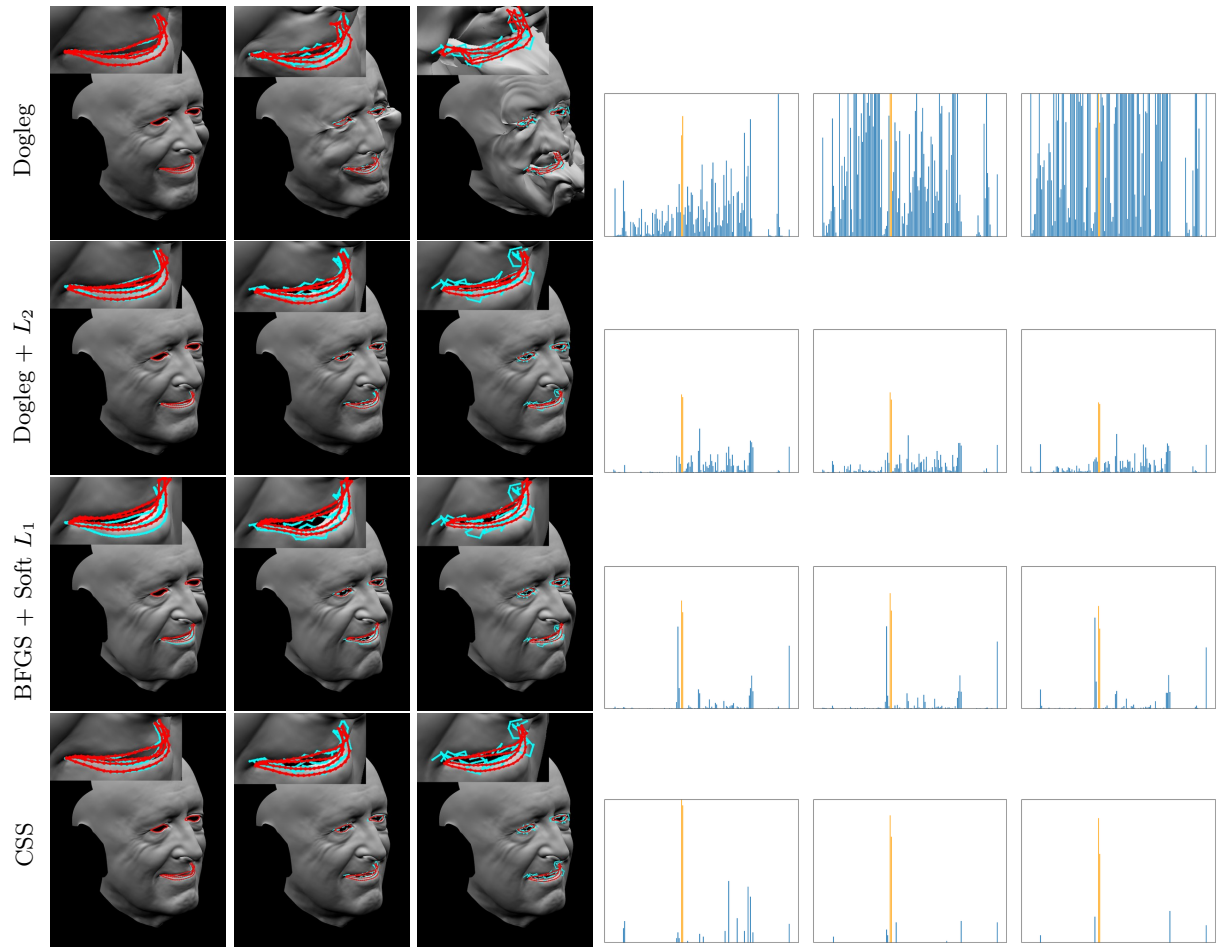
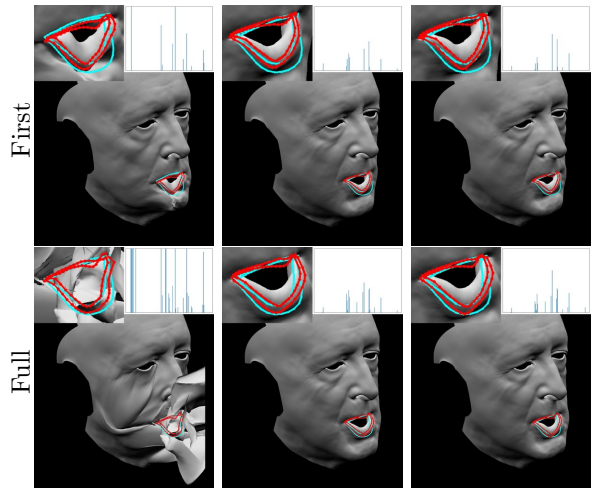
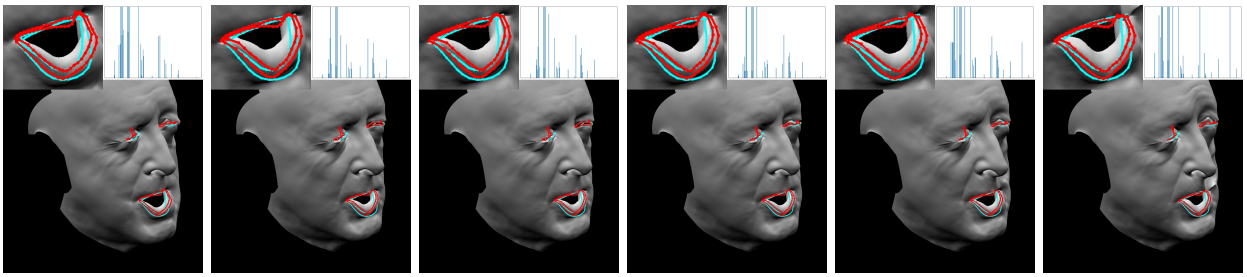


Figure 22: A synthetic test where a known w^* is used to create blue target curves. *(Left)* As we increase the amount of noise added to the points on the blue target curve, the Dogleg method without regularization overfits causing the mesh to "explode" in spite of having the smallest error as measured by Equation 23 (typical of overfitting). On the other hand, both standard regularization and our approach prevent the model from overfitting to the noisy curves. *(Right)* The corresponding facial parameters. The target solution was generated by setting the two orange columns to one and the blue columns to zero. The figure heights are clipped at 1.0 and many parameter values exceed that. Though the regularized solves have smaller, spurious weights than the non-regularized version (second and third row vs. first row), our approach (last row) produces a much sparser solution with more semantic meaning even in the presence of noise.



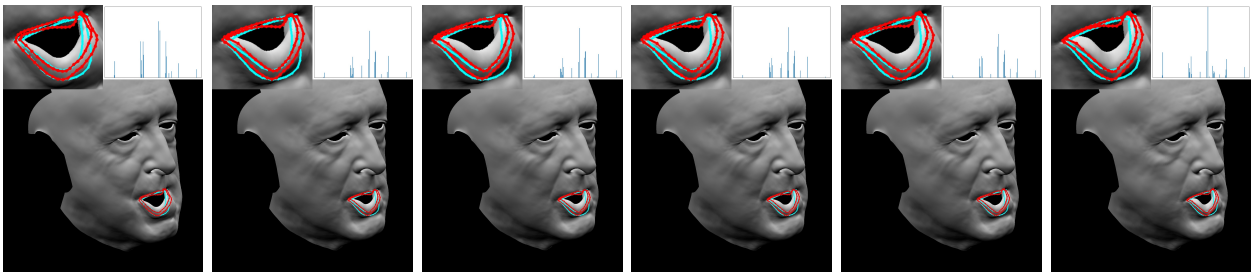
(a) MBI (b) GS (c) CSS

Figure 23: A comparison of the coordinates chosen by the MBI rule, the Gauss-Southwell (GS) rule, and CSS when solving without the eye rotoscope curves. The top row are the results after a single Gauss-Newton iteration, and the bottom row are the results after 10 Gauss-Newton iterations.



(a) Greedy (b) 0.01 (c) 0.02 (d) 0.1 (e) 0.5 (f) 1.0

Figure 24: We compare the behavior of the geometry and the facial parameters when using different step sizes.



(a) Greedy (b) 0.01 (c) 0.02 (d) 0.1 (e) 0.5 (f) 1.0

Figure 25: A comparison of the behavior of the geometry and the facial parameters when using different step sizes without the eye rotoscope curves.

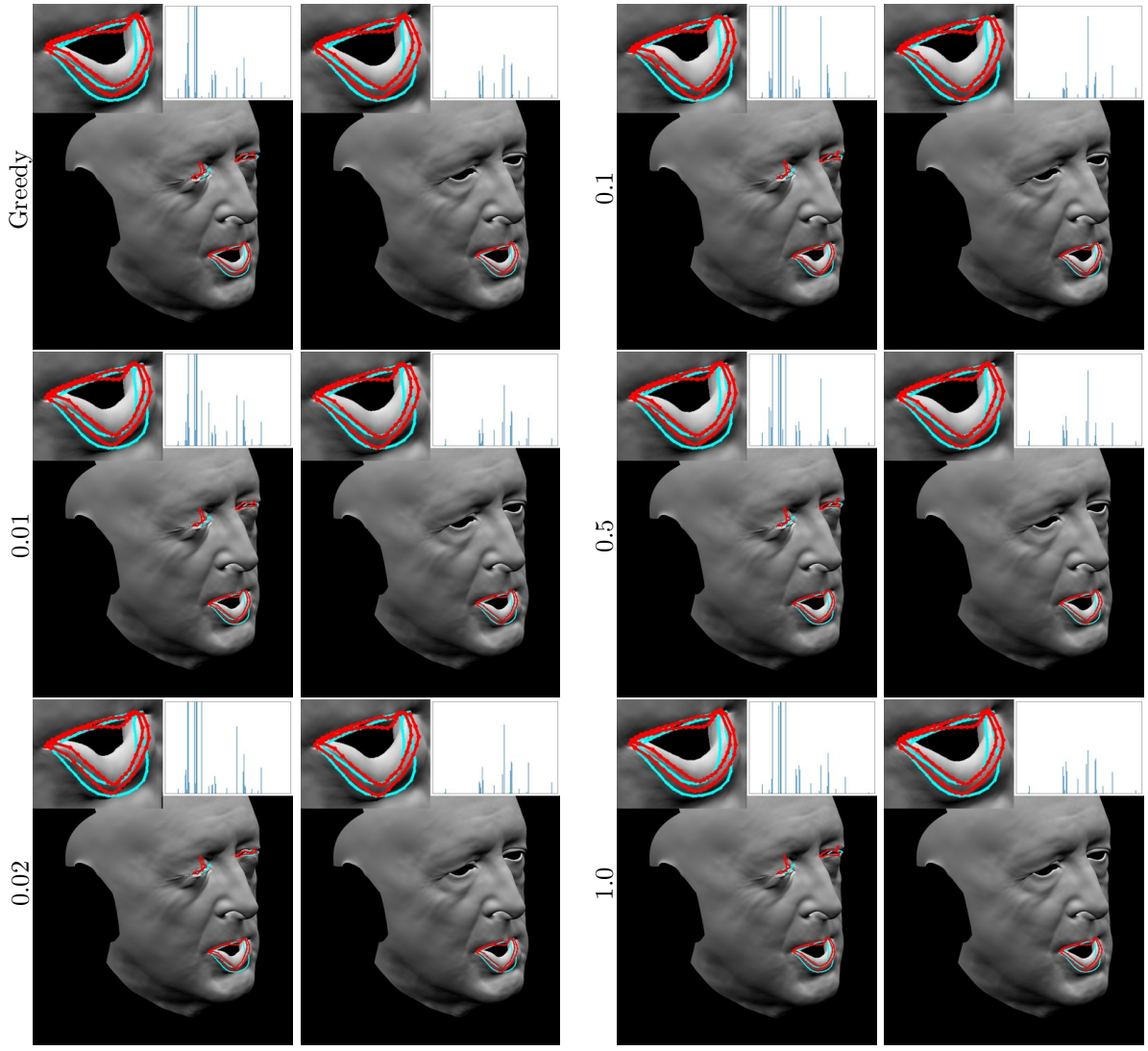
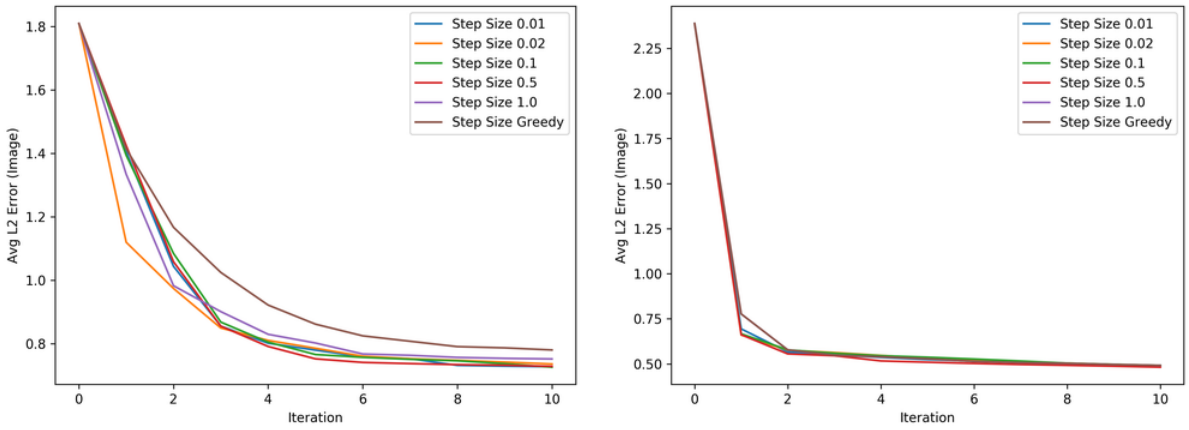
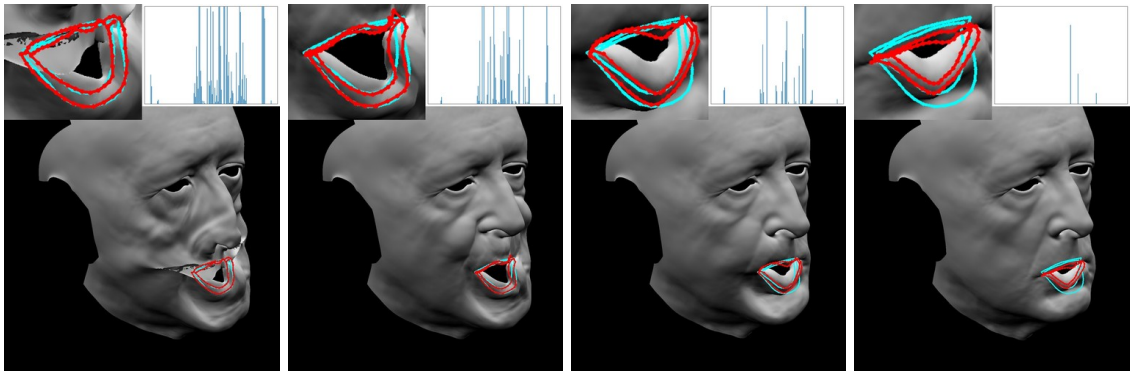


Figure 26: A comparison of the geometry and parameter results from varying the step size used for choosing the next coordinate in CSS.



(a) with eye rotoscope (b) no eye rotoscope

Figure 27: A comparison of the average L_2 errors plotted before every Gauss-Newton iteration when varying the step size used to choose the next coordinate direction in CSS. The brown lines plot the average L_2 errors when using the Gauss-Southwell approach; notice how CSS allows for a faster reduction in error.



(a) No Pruning (b) 0.2 (c) 0.3 (d) 0.5

Figure 28: As we increase the threshold for pruning, the resulting solution becomes sparser and more regularized.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Frédéric Abergel and Roger Temam. On some control problems in fluid mechanics. *Theoretical and Computational Fluid Dynamics*, 1:303–325, 1990.

[3] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[4] Jose M. Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2270–2278. Curran Associates, Inc., 2016.

[5] James Baglama and Lothar Reichel. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM J. Sci. Comput.*, 27(1):19–42, July 2005.

[6] Michael Bao, Jane Wu, Xinwei Yao, and Ronald Fedkiw. Deep energies for estimating three-dimensional facial pose and expression, 2018, 1812.02899.

[7] Michael H. Bao, Matthew D. Cong, Stéphane Grabli, and Ronald Fedkiw. High-quality face capture using anatomical muscles. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10794–10803, 2019.

[8] Michael H. Bao, David Hyde, Xinru Hua, and Ronald Fedkiw. Improved search strategies with application to estimating facial blendshape parameters, 2020, 1812.02897.

[9] Thabo Beeler, Bernd Bickel, Paul Beardsley, Bob Sumner, and Markus Gross. High-quality single-shot capture of facial geometry. *ACM Trans. Graph.*, 29(4), July 2010.

[10] Thabo Beeler, Fabian Hahn, Derek Bradley, Bernd Bickel, Paul Beardsley, Craig Gotsman, Robert W. Sumner, and Markus Gross. High-quality passive facial performance capture using anchor frames. *ACM Trans. Graph.*, 30(4), July 2011.

[11] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[12] Mario Bertero and Patrizia Boccacci. *Introduction to inverse problems in imaging.* CRC press, 1998.

[13] Kiran S. Bhat, Rony Goldenthal, Yuting Ye, Ronald Mallet, and Michael Koperwas. High fidelity facial animation capture and retargeting with contours. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 7–14. ACM, 2013.

[14] Åke Björck. *Numerical methods for least squares problems.* SIAM, 1996.

[15] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques*, pages 187–194. ACM Press/Addison-Wesley Publishing Co., 1999.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016, 1604.07316.

[17] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[18] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2016, 1606.04838.

[19] Léon Bottou, Jonas Peters, Joaquin Quiñonero Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *J. Mach. Learn. Res.*, 14(1):3207–3260, January 2013.

[20] Sofien Bouaziz, Yangang Wang, and Mark Pauly. Online modeling for realtime facial animation. *ACM Transactions on Graphics (TOG)*, 32(4):40, 2013.

[21] Charles G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965.

[22] Charles G. Broyden. Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

[23] Charles G. Broyden. A new double-rank minimisation algorithm. preliminary report. In *Notices of the American Mathematical Society*, volume 16, page 670. Amer. Mathematical Soc., 201 Charles St., Providence, RI 02940-2213, 1969.

[24] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2D 3D face alignment problem? (and a dataset of 230,000 3D facial landmarks). In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1021–1030, 2017.

[25] Chen Cao, Yanlin Weng, Shun Zhou, Yiying Tong, and Kun Zhou. Facewarehouse: A 3D facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):413–425, 2014.

[26] Tony F. Chan and Xue-Cheng Tai. Level set and total variation regularization for elliptic inverse problems with discontinuous coefficients. *Journal of Computational Physics*, 193(1):40–66, 2004.

[27] Pierre Charbonnier, Laure Blanc-Féraud, Gilles Aubert, and Michel Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on image processing*, 6(2):298–311, 1997.

[28] Bilian Chen, Simai He, Zhening Li, and Shuzhong Zhang. Maximum block improvement and polynomial optimization. *SIAM Journal on Optimization*, 22(1):87–107, 2012.

[29] Matthew Cong, Michael Bao, Jane L. E, Kiran S. Bhat, and Ronald Fedkiw. Fully automatic generation of anatomical face simulation models. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 175–183, New York, NY, USA, 2015. Association for Computing Machinery.

[30] Matthew Cong, Lana Lan, and Ronald Fedkiw. Muscle simulation for facial animation in Kong: Skull Island. In *ACM SIGGRAPH 2017 Talks*, SIGGRAPH '17, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Matthew D. Cong, Kiran S. Bhat, and Ronald Fedkiw. Art-directed muscle simulation for high-end facial animation. In *Proceedings of the 15th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA'16)*, pages 119–127, 2016.

[32] Matthew D. Cong, Lana Lan, and Ronald Fedkiw. Local geometric indexing of high resolution data for facial reconstruction from sparse markers. *arXiv preprint arXiv:1903.00119*, 2019.

[33] William C Davidon. Variable metric method for minimization. Technical Report ANL-5990, Argonne National Laboratory, 5 1959.

[34] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[35] Shelley Derksen and Harvey J. Keselman. Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables. *British Journal of Mathematical and Statistical Psychology*, 45(2):265–282, 1992.

[36] Fernando Diaz, Donald Metzler, and Sihem Amer-Yahia. Relevance and ranking in online dating systems. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, page 66–73, New York, NY, USA, 2010. Association for Computing Machinery.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[38] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[39] Heinz Werner Engl, Martin Hanke, and Andreas Neubauer. *Regularization of inverse problems*, volume 375. Springer Science & Business Media, 1996.

[40] Jinyan Fan. The modified Levenberg-Marquardt method for nonlinear equations with cubic convergence. *Mathematics of Computation*, 81(277):447–466, 2012.

[41] Ronald Fedkiw, Yilin Zhu, Winnie Lin, and Jane Wu. Continuous mathematical methods, emphasizing machine learning, 2020. Stanford CS205L Winter 2020 Lecture Slides.

[42] Ronald P. Fedkiw, Guillermo Sapiro, and Chi-Wang Shu. Shock capturing, level sets, and PDE based methods in computer vision and image processing: a review of Osher's contributions. *Journal of Computational Physics*, 185(2):309 – 341, 2003.

[43] Erwin Fehlberg. Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. Technical Report Technical Report 315, NASA, 1969.

[44] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[45] Roger Fletcher. *Practical Methods Of Optimization*. John Wiley & Sons, 1980.

[46] Roger Fletcher and Michael J. D. Powell. A rapidly convergent descent method for minimization. *The computer journal*, 6(2):163–168, 1963.

[47] Jessica G. Gaines and Terry J. Lyons. Variable step size control in the numerical solution of stochastic differential equations. *SIAM Journal on Applied Mathematics*, 57(5):1455–1484, 1997.

[48] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[49] Zhenglin Geng, Daniel Johnson, and Ronald Fedkiw. Coercing machine learning to output physically accurate results. *Journal of Computational Physics*, 406:109099, 2020.

[50] Abhijeet Ghosh, Graham Fyffe, Borom Tunwattanapong, Jay Busch, Xueming Yu, and Paul Debevec. Multiview face capture using polarized spherical gradient illumination. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, New York, NY, USA, 2011. Association for Computing Machinery.

[51] Frederic Gibou, David Hyde, and Ron Fedkiw. Sharp interface approaches and deep learning techniques for multiphase flows. *Journal of Computational Physics*, 380:442–463, 2019.

[52] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[54] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. MorphNet: Fast simple resource-constrained structure learning of deep networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

[55] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 13–20, Madison, WI, USA, 2010. Omnipress.

[56] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems (NIPS)*, 2016.

[57] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, 2015, 1510.00149.

[58] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[59] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer series in statistics New York, NY, USA:, 2001.

[60] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[61] Michael T. Heath. *Scientific computing: an introductory survey*. SIAM, 2002.

[62] Janne Heikkila and Olli Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1106–1112. IEEE, 1997.

[63] Pei-Lun Hsieh, Chongyang Ma, Jihun Yu, and Hao Li. Unconstrained realtime facial performance capture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1675–1683, 2015.

[64] Haoda Huang, Jinxiang Chai, Xin Tong, and Hsiang-Tao Wu. Leveraging motion capture and 3D scanning for high-fidelity facial performance acquisition. In *ACM Transactions on Graphics (TOG)*, volume 30, page 74. ACM, 2011.

[65] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies, 2017, 1702.02284.

[66] Niall Hurley and Scott Rickard. Comparing measures of sparsity. *IEEE Transactions on Information Theory*, 55(10):4723–4741, 2009.

[67] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving, 2015, 1504.01716.

[68] Alexandru Eugen Ichim, Sofien Bouaziz, and Mark Pauly. Dynamic 3d avatar creation from hand-held video input. *ACM Transactions on Graphics (ToG)*, 34(4):45, 2015.

[69] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[70] Antony Jameson, Luigi Martinelli, and Niles A. Pierce. Optimum aerodynamic design using the Navier-Stokes equations. *Theoretical and Computational Fluid Dynamics*, 10:213–237, 1998.

[71] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. Association for Computing Machinery.

[72] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014, 1412.6980.

[73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[74] Igor' Anatol'yevich Krylov and Feliks Leonidovich Chernous'ko. Solution of problems of optimal control by the method of local variations. *USSR Computational Mathematics and Mathematical Physics*, 6(2):12–31, 1966.

[75] Lana Lan, Matthew Cong, and Ronald Fedkiw. Lessons from the evolution of an anatomical facial muscle model. In *Proceedings of the ACM SIGGRAPH Digital Production Symposium*, DigiPro '17, New York, NY, USA, 2017. Association for Computing Machinery.

[76] Jeff Lander. Skin them bones: Game programming for the web generation. *Game Developer Magazine*, 5(1):10–18, 1998.

[77] Rasmus Larsen. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 27(537), Dec. 1998.

[78] Quoc V. Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. 2011.

[79] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann, 1990.

[80] Yann LeCun, Léon Bottou, and Yoshua Bengio. Reading checks with multilayer graph transformer networks. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 151–154 vol.1, 1997.

[81] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[82] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–104 Vol.2, 2004.

[83] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[84] John P. Lewis, Ken Anjyo, Taehyun Rhee, Mengjie Zhang, Fred Pighin, and Zhigang Deng. Practice and Theory of Blendshape Facial Models. In Sylvain Lefebvre and Michela Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014.

[85] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets, 2016, 1608.08710.

[86] Hao Li, Thibaut Weise, and Mark Pauly. Example-based facial rigging. *ACM Transactions on Graphics (TOG)*, 29(4):32, 2010.

[87] Hao Li, Jihun Yu, Yuting Ye, and Chris Bregler. Realtime facial animation with on-the-fly correctives. *ACM Trans. Graph.*, 32(4):42–1, 2013.

[88] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6), December 2018.

[89] Julia Ling, Reese Jones, and Jeremy Templeton. Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, 318:22 – 35, 2016.

[90] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017.

[91] Matthew M. Loper and Michael J. Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014.

[92] Yifei Lou, Xiaoqun Zhang, Stanley Osher, and Andrea Bertozzi. Image recovery via nonlocal operators. *Journal of Scientific Computing*, 42:185–197, 2010.

[93] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3288–3298. Curran Associates, Inc., 2017.

[94] Manolis I. A. Lourakis and Antonis A. Argyros. Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Tenth IEEE International Conference on Computer Vision (ICCV'05)*, volume 2, pages 1526–1531. IEEE, 2005.

[95] Rongrong Ma, Jianyu Miao, Lingfeng Niu, and Peng Zhang. Transformed ℓ_1 regularization for learning sparse deep neural networks. *Neural Networks*, 119:286 – 298, 2019.

[96] Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88*, pages 26–33, CAN, 1989. Canadian Information Processing Society.

[97] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[98] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[99] Brian McFee and Gert Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 775–782, Madison, WI, USA, 2010. Omnipress.

[100] Abdel-Rahman Mohamed, George E. Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[101] Stephen G. Nash. A survey of truncated-Newton methods. *Journal of computational and applied mathematics*, 124(1-2):45–59, 2000.

[102] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured Bayesian pruning via log-normal multiplicative noise. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6775–6784. Curran Associates, Inc., 2017.

[103] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

[104] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[105] Thomas Neumann, Kiran Varanasi, Stephan Wenger, Markus Wacker, Marcus Magnor, and Christian Theobalt. Sparse localized deformation components. *ACM Transactions on Graphics (TOG)*, 32(6):179, 2013.

[106] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[107] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

[108] Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the Gauss-Southwell rule than random selection. In *International Conference on Machine Learning*, pages 1632–1641, 2015.

[109] Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster CNNs with direct sparse convolutions and guided pruning, 2016, 1608.01409.

[110] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[111] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998.

[112] Michael J. D. Powell. A hybrid method for nonlinear equations. In Philip Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114. Gordon and Breach, 1970.

[113] Yinghe Qi, Jiacai Lu, Ruben Scardovelli, Stéphane Zaleski, and Grétar Tryggvason. Computing curvature for volume of fluid methods using machine learning. *Journal of Computational Physics*, 377:155–161, 2019.

[114] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[115] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019.

[116] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 59–66, 1998.

[117] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016, 1609.04747.

[118] Moktar A. Salama, John A. Garba, Laura A. Demsetz, and Firdaus E. Udwadia. Simultaneous optimization of controlled structures. *Computational Mechanics*, 3:275–282, 1988.

[119] David F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[120] Hao-Jun Michael Shi, Shenyinying Tu, Yangyang Xu, and Wotao Yin. A primer on coordinate descent algorithms, 2016, 1610.00040.

[121] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 417–425, New York, NY, USA, 2005. Association for Computing Machinery.

[122] Eftychios Sifakis, Andrew Selle, Avram Robinson-Mosher, and Ronald Fedkiw. Simulating speech with a physics-based facial muscle model. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, page 261–270, Goslar, DEU, 2006. Eurographics Association.

[123] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018.

[124] Danny C. Sorensen. Newton's method with a model trust region modification. *SIAM Journal on Numerical Analysis*, 19(2):409–426, 1982.

[125] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[126] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[127] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[128] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2Face: Real-time face capture and reenactment of RGB videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2387–2395, 2016.

[129] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5: rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural networks for machine learning*, 4(2):26–31, 2012.

[130] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

[131] Marko Vauhkonen, Dénes Vadász, Pasi A. Karjalainen, Erkki Somersalo, and Jari P. Kaipio. Tikhonov regularization and prior information in electrical impedance tomography. *IEEE Transactions on Medical Imaging*, 17(2):285–293, 1998.

[132] Min Wang, Siu Wun Cheung, Wing Tat Leung, Eric T. Chung, Yalchin Efendiev, and Mary Wheeler. Reduced-order deep learning for flow dynamics. the interplay between deep learning and model reduction. *Journal of Computational Physics*, 401:108939, 2020.

[133] Wei Wen, Yiran Chen, Hai Li, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, and Bin Hu. Learning intrinsic sparse structures within long short-term memory. In *ICLR 2018 Conference*, February 2018.

[134] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2074–2082. Curran Associates, Inc., 2016.

[135] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[136] Chenglei Wu, Derek Bradley, Markus Gross, and Thabo Beeler. An anatomically-constrained local deformation model for monocular face capture. *ACM Transactions on Graphics (TOG)*, 35(4):115, 2016.

[137] Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *International Conference on Learning Representations*, 2020.

[138] Jihun Yun, Peng Zheng, Eunho Yang, Aurelie Lozano, and Aleksandr Aravkin. Trimming the ℓ_1 regularizer: Statistical analysis, optimization, and applications to deep learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7242–7251, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[139] Matthew D. Zeiler. ADADELTA: An adaptive learning rate method, 2012, 1212.5701.

[140] Xiaoqun Zhang, Martin Burger, Xavier Bresson, and Stanley Osher. Bregmanized nonlocal regularization for deconvolution and sparse reconstruction. *SIAM Journal on Imaging Sciences*, 3(3):253–276, 2010.

[141] Olgierd Cecil Zienkiewicz and Robert Leroy Taylor. *The finite element method, volume 1: the basis*, volume 1. Butterworth-Heinemann, 2000.

[142] Olgierd Cecil Zienkiewicz and Robert Leroy Taylor. *The finite element method, volume 2: solid mechanics*, volume 2. Butterworth-Heinemann, 2000.

[143] Gaspard Zoss, Derek Bradley, Pascal Bérard, and Thabo Beeler. An empirical rig for jaw animation. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.