Mathematical Formulations with Uncertain Data in Optimization and Inverse Problems

by

Richard J. Clancy

B.A., University of Colorado at Boulder, 2007

M.S., Texas State University, 2015

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2022

Committee Members:

Stephen Becker, Chair

Jem Corcoran

William Kleiber

Emiliano Dall'Anese

Svenja Knappe

Clancy, Richard J. (Ph.D., Applied Mathematics)

Mathematical Formulations with Uncertain Data in Optimization and Inverse Problems

Thesis directed by Prof. Stephen Becker

This dissertation broadly focuses on incorporating uncertainty into mathematical models and can be broken into three distinct sections. In the first (Ch. 2 and 3), we consider design matrix uncertainty for linear regression. The motivation is that many regressors or covariates used to build linear models, such as survey responses and other data subjective in nature, are intrinsically uncertain. We approach the problem from two angles: 1) using robust optimization and 2) an approximation method to construct an otherwise cumbersome objective for maximum likelihood estimation. Expressions for (sub)gradients are provided allowing for the use of efficient solvers. We illustrate the merit of both methods on synthetic data.

In the second section (Ch. 4 and 5), we focus on novel trust region methods that accept uncertainty in objective and gradient evaluations to reduce the computational burden. We introduce an algorithm named TROPHY (**T**rust **R**egion **O**ptimization using a **P**recision **H**ierarch**Y**) which uses variable precision arithmetic to reduce communication, storage, and energy costs. The other project uses a Hermite interpolation based framework to build model objectives using function and derivative information. Since not all partial derivatives are available or are expensive to compute, we make use of uncertain gradients to improve performance beyond standard interpolation methods. We also propose the use of sketching to ameliorate issues with redundant data and reduce the burden of inverting very large matrices.

The final section (Ch. 6) investigates how operator uncertainty impacts the ability to solve inverse problems accurately. Our goal is to localize regions of brain activity using optically pumped magnetometers which are novel sensors that show promise for use in magnetoencephalography. Through a series of simulations, we establish guidelines for sensor count, noise level, and forward model fidelity needed to localize brain activity to a specified accuracy.

Dedication

For Mina – my best friend whose selflessness and good nature allowed me to pursue my academic aspirations.

Acknowledgements

A good friend of mine often complained about her Ph.D. program. I felt it was a shame since I encouraged her to pursue the degree, but also because I've had a distinctly different experience. Outside of the pandemic and periodic bouts with insecurity and stress, I've had a blast. I attribute my experience to the people I've had the pleasure of spending time with over the past five years. First and foremost, I would like to thank Mina for uprooting herself from a comfortable life in Austin allowing me to complete this dissertation. I would like to thank my family: Mom, Dad, Rocky, and Kathy for providing me with a curiosity that made the Ph.D. possible and encouraging me to continue learning in all avenues of life. I'd like to thank my cohort: Daniel, Kesler, Subekshya, Erin, Liam, Lyndsey, and Zofia that made studying for prelims and spending long hours on campus a highlight of my early days in APPM. The APPM Department at large which was always supportive and provided me with everything necessary to succeed. My wonderful neighbors: Ruyu, Felix, Amanda, Shari, Minah, and Caleb who brightened my day whenever I'd run into them and would always lend an ear when I felt like complaining. Osman, Erik, Allen, and Kevin, thanks for always making time for what's important. Nick for many great squash games and walks to see the geese and turtles, they will be sorely missed. I'd like to thank Stephen, Young Ju, Svenja, Orang, and Jeramy for playing instrumental roles in developing me as a researcher. I'd also like to thank my committee: Jem, Will, Svenja, and Emiliano for agreeing to see me through this phase of my academic career. Last but not least, I'd like to thank my good friends Claire, Dane, Joe, and Maura who provided a safe harbor and strong community away from APPM when I needed a break.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Having worked on broad range of problems throughout my Ph.D. including convex optimization, numerical methods, blind source separation, and inverse problems, I struggled to choose a title that would tie them together. A unifying theme touching every project in some form is uncertainty. Typically, uncertainty is an obstacle that must be overcome. One method to handle uncertainty is to act as though it doesn't exist.

As a cautionary tale, consider the simple problem of floating point arithmetic. Unlike real arithmetic, floating point addition isn't associative, i.e., $\text{fl}((a+b)+c) \neq \text{fl}(a+(b+c))$. Although the effects of round-off error are well documented for matrix inversion [Golub and Van Loan, 1996] and catastrophic cancellation for division [Goldberg, 1991], simple addition poses less of a concern.

At Los Alamos National Laboratory, I was charged with the task of characterizing round-off error for large summations. To develop intuition, we looked at the CLAMR simulation that models a cylindrical damn breaking with boundary conditions. The system is propagated in time via the state update equation,

$$U_{i+1} = U_i - \frac{\Delta t}{\Delta r}\left(F^+ - F^- + G^+ - G^-\right) + \left(w_x^+ - w_x^- + w_y^+ - w_y^-\right). \tag{1.1}$$

We considered three common data types: half, single and double precision using 11, 24, and 53 bits for the mantissa, respectively. Half (single) precision has a dynamic range of approximately 4 (7) orders of magnitude. In half precision floating point arithmetic, $1\times10^4 + 1 = 1\times10^4 \neq 1.0001\times10^4$. To understand the frequency of dropped summands, we looked at the difference in exponents (base

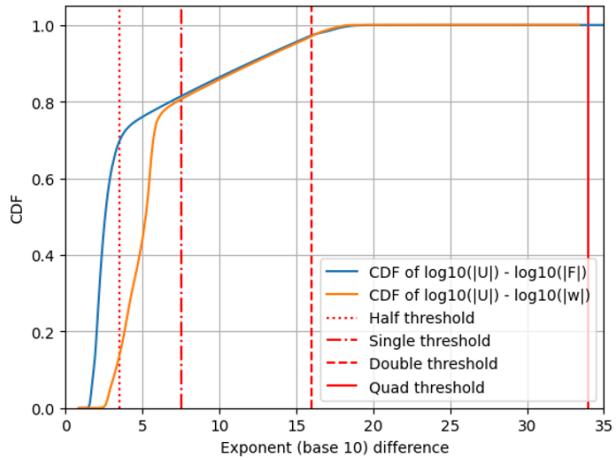

Figure 1.1: CDF for the difference in logarithms of magnitudes (orders of magnitude difference) between terms U_i, F^+, and w_x^+ terms. Vertical lines show threshold over which a term in the sum is lost due to limited dynamic range of floating point arithmetic using half, single, and double precision for Eq. 1.1.

10) between U_i and both the F^+ and w_x^+ terms for each state update. An empirical cumulative density of function (CDF) for the difference in exponents over the course of the simulation is shown in Figure 1.1. We found that due to the limited dynamic range of floating point data types, 3% of all sums between w, F and U terms were completely lost when using double precision, 20% were lost when using single precision, and astoundingly, over 70% of all computations where lost when using half precision! This remarkable result illustrates the significance of imprecision in computational mathematics. It is clear that uncertainty must be accounted for.

A central focus of this dissertation is how to solve linear regression problems when the design matrix is subject to uncertainty. That is, given y and A, find parameters x that best explains observations $y = Ax + z$. We assume that z is an unobserved random variable similar to the classical setting, but also assume that A is either random or only approximately known.

In Chapter 2, which is based on work published in *Signal Processing*[1] , we formulate and solve a robust least squares problem for a system of linear equations subject to quantization error in the data matrix. We are motivated by the fact that ordinary least squares fails to consider

[1] Becker, S. and Clancy, R.J., 2020. Robust least squares for quantized data matrices. *Signal Processing*, 176, p.107711.

uncertainty in the design matrix, modeling all noise in the observed signal. Total least squares accounts for uncertainty in the data matrix, but necessarily increases the condition number of the operator compared to ordinary least squares which is undesirable. Tikhonov regularization or ridge regression is frequently employed to combat ill-conditioning, but requires parameter tuning which presents a host of challenges and places strong assumptions on parameter prior distributions. The proposed method also requires selection of a parameter, but it can be chosen in a natural way, e.g., a matrix rounded to the 4th digit uses an uncertainty bounding parameter of 0.5×10^{-4}. We show that our robust method is theoretically appropriate, tractable, and performs favorably against ordinary and total least squares.

We consider an alternative approach to the regression problem in Chapter 3 drawing on ideas from complex and Fourier analysis. Assuming that the design matrix is a random variable with independent components, we use the saddle point method to construct an approximate probability density, and by extension, log-likelihood function to estimate model parameters. We once again compare it to classical methods with different noise settings. This work is currently posted on arXiv[2] .

The next two chapters focus on using uncertainty in trust region methods to improve algorithmic performance. Chapter 4 investigates the use of partial, incomplete, or infrequently computed gradient information to accelerate convergence for interpolation based models. Although we don't know the gradient at each step making first order methods impossible, we would like to use derivative information to improve our surrogate model when possible. We propose an algorithm and show that it improves the performance of interpolation based methods and allows for model flexibility.

Returning to our cautionary tale, we recognize that despite the loss of accuracy, the use of reduced precision data types can lighten the computational load. We introduce the TROPHY algorithm (Trust Region Optimization using a Precision HierarchY) in Chapter 5 which has been accepted to the *International Conference in Computational Science* and will appear in *Lecture Notes*

[2] Clancy, R.J. and Becker, S., 2021. Approximate maximum likelihood estimators for linear regression with design matrix uncertainty. *arXiv preprint* arXiv:2104.03307.

in Computer Science[3] . The algorithm accepts uncertainty in function and gradient evaluations to reduce communication, memory, and energy costs while optimizing an objective. The algorithm solves the problem to the highest accuracy possible before proceeding to the next precision level. We illustrate the method's merit on the CUTEst test set and a large-scale data assimilation problem to estimate wind fields from radar returns.

In the final chapter, we characterize novel sensors used in magnetoencelphalography. This chapter was published in *Physics in Medicine and Biology*[4] In particular, we formulate the objective used in the inverse problem to localize regions of brain activity, propose an algorithm to solve it, and establish system specifications through simulation studies to localize neural currents within a given tolerance. A central focus of this work was to establish how localization error depends on accurately specifying a forward model. In other words, how does uncertainty in sensor location and orientation translate to inverse problem error?

As can be seen from the above collection, this dissertation spans a diverse set of topics, but all touch on the issue of uncertainty.

[3] Clancy, R.J., Menickelly, M., Hückelheim, J., Hovland, P., Nalluri, P., & Gjini, R. (2022). TROPHY: Trust Region Optimization Using a Precision Hierarchy. *arXiv preprint* arXiv:2202.08387.

[4] Clancy, R.J., Gerginov, V., Alem, O., Becker, S. and Knappe, S., 2021. A study of scalar optically-pumped magnetometers for use in magnetoencephalography without shielding. *Physics in Medicine & Biology*, 66(17), p.175030.

Chapter 2

Robust Least Squares

2.1 Introduction

The primary goal of this paper is to recover unknown parameters from a noisy observation and an uncertain linear operator. In particular, our mathematical model for robust least squares is

$$\min_x \left\{ \max_{\Delta \in \mathcal{U}} \|(A+\Delta)x - b\|^2 \right\} \tag{2.1}$$

which we refer to as our robust optimization (RO) problem. The Euclidean norm is denoted by $\|\cdot\|$ and $\mathcal{U}$ is the uncertainty set from which perturbations in A are drawn.

The above RO formulation is motivated by two situations. In both cases, we let $\bar{A}$ and $\bar{x}$ represent the *true and unknown* data matrix and parameter vector, respectively. We model $b = \bar{A}\bar{x}+\eta$, with η i.i.d. Gaussian, and we only have knowledge of $A = \bar{A}-\Delta$. We don't know Δ explicitly but can make inferences based on the problem. In the first situation, we consider a data matrix subject to quantization or round-off error. Suppose the observed matrix A has elements rounded to the hundredth place. Our uncertainty set can be written as $\mathcal{U} = \{\Delta \in \mathbb{R}^{m\times n} : \|\Delta\|_\infty \leq \delta\}$ with $\delta = 0.005$. If $A_{i,j} = 0.540$, then we know the true $\bar{A}_{i,j} \in (0.535,\, 0.545]$, hence $\Delta_{i,j} \in (-0.005, 0.005]$. The norm $\|\cdot\|_\infty$ takes the maximum absolute value of any element in the matrix.

The second problem considers a data matrix with uncertainty proportional to the magnitude of each entry, i.e. $\mathcal{U} = \{\Delta \in \mathbb{R}^{m\times n} : \Delta_{i,j} \in (-p|A_{i,j}|, p|A_{i,j}|]\}$. Here, p denotes a proportionality constant. Data subject to $\pm 1\%$ uncertainty would have $p = 0.01$. The two cases cover the effects

of finite-precision in fixed and floating point representations, respectively. In both problems, the uncertainty sets are specified element-wise allowing us to decouple along rows.

Due to limitations in both ordinary (OLS) and total least squares (TLS), the signal processing community has sought alternatives when operator uncertainty is present. In [Wiesel et al., 2008] and [Zhu et al., 2014], the authors derive maximum likelihood estimators (MLE) and Cramér Rao bounds for problems with data matrices subject to Gaussian noise under two different models; errors-in-variables and a random variable model. The setup for [Zhu et al., 2014] only has access to sign measurements which is more restrictive than we consider here. Both papers focus primarily on the case where variance in A and b are known; in contrast, we make no distributional assumption on A (it could be stochastic, deterministic, or adversarial). In the absence of knowing the variance, [Wiesel et al., 2008] shows their estimator reduces to the OLS solution. The problem is treated through an approximate message passing framework in [Zhu et al., 2020] for more general, structured perturbations. Our use case fits under this umbrella but they impose a sparsity inducing prior. Although our method performs well with sparse solutions, we do not assume it.

Note that the model in Eq. 2.1 differs from other robust least squares problems considered in [Xu et al., 2010], [Shivaswamy et al., 2006] and classic papers from the late 1990's [El Ghaoui and Lebret, 1997] and early 2000's [Goldfarb and Iyengar, 2003]. Those works usually made special assumptions like the constraint and objective norms match (e.g. minimize $\|x\|_p$ subject to $\|Ax - b\|_p \leq v$), column-wise separability, or ellipsoidal uncertainty sets. The work in [Shivaswamy et al., 2006] is similar regarding row-wise separability, but they impose a hard constraint on the 2-norm of the solution, i.e., $\|x\| \leq k$ for $k \in \mathbb{R}$. For an overview of robust optimization, see [Bertsimas et al., 2011] and [Gorissen et al., 2015] with the latter focusing on implementation.

In this paper, we consider box constraints over entries of the data matrix. Our main contribution is the formulation of a robust objective to handle quantization error in least squares problems and the presentation of methods to solve it. Although the proposed method requires selection of an uncertainty parameter, it is chosen in a natural and theoretically appropriate way based on the observed extent of quantization. This is in contrast to ridge regression and MLE based methods

that require involved parameter tuning or *a priori* knowledge of the probability distributions from which model uncertainty is drawn. We anticipate our method to be most effective under moderate to heavy quantization where fidelity loss is greater than 0.1%.

2.1.1 Limitations of Ordinary and Total Least Squares

The ordinary least squares (OLS) problem seeks a vector x to minimize the residual given by

$$\min_x \|Ax - b\|^2. \tag{2.2}$$

We focus our attention on the over-determined case where $m > n$ and further assume that A is full rank. It is well known that the closed form solution to (2.2) is given by

$$\hat{x}_{\text{OLS}} = (A^T A)^{-1} A^T b. \tag{2.3}$$

A key assumption for OLS is that A is known with certainty and b is subject to additive noise. The OLS solution (2.3) is the MLE for the model $Ax - b = \eta \sim \mathcal{N}(0, \sigma^2 I)$.

In practice, it is uncommon to know A precisely. Typical causes of uncertainty are sampling error, measurement error, human error, modeling error, or rounding error. There were attempts at addressing this model limitation in [Hodge and Moore, 1972] but these relied on small magnitude errors to use a truncated series approximation for an inverse. Total least square (TLS) was developed in response to this lack of symmetry in uncertainty. Rather than isolating noise to the observed signal or right hand side, the TLS model allows for an uncertain operator and is given by $(A + \Delta)\bar{x} = b + \eta$ where A and b are observed with Δ and η a random matrix and vector, respectively. We assume that Δ and η have been scaled to have the same variance; see [Markovsky and Van Huffel, 2007] for a modern overview of the topic. The TLS solutions, $\hat{x}_{\text{TLS}}$, solves

$$\min_{x,\Delta,\eta} \quad \left\|[\Delta, \ \eta]\right\|_F \quad \text{subject to} \quad (A + \Delta)x = b + \eta \tag{2.4}$$

where $[\Delta, \ \eta] \in \mathbb{R}^{m \times (n+1)}$, A and b are the observed matrix/signal pair, and $\|\cdot\|_F$ is the Frobenius norm. It was shown in [H. Golub and F. van Loan, 1980] that (2.4) can be solved via a singular

value decomposition (SVD) with the closed form solution of

$$\hat{x}_{\text{TLS}} = (A^T A - \sigma_{n+1}^2 I)^{-1} A^T b \tag{2.5}$$

where $\sigma_{n+1} \in \mathbb{R}$ being the smallest singular value of the augmented matrix $[A,\ b] \in \mathbb{R}^{m\times(n+1)}$. Similar to OLS, the TLS solution yields a MLE for the model of Gaussian noise in A and b. It should be noted that $(A^T A - \sigma_{n+1}^2 I)$ is necessarily worse conditioned than $A^T A$. Since $A^T A$ is positive definite by virtue of A being full rank, all eigenvalues of $A^T A$ are shifted closer to zero by the amount of σ_{n+1}^2. For a small spectral gap between σ_n and σ_{n+1}, the matrix will be close to singular making solutions extremely sensitive to perturbations in A.

This can be understood intuitively; uncertainty in A permits additional degrees of freedom allowing the model to "fit" noise. To illustrate, consider the simple linear regression problem. We have access to several regressor/response pairs $(a, b) \in \mathbb{R}^2$. Suppose our data is generated from the model $b = a \cdot 0 + \delta$ with $\delta \sim Uniform(\{-1, 1\})$ (the zero function plus discrete and uniform noise). We'd like to recover the true slope parameter, $x = 0$. In this instance, our three samples are given by $(-0.10,\ 1.00)$, $(0.00,\ -1.00)$, and $(0.11,\ 1.00)$. The OLS solution returns a slope of $\hat{x}_{\text{OLS}} \approx 0.45$ whereas TLS gives $\hat{x}_{\text{TLS}} \approx 297.79$. This is shown graphically in Figure 2.1. TLS's ability to vary the operator results in extreme sensitivity to data provided. Although our example appears exceptional, this behavior is typical of TLS. Because of this sensitivity, a number of alternatives have been studied for uncertain operators as mentioned above. A more traditional approach for addressing the ill-conditioning encountered in TLS is ridge regression or Tikhonov regularization which we consider below.

2.1.2 The Trouble with Tikhonov Regularization

Poor conditioning as observed in TLS is often combated with Tikhonov regularization or ridge regression (RR) whereby solutions with large norms are penalized via an ℓ_2 regularization term. The solution to the ridge regression problem solves

$$\min_x \left\{ \|Ax - b\|^2 + \|\lambda x\|^2 \right\} \tag{2.6}$$

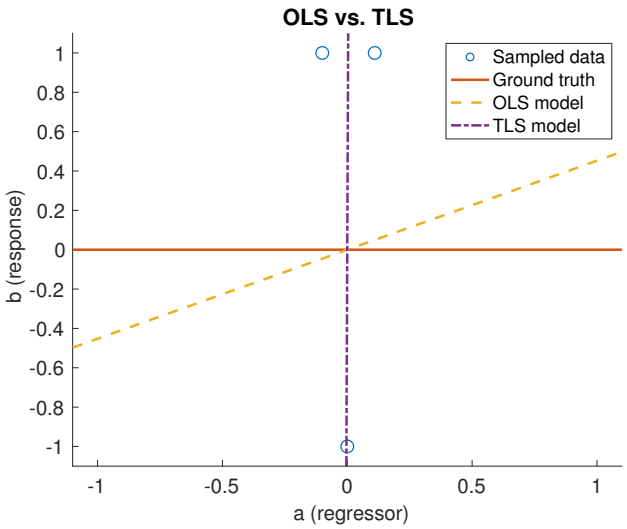

Figure 2.1: Instance of data sampled from a constant function, $\bar{b} = 0$ subject to additive noise. That is, $b = \bar{b} + \delta = \delta$. Solid line indicates ground truth model ($\bar{b} = 0$). Dashed line shows OLS model ($b = a \cdot \hat{x}_{\text{OLS}}$). TLS model is dot-dashed line ($b = a \cdot \hat{x}_{\text{TLS}}$). Since TLS minimizes orthogonal distance, the model over-fits data.

with $\lambda \geq 0$. The minimizing x has a closed form solution of

$$\hat{x}_{\text{RR}_\lambda} = (A^T A + \lambda^2 I)^{-1} A^T b. \tag{2.7}$$

There is a structural connection between the RR and TLS; their solutions are nearly identical with TLS subtracting from the diagonal of $A^T A$ and RR adding to it. This can be understood from a Bayesian statistics point of view. Large values of λ correspond to stronger evidence of lower variance in x (and zero mean). In the case of an uncertain data matrix, we should have *less* confidence in low variance.

Golub, Hansen, and O'Leary explored the interplay between TLS and RR in [Golub et al., 1999]. They considered a generalized version of the regularized total least squares (RRTLS) problem

$$\begin{aligned} &\min_{x,\Delta,\eta} \quad \|[\Delta,\ \eta]\|_F \\ &\text{subject to } (A+\Delta)x = b + \eta \ \text{ and } \ \|x\| \leq \gamma. \end{aligned} \tag{2.8}$$

The second constraint is equivalent to imposing a regularization term on the objective of the TLS problem. When the inequality constraint is replaced by an equality, the solution to (2.8), $\hat{x}_{\text{RRTLS}_\alpha}$

say, is given by the x that solves

$$x = \left(A^T A + \alpha I\right)^{-1} A^T b \quad \text{with} \quad \alpha = \mu(1+\gamma^2) - \frac{\|b - Ax\|^2}{1+\gamma^2} \tag{2.9}$$

and μ the corresponding Lagrange multiplier. Note that for all $\gamma > 0$, we have $\alpha \geq -\sigma_{n+1}^2$. Similarly, $\alpha = -\sigma_{n+1}^2$ when $\gamma = \|\hat{x}_{\text{TLS}}\|$ and $\alpha = 0$ when $\gamma = \|\hat{x}_{\text{OLS}}\|$. This suggests that regularized total least squares merely adds a standard regularization term to the poorly conditioned TLS matrix, $A^T A - \sigma_{n+1}^2 I$.

Given the interplay between TLS, RR, and RRTLS, it is reasonable to solve either TLS or RR since they are effectively the same problem with different parameters; the additional term becomes regularizing when $\alpha = \lambda^2 > 0$. Although TLS is most appropriate for the case of an uncertain matrix, its "deregularizing" effect inflames issues with conditioning making it an unpopular choice for solving typical linear inverse problems.

There is also the tricky consideration of choosing an appropriate regularization parameter, whether in the standard RR form of (2.6) or γ in (2.8). Although there are many approaches for choosing a parameter such as Morozov's discrepancy principle (MDP), the unbiased predictive risk estimator method (UPR), the generalized cross validation method (GCV), or the "elbow" or "L" method to name a few, parameter selection is based on heuristic arguments or requires unknown information *a priori*. A detailed treatment of the above methods and their analysis can be found in [Vogel, 2002]. Through the remainder of this paper, we drop our discussion of RRTLS, focusing instead on TLS and RR.

2.1.3 Robust Least Squares

The central focus of robust optimization (RO) is to find a solution that is feasible over all possible realizations of uncertain variables. In our case, $\Delta \in \mathcal{U}$ is unknown where $\mathcal{U} = \{M \in \mathbb{R}^{m \times n} : |M| \leq D\}$ and D is an element-wise positive matrix chosen in a principled fashion, i.e., the degree to which matrix entries are quantized. Our robustified version is written as a minimax optimization problem in (2.1) with the appropriate $\mathcal{U}$. The inner maximization problem guards

against over-fitting, effectively regularizing our solution, and reducing sensitivity to perturbations in A. Note that RO avoids placing a statistical prior on Δ, which can be a strength or weakness depending on the model.

The outline of the rest of the paper is as follows: in section II we derive a closed form solution to the inner maximization problem thereby showing its tractability, section III discusses computational methods for solving problem (2.1), and section IV provides the results to numerical experiments.

2.2 Closed Form Solution of Inner Objective and Theory

2.2.1 Floating Point Uncertainty

We begin by breaking (2.1) into an inner maximization and outer minimization problem:

$$\min_x \underbrace{\left\{ \max_{|\Delta| \leq D} \|(A+\Delta)x - b\|^2 \right\}}_{f(x)} \tag{2.10}$$

which is equivalent to

$$\min_x f(x) \quad \text{subject to} \quad f(x) = \max_{|\Delta| \leq D} \big\|(A+\Delta)x - b\big\|^2. \tag{2.11}$$

Here $|\Delta| \leq D$ indicates that the magnitude of elements in Δ are bound by the corresponding non-negative components of D. That is, $D_{i,j}$ constrains element $\Delta_{i,j}$ of the uncertainty matrix. The inner maximization problem will recover $f(x)$. Because the function $x \mapsto \|(A+\Delta)x - b\|^2$ is convex in x and the supremum over an arbitrary family of convex functions is convex, it follows that f is convex making $\min_x f(x)$ an unconstrained convex optimization problem. To evaluate f and find a subgradient, it remains to find the maximizing $\Delta \in \mathbb{R}^{m \times n}$.

Theorem 1. *The maximizing function $f(x)$ in (2.10) is given by*

$$f(x) = \big\|Ax - b\big\|^2 + 2\langle\, |Ax-b|, D|x|\,\rangle + \big\|\, D|x|\,\big\|^2 \tag{2.12}$$

where $|x|$ and $|Ax-b|$ denote the vectors of component-wise absolute values.

Proof. Since $f(x) = \max_{|\Delta| \le D} \left\|(A+\Delta)x - b\right\|^2$ must be maximized over Δ, we can fix x, define $c := Ax - b$, then treat it as constant. Exploiting row-wise separability, we write

$$\max_{|\Delta| \le D} \|\Delta x + c\|^2 = \sum_{i=1}^{m} \max_{|\Delta_i^T| \le D_i^T} (\Delta_i^T x + c_i)^2 \tag{2.13}$$

where Δ_i^T and D_i^T are the i^{th} row of Δ and D, respectively, and c_i is the i^{th} element of vector c. We now work row by row. Note that we can switch to absolute values rather than squares when maximizing for each row. Applying the triangle inequality gives an upper bound

$$\begin{aligned} |\Delta_i^T x + c_i| \;&\le |c_i| + \sum_{j=1}^{n} |\Delta_{i,j}|\,|x_j| \\ &\le |c_i| + \sum_{j=1}^{n} D_{i,j}|x_j| \quad = \quad D_i^T|x| + |c_i|. \end{aligned} \tag{2.14}$$

It is easily verified that the upper bound is achieved when

$$\Delta = D \odot \operatorname{sign}(x\, c^T) \tag{2.15}$$

making it a solution to the inner maximization problem. Here, $\odot$ denotes the Hadamard or element-wise product. Recalling that $c_i = [Ax - b]_i$, we simplify to

$$f(x) = \left\|Ax - b\right\|^2 + 2\langle\, |Ax - b|, D|x|\,\rangle + \left\|\, D|x|\,\right\|^2. \tag{2.16}$$

□

Using Theorem 1, the optimization problem in (2.10) can be rewritten as

$$\min_x f(x) = \min_x \left\{ \left\|Ax - b\right\|^2 + 2\langle\, |Ax - b|, D|x|\,\rangle + \left\|\, D|x|\,\right\|^2 \right\}. \tag{2.17}$$

In general, f is not differentiable $\big($e.g., let $m = n = 1$ and $A = D = b = 1$, then $f(x) = (x-1)^2 + 2|x| \cdot |x-1| + x^2$ is not differentiable at $x \in \{0, 1\}\big)$, but we are guaranteed a subgradient by virtue of convexity. Furthermore, a generalization of Danskin's Theorem [Bertsekas, 1971] provides us with a method to find elements of the subgradient for all x. In particular,

$$f'(x) := 2(A + \Delta_x)^T \left[(A + \Delta_x)x - b\right] \;\in\; \partial f(x) \tag{2.18}$$

where Δ_x indicates the optimal Δ for a given x as provided in Eq. (2.15).

2.2.2 Fixed Point Uncertainty

Fixed point uncertainty is a special case of (2.10) with all elements bound by the same value. We can write this constraint as $\|\Delta\|_\infty \leq \delta$ with $\|\Delta\|_\infty$ representing the largest magnitude element of Δ. Problem (2.10) becomes

$$\min_x \left\{ \max_{\|\Delta\|_\infty \leq \delta} \|(A+\Delta)x - b\|^2 \right\} \tag{2.19}$$

with a solution denoted by $\hat{x}_{\text{RO}}$. We focus on fixed point error for the remainder of the paper. This instance gives rise to the following corollary.

Corollary 1. *For fixed point uncertainty, problem* (2.19) *reduces to*

$$\min_x \left\{ \|Ax-b\|^2 + 2\delta\|x\|_1\|Ax-b\|_1 + m\delta^2\|x\|_1^2 \right\}. \tag{2.20}$$

Proof. We take $\mathbf{1}_k \in \mathbb{R}^k$ to be the vector composed of ones. Note that $\|\Delta\|_\infty \leq \delta$ is equivalent to $|\Delta| \leq D$ when $D = \delta\, \mathbf{1}_m \mathbf{1}_n^T$. By Theorem 1, the result follows easily using properties of inner products. □

2.2.3 Explicit Regularization of Robust Objective

The robust formulation presented above is intended to address uncertainty in the data matrix from rounding error. Although implicit regularization occurs via the 3rd term in both (2.17) and (2.20), we don't address poor conditioning of A directly. To illustrate this point, note that $\hat{x}_{\text{RO}} \to \hat{x}_{\text{OLS}}$ as $\delta \to 0$. If A is poorly conditioned, such behavior is undesirable. This can be remedied by adding an ℓ_2 regularization term and solving

$$\min_x \left\{ \left(\|Ax-b\|^2 + 2\delta\|x\|_1\|Ax-b\|_1 + m\delta^2\|x\|_1^2\right) + \lambda^2\|x\|^2 \right\}. \tag{2.21}$$

Since the robust objective and regularization term are both convex, so is their sum. Furthermore, the modified objective is $2\lambda^2$ strongly convex yielding a unique minimizer, $\hat{x}_{\text{RRO}_\lambda}$. Strong convexity also implies a bounded sequence of iterates, i.e., $\|x_k - x^*\| \leq M$, and therefore guarantees convergence under the mirror-descent method. In fact, the same technique works with regularizers other than ℓ_2, and can be solved via proximal mirror-descent methods when the regularizer is smooth or has an easy-to-compute proximity operator [Beck, 2017, Ch. 9].

2.3 Algorithms

Equipped with an element of the subdifferential, we can employ a variety of solvers. Bundle methods are a promising choice as they sequentially form an approximation to the objective. This is done by using the location, function value, and subgradient of previous iterates to lower bound the objective with supporting hyperplanes. At each step, a direction finding quadratic program must be solved, but can be dealt with rapidly thanks to software such as CVXGEN [Mattingley and Boyd, 2012] and ECOS [Domahidi et al., 2013]. A survey on the topic can be found in [Mäkelä, 2002].

Since the problem is unconstrained, another option is to use smooth optimization techniques, and in particular quasi-Newton methods that form low-rank approximations of $\nabla^2 f$. Our objective f is not differentiable, much less twice so, and consequently methods requiring second derivatives seem theoretically inappropriate. However, there is a growing body of literature going back to Lemaréchal [Lemarechal, 1982] recognizing the empirical success of these methods. See [Lewis and Overton, 2013, Guo and Lewis, 2018] and references therein. The success of these methods depends on the smoothness near the solution. Using the `minFunc` MATLAB package [Schmidt, 2005] with random and zero vector initializations, and using the package's default solver of limited-memory BFGS [Nocedal, 1980], we observed fast and accurate convergence for several test problems. We used this method for the numerical experiments presented in the next section.

Subgradient descent is an appealing option due to its simplicity and flexibility. The method can easily handle constraints via projections or a regularization term $h(x)$ with proximal operators. Recent advances in proximal subgradient descent methods and their analysis can be found in [Cruz, 2017] and [Millán and Machado, 2019]. Convergence may be slow, but is less of a concern with heavy quantization. The 3rd terms in (2.17) and (2.20) effectively regularize the objective when δ and $\|D\|$ are large yielding faster convergence. A drawback of the subgradient descent method is that a step-size scheme must be chosen in advance. One popular choice is the diminishing, non-summable step length given by $t_k = \frac{1}{\sqrt{k+1}\cdot\|f'(x_k)\|}$. A discussion on the choice of step lengths can

be found in [Boyd et al., 2003]. Stopping criteria are also difficult to specify since it is not a true descent method (the objective is not guaranteed to decrease on every iteration). The algorithm therefore typically runs for a fixed number of iterations.

2.4 Numerical Experiments

Setup: We use MATLAB's pseudo-random number generator `randn` to draw 10,000 standard normal matrices, $\bar{A} \in \mathbb{R}^{30\times 15}$. The bar notation in this section indicates true and unobserved values. We fixed the condition number of our matrices at 100. We did so by performing a singular value decomposition (SVD) such that $\bar{A} = U\Sigma V^T$ where U, V are unitary and Σ is diagonal, then replaced the diagonal of Σ by linearly decaying values from $1 \to \frac{1}{100}$. This ensures our test matrices show sensitivity to noisy measurements prior to quantization.

We conducted two experiments. In the first, we draw random vectors from a heavy-tailed Cauchy distribution with median 0 and scale parameter 1. In the second, the true solution $\bar{x}$ has its first element drawn from $\bar{x}_1 = \pm 100$ with a sign drawn uniformly at random and the remaining elements drawn from a standard normal distribution, $\bar{x}_{2:15} \sim \mathcal{N}(0, I)$. This setup exposes the bias of RR solutions. We remark that we do this for illustrative purposes; if one suspected a signal of having such a large element, it would of course make sense to run an outlier detection method first.

We obtain our "true" right hand side by taking the image of $\bar{x}$ under $\bar{A}$, that is, $\bar{b} := \bar{A}\bar{x}$, then generate our observed measurement vector by letting $b := \bar{b} + \eta$ where $\eta \sim \mathcal{N}\left(0, \frac{\|\bar{b}\|^2}{m \cdot \mathrm{SNR}} I\right)$ with the desired signal-to-noise ratio (SNR) fixed at $\mathrm{SNR} = 50$. Finally, we quantize $\bar{A}$ which yields our observed A, then solve problems for OLS (2.2), TLS (2.4), RR_λ (2.6), RO (2.19), and RRO_λ (2.21). The λ for RR_λ and RRO_λ are chosen according to the GCV and MDP criteria discussed in Section 2.1.2, and the δ for RO and RRO_λ is based on the observed accuracy, e.g., if $\bar{A}$ is rounded to the 2^{nd} decimal, then $\delta = 0.5 \cdot 10^{-2}$. We use the `minfunc` implementation of limited-memory MFGS to solve the robust problems with a random initialization. The GCV parameters is recovered using an approximate line search for the corresponding objectives provided in Vogel's text [Vogel, 2002]. We omit UPR because of its similarity to the GCV parameter.

The MDP parameter is found using the bisection method to solve $\|A\,\hat{x}_{\mathrm{RR}_\lambda} - b\|^2 - \rho = 0$ for λ, where $\hat{x}_{\mathrm{RR}_\lambda}$ is the solution to (2.6) and the parameter ρ ideally reflects the residual's noise floor. We chose this method since it is computationally cheap and easy to program, though it is also possible to solve a constrained least-squares problem directly using standard software like `cvxopt` [Dahl and Vandenberghe, 2010]. The goal is to find λ such that $\|A\hat{x}_{\mathrm{RR}_\lambda} - b\|^2$ equals the expected uncertainty introduced by noise. For $\bar{A}\bar{x} - b = \eta \sim \mathcal{N}(0, \sigma^2 I)$ which is an m component Gaussian random variable,

$$\frac{1}{\sigma^2}\|\eta\|^2 = \sum_{i=1}^{m} \left(\frac{\eta_i}{\sigma}\right)^2 \sim \chi^2(m) \tag{2.22}$$

since $\frac{\eta_i}{\sigma} \sim \mathcal{N}(0,1)$ and $\left(\frac{\eta_i}{\sigma}\right)^2 \sim \chi^2(1)$. When $m = 30$, as in our case, both the mean and median are approximately 30 (exact for mean). To ensure feasibility, i.e., a real root, we choose ρ such that $\mathbb{P}\left(\|\bar{A}\bar{x} - b\|^2 - \rho \leq 0\right) = 95\%$, which can be easily calculated using χ^2 inverse CDF tables. For our experiments, $\rho \approx \frac{2\|\bar{b}\|^2}{3 \cdot \mathrm{SNR}}$. An appropriate ρ is generally unknown *a priori* unless one knows the noise variance.

Numerical Results: Results for the Cauchy distributed measurement vector are displayed in Figure 2.2, which shows mean error as a function of rounding digit for $\bar{A}$, and Figure 2.3, which illustrates an empirical probability density function (pdf) of component-wise errors for different methods using a standard Gaussian kernel. In this case, A is quantized to the hundredth spot. Note that the robust solution does well at reducing error in the face of heavy quantization. This is evidenced by low mean error through the thousandth spot and small variance as seen in the pdf.

The RO solution converges to the OLS solution as expected when $\delta \to 0$, i.e., when the quantization effect is small. This might be undesirable, especially when A is ill-conditioned, and can be addressed via the inclusion of a regularization term. The mean relative error performance for the regularized robust problem given by (2.21) is shown in Figure 2.4. Since the proposed method does not improve estimation universally, we recommend its use when quantization results in precision loss of greater than 0.1%. We reiterate that when quantization effects are heavy, there

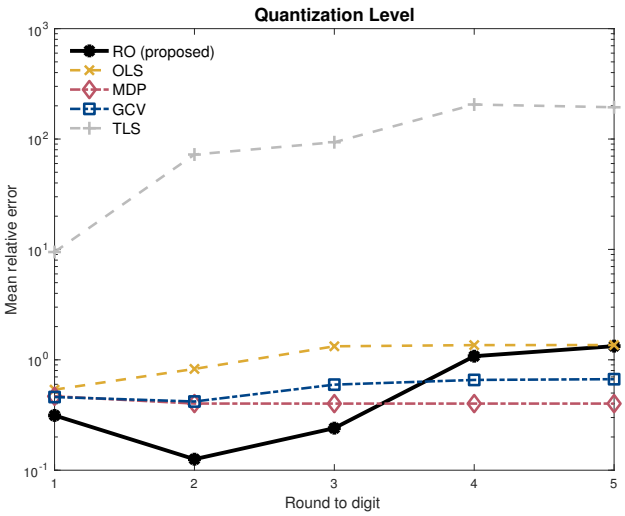

Figure 2.2: Mean relative error $\|e\| = \|\hat{x} - \bar{x}\|/\|\bar{x}\|$ over 10k simulations as a function of the digit to which matrix entries were rounded to. Note that $\delta = 0.5 \times 10^{-\text{Round to digit}}$.

is a notable benefit of the proposed method and the parameter δ can be chosen in a natural way.

A second observation is that RO doesn't sacrifice accuracy for bias as RR does. This phenomenon can be observed in Figure 2.5 which depicts empirical pdf's of component-wise error for different methods. The plots to the right show error behavior for small components of the solution, that is, for $\bar{x}_{2:15} \sim \mathcal{N}(0, I)$. Since values are close to zero, RR is expected to perform well at estimating the true solution. Indeed, RR implicitly assumes a prior with mean zero. The densities on the left show errors for "large" components drawn from $\{\pm 100\}$ (and the sign of the error is adjusted by the sign of the large component, so a negative error indicates that the estimate is biased toward zero, and a positive error indicates bias to $\pm\infty$). Rather than localizing about zero, the absolute error's mode is observed around -7 for GCV and UPR and -25 for MDP. Heavier penalty terms (bigger λ) place less weight on the LS term in the objective and bias estimates more aggressively towards zero. Typical λ values for GCV, UPR, and MDP are 0.041, 0.042, and 0.17, respectively. The robust solution accurately estimates the large and small components values without the extreme errors observed with OLS and TLS; that is, RO appears to have a low-variance, in-line with RR, but also low-bias, comparable or better than OLS.

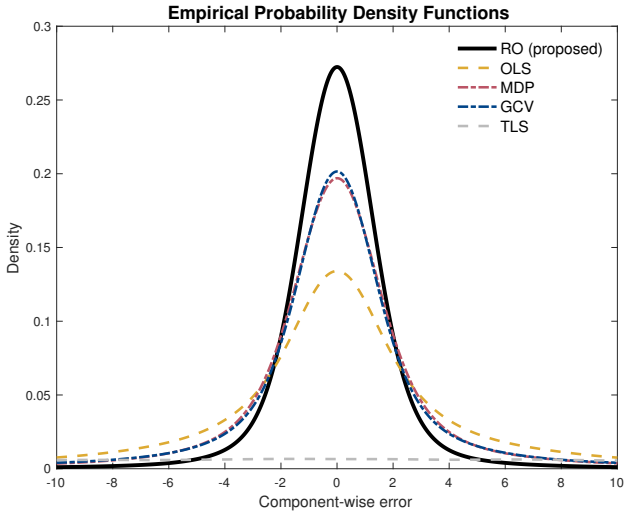

Figure 2.3: Empirical probability density function of component-wise error over 10k simulation when $\bar{x}$ is drawn from a Cauchy distribution and $\bar{A}$ is quantized to the hundredth spot (round to digit $= 2$ and $\delta = 0.5 \times 10^{-2}$).

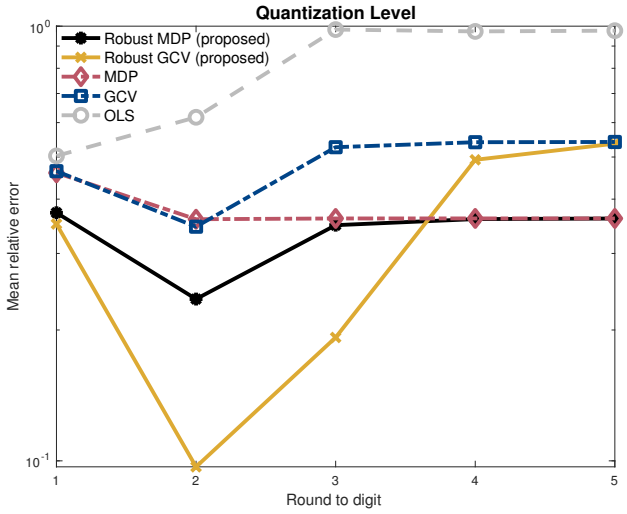

Figure 2.4: Mean relative error $\|e\| = \|\hat{x} - \bar{x}\|/\|\bar{x}\|$ over 10k simulations of RR_λ and its robust counter part, RRO_λ. Note that $\delta = 0.5 \times 10^{-\text{Round to digit}}$. The robust counterpart shows benefit for large quantization then converges to corresponding RR solution.

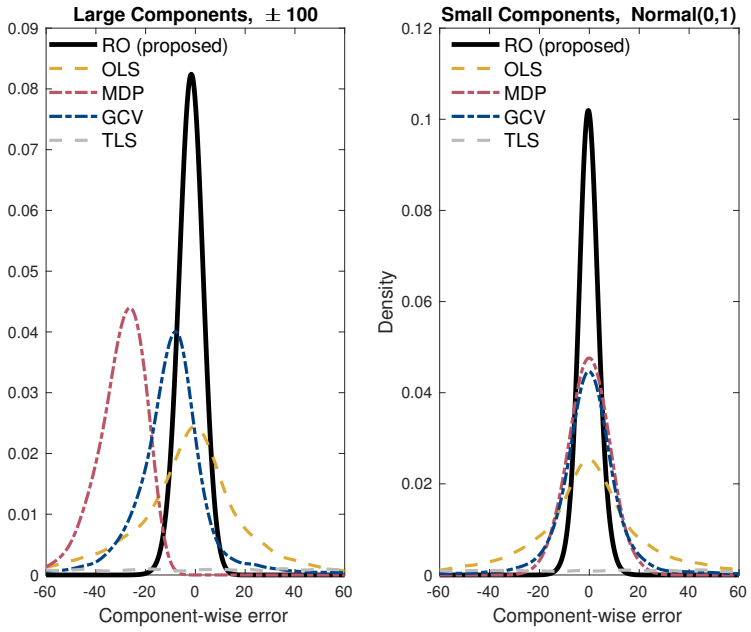

Figure 2.5: Empirical probability densities of component-wise errors for the single large element experiment when $\bar{A}$ is quantized to the hundredth. Left: Pdf of the error (adjusted by the sign of the large element $\bar{x}_1$) of different methods for the large element ($\bar{x}_1 = \pm 100$). Note that RR and GCV have have modes away from zero indicating bias. Right: Pdf of the error for the 2^{nd}–15^{th} components (which are drawn from $\mathcal{N}(0, 1)$).

2.5 Conclusion

In this paper, we presented a robust method for addressing fixed and floating point uncertainty due to quantization error. After reviewing the limitations of existing methods, we formulated a tractable robust objective and presented algorithms to solve it. The only parameter necessary in our formulation is chosen in a principled fashion, i.e., by observing the degree to which matrix elements are rounded. Our numerical experiments show that robust least squares outperforms OLS, TLS, and RR, effectively balancing bias and accuracy.

Acknowledgments: We would like to thank Marek Petrik for his keen eye and helping us spot an elegant simplification to our approach.

Chapter 3

Approximate Maximum Likelihood Estimators for Regression with Data Uncertainty

3.1 Introduction

We consider the linear regression problem subject to uncertainty in the operator or data/design matrix given by the generative model

$$\mathbf{y} = \mathbf{G}\mathbf{x} + \boldsymbol{\eta}, \tag{3.1}$$

where $\mathbf{G} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{\eta} \in \mathbb{R}^m$ are random variables with independent components, and $\mathbf{x} \in \mathbb{R}^n$ is a fixed but unknown parameter vector. Typical causes of operator uncertainty are sampling error, measurement error, human error, modeling error, or rounding error. Our focus is on the over-determined case when $m > n$. We assume distributional knowledge of both $\mathbf{G}$ and $\boldsymbol{\eta}$. Our goal is to recover an estimate for $\mathbf{x}$ given observations of $\mathbf{y}$ using a maximum likelihood estimation (MLE) framework.

We have several motivating examples in mind. The first involves quantization error where design matrix elements are rounded to a fixed decimal place. Such instances naturally arise during digitization and can be modeled as Berkson error. Another typical case is for surveys or ratings where respondents answer questions on a Likert scale to estimate a response variable, e.g., suitability of an applicant for a particular job. A continuum of preferences are forced to take integer values. Furthermore, survey responses are subjective in nature introducing more uncertainty.

A second example involves effectively estimating $\mathbf{x}$ when the design matrix is subject to

floating point error. This might be encountered when data is roughly transcribed or the number of significant figures (digits in the mantissa) stored on a drive are limited for memory savings. Floating point error is a generalization of rounding error.

The third example is for clipping, i.e., we observe $\mathbf{H} = \text{sign}(\mathbf{G}) \cdot \max(\mathbf{G}, \gamma)$ where $\max(\cdot, \gamma)$ operates element-wise and γ is a clipping threshold. If $\mathbf{G}$ is drawn from a double exponential, then the uncertainty in elements of $\mathbf{H}$ taking extreme values of $\pm\gamma$ will be distributed exponentially (up to a sign). Given the heavy tails of exponentials, it is desirable to incorporate operator uncertainty in the problem formulation.

Our decision to focus on MLEs is supported by the fact that well known regression formulations are in actuality MLEs for additive noise, that is, for $\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\eta}$ where $\boldsymbol{\eta}$ is random but $\mathbf{A}$ is known. In particular, $\text{argmin}_x \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$ (ordinary least squares), $\text{argmin}_x \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_1$ (least deviation regression), and $\text{argmin}_x \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_\infty$ (minimax regression) correspond to the MLEs for Gaussian, double exponential, and uniform noise, respectively.

These MLE regression problems rely on knowledge of the vector $\mathbf{y}$'s joint probability density function (PDF). Note that each component of $\mathbf{y}$ in (3.1) is the sum of scaled random variables, i.e., $y_i = \mathbf{g}_i^T\mathbf{x} + \eta_i = \sum_{i=1}^n G_{ij}x_j + \eta_i$ where $\mathbf{g}_i^T$ is the i^{th} row of $\mathbf{G}$ and subscripts denote the component of the corresponding vector. Despite the innocuous form, sums of random variables are difficult to work with: individual PDFs must be convolved to obtain a PDF for their sum.

One option is to ignore uncertainty in $\mathbf{G}$ altogether, focusing on additive noise in $\mathbf{y}$ alone as done in ordinary least squares (OLS). Although reasonable when uncertainty in the design matrix is small, this simplification often fails in practice. Total least squares (TLS) was developed to resolve the asymmetry in uncertainty between the design matrix and the measurement vector. It is often used with the errors-in-variables model (EIV). TLS solves the problem

$$\min_{\mathbf{x},\boldsymbol{\eta},\Delta} \left\| [\Delta, \quad \boldsymbol{\eta}] \right\|_F \text{ subject to } (\mathbf{A} + \Delta)\mathbf{x} = \mathbf{y} + \boldsymbol{\eta} \tag{3.2}$$

where $[\Delta, \boldsymbol{\eta}] \in \mathbb{R}^{m\times(n+1)}$ is minimized with respect to the Frobenius norm. The model supposes that for an observed $\mathbf{A}$, there is true deterministic $\mathbf{A}_{\text{TRUE}} = \mathbf{A} + \Delta^*$ where Δ^* solves problem

(3.2). Golub and Van Loan showed that TLS can be solved via a singular value decomposition with a closed form solution $\hat{\mathbf{x}}_{\text{TLS}} = (\mathbf{A}^T\mathbf{A} - \sigma_{n+1}^2\mathbf{I})^{-1}\mathbf{A}^T\mathbf{y}$ [H. Golub and F. van Loan, 1980] with σ_{n+1} being the smallest singular value of $[\mathbf{A}, \mathbf{y}]$. The TLS solution coincides with the MLE for a deterministic $\mathbf{A}_{\text{TRUE}}$ with independent, identically, distributed additive Gaussian noise [Van Huffel and Vandewalle, 1991, Fuller, 1987]. An analytic solution is appealing but it suffers from a deterioration in conditioning.

As an alternative to TLS, Wiesel, Eldar, and Yeredor [Wiesel et al., 2008] devised a MLE when the design matrix is a random variable with all uncertainty normally distributed. We use the same generative model in our setup, but allow for noise from general distributions. By exploiting properties of Gaussian distributions, they formed a likelihood function (LF), showed its equivalence to a (de)regularized least squares problem, and provided algorithms to find the estimator. Gaussiantity is central to their analysis, simplifying otherwise intractable calculations.

Efforts have been made to move away from Gaussian noise through robust optimization where the goal is to generate estimates impervious to perturbations in the observed data. Many robust optimization problems are cast in a minimax form [Xu et al., 2010, Goldfarb and Iyengar, 2003, El Ghaoui and Lebret, 1997, Bertsimas et al., 2011, Becker and Clancy, 2020]. Typically, an uncertainty set $\mathcal{U}$ and objective function f are specified, then the aim is to solve $\min_x \{\max_{U\in\mathcal{U}} f(x, U)\}$. There are a number of drawbacks to the robust framework; in particular, estimates tend to be overly conservative. Furthermore, these methods discard distributional knowledge of the noise focusing instead on set geometry.

To make progress on the MLE in general, we require a method to efficiently construct a probability density. Although this is a difficult task for all but a few special distributions, there is hope. Given certain regularity conditions, work in the Laplace domain is possible through a bilateral transformation. Rather than working with PDFs directly, we can use their moment generating functions (MGF). The ideas in this paper rely on two important properties of MGFs:

(1) the MGF for a sum of independent random variables is the product of their individual

MGFs, i.e., $M_{X+Y}(t) = M_X(t)M_Y(t)$ and

(2) the MGF uniquely characterizes a random variable as its PDF does.

Not all random variables permit an MGF (e.g., the Cauchy distribution lacks one) but many distributions of practical interest do, including Bernoulli, binomial, Poisson, uniform, Gaussian, exponential distributions, as well as all distributions with bounded support. It is assumed throughout this paper that all random variables discussed have MGFs.

Using moment generating functions, we can easily specify the distribution for linear combinations of random variables found in the regression problem. Recovery of a PDF for use in a LF is possible, in theory, through inversion of the MGF but is often difficult in practice. Instead, by using the MGF, we can employ the saddle point method to estimate the PDF and form an approximate LF. We then maximize the approximate LF to recover an estimate of $\mathbf{x}$ in Eq. 3.1 accounting for uncertainty in matrix $\mathbf{G}$.

The saddle point method is a generalization of Laplace's method and was first used by Debye to study Bessel functions of high order [Debye, 1909] then by Watson for statistical mechanics [Watson, 1995]. It was extended by Daniels in his seminal paper [Daniels, 1954] to estimate the PDF of sample means. Barndorff-Nielsen and Cox [Barndorff-Nielsen and Cox, 1979] and Luggannani and Rice [Lugannani and Rice, 1980] generalized his work for independent sums of random variables. Spady [Spady, 1991] discussed saddle point approximations for linear regression problems when additive noise is independent but not identically distributed and focused on the distribution of user specified estimating equations such as $\nabla_{\mathbf{x}}\|\mathbf{Gx} - \mathbf{y}\|^2$ or the subdifferential of $\|\mathbf{Gx} - \mathbf{y}\|_1$. Stawdermann, Casella, and Wells [Strawderman et al., 1996] used the saddle point method to approximate distributions of MLEs in the regression problem, but focused on its statistical properties rather than point estimation. A number of other works use Laplace's approximation for parameter estimation in general linear models but most applications are concerned with longitudinal scientific studies and assume Gaussianity throughout [Vonesh, 1996, Ko and Davidian, 2000, Battauz, 2011].

In this paper, we employ the ideas established in [Strawderman et al., 1996] and derive an

approximate likelihood function for point estimation in linear regression problems with uncertainty in both the measurement vector and design matrix. The approximate MLE is based on easily computed univariate MGFs and can accommodate noise from general distributions. We cast the problem as a constrained mathematical program and provide an expression for the corresponding gradient allowing for use of "off-the-shelf" first-order solvers and discuss algorithmic considerations. Although the saddle point method for approximating densities (and maximum likelihood estimation) is known in statistical literature, its use for parameter estimation in the signal processing community is limited; a central goal of this work is to bridge the gap.

In Section 3.2, we provide the mathematical background for the proposed method. We formulate and present our approximate (log) likelihood function and its gradient in Section 3.3 then introduce algorithms to solve the approximate MLE problem in Section 3.4. We motivate its utility through illustrative examples in Section 3.5 then present results of numerical experiments in Section 3.6.

3.2 Background

Maximum likelihood estimation is one of the most commonly used techniques in statistics. MLEs require the construction of a LF which varies in difficulty. In the regression problem, we are concerned with sums of random variables and their corresponding densities. Although there are several well known distributions for sums of random variables, such as sums of normals, χ^2's, exponentials, etc., the majority of distributions do not enjoy elegant forms. Furthermore, many densities for sums require them to be identical, severely curtailing their usefulness; linear combinations are out of reach. Density construction is possible through convolution but presents serious difficulties in practice.

Rather than manipulating densities directly, we can work with their moment generating functions (MGF). The idea is closely related to Fourier analysis where a convolution in time becomes multiplication in the frequency domain. The MGF is simply a bilateral Laplace transform of the

PDF, f_X, given by

$$M_X(t) = \mathbb{E}\left(e^{tX}\right) = \int_{-\infty}^{\infty} e^{tx} f_X(x)\, dx. \tag{3.3}$$

The existence of an MGF is not guaranteed, but when it does exist, it uniquely characterizes the random variable. We exploit this idea by encoding the random variable's statistics in the moment generating function, then use it to construct an approximate LF.

To illustrate, let $U \sim \text{Uniform}(0,1)$ and $Z \sim \mathcal{N}(0,1)$ with known densities of $f_U(u) = I_{(0,1)}(u)$ and $f_Z(z) = (1/\sqrt{2\pi}) \exp\{-z^2/2\}$, respectively (we use $I_{\mathcal{A}}$ to denote indicator function over $\mathcal{A}$). The density of $U + V$ is

$$f_{U+Z}(y) = \frac{1}{\sqrt{2\pi}} \int_0^1 e^{-\frac{(y-s)^2}{2}}\, ds$$

which has no analytic form. In contrast, the MGFs are $M_U(t) = (\exp\{t\} - 1)/t$ and $M_Z(t) = \exp\{-t^2/2\}$. The MGF of their sum is

$$M_{U+Z}(t) = \frac{(e^t - 1)e^{-t^2/2}}{t}.$$

The MGF appears more complicated but is *exact*. In contrast, the PDF requires evaluation of an integral at each point and relies on approximation through numerical integration. Use of the PDF presents no real difficulty in our example, but as the number of random variables in the sum increase or distributions vary, construction and/or evaluation of the PDF, as well as finding the gradient of its associated likelihood, becomes problematic.

Using the generative model, we aim to form a likelihood function, L, based on observations of $\mathbf{y}$. We assume that both $\mathbf{G}$ and $\boldsymbol{\eta}$ are component-wise independent and that their distributions are known. We cannot observe $\mathbf{G}$ or $\boldsymbol{\eta}$ directly, but instead observe $\mathbf{y}$ which is a function of both. For notational simplicity, let $\mathbf{G} \sim \mathcal{P}_{\mathbf{G}}$ and $\boldsymbol{\eta} \sim \mathcal{P}_{\boldsymbol{\eta}}$ with $\mathcal{P}$ denoting the respective distributions. The likelihood will be a function of $\mathbf{x}$ and depend on $\mathcal{P}_{\mathbf{G}}, \mathcal{P}_{\boldsymbol{\eta}},$ and $\mathbf{y}$ given as

$$L(\mathbf{x}) = p(\mathbf{y};\, \mathbf{x}, \mathcal{P}_{\mathbf{G}}, \mathcal{P}_{\boldsymbol{\eta}}). \tag{3.4}$$

We require a closed form expression for p, the PDF of $\mathbf{y} = \mathbf{G}\mathbf{x} + \boldsymbol{\eta}$, which is a component-wise weighted sum of random variables. As discussed earlier, it is difficult to compute in general.

Given the trouble of forming an exact PDF, we focus on approximation methods. One option is to randomly sample the distribution then form a kernel approximation [Rosenblatt, 1956, Parzen, 1962]. Unfortunately, kernel density estimation requires many samples to adequately capture the structure of the distribution and its accuracy is influenced by the user's choice of a kernel, making in unsuitable for our regression problem. Another popular choice is to use an Edgeworth series expansion [Hall, 2013] where a polynomial approximation for the density is used that matches the first several cumulants of the true density. A major drawback of the method is that it introduces false critical points unrelated to the actual density. Since the ultimate objective is to maximize the LF, the addition of phantom critical points creates otherwise avoidable difficulties. Furthermore, the Edgeworth expansion can take negative values which violates the properties of a PDF. At this point, we turn our attention to the saddle point method to approximate our PDF based on the exact MGF.

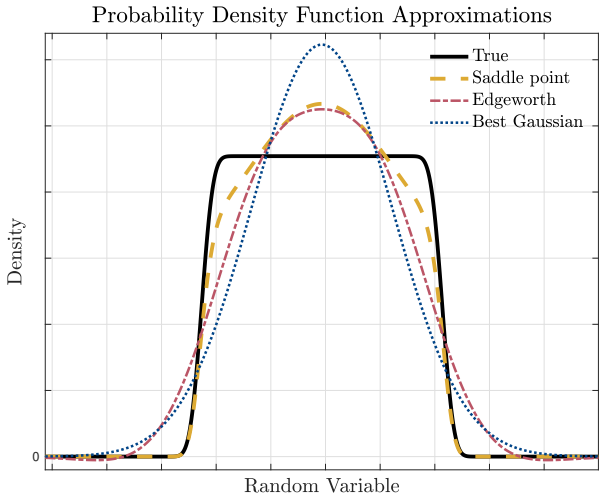

Figure 3.1: Example of the true density for y and several approximations when $y = \mathbf{g}^T\mathbf{x} + \eta$ with $\mathbf{g}$ uniform, $\mathbf{x}$ from a Cauchy distribution, and η normal. Note that $\mathbf{x}$ is drawn randomly to start but fixed for pdf of y. The saddle point approximation fits distributional tails exceptionally well. Note that Edgeworth expansion takes negative values and introduces phantom critical points. The best Gaussian has a matching mean and variance as the true distribution.

3.2.1 Saddle point approximation

We outline the principle behind the saddle point method here with an informal treatment. The method is closely related to the method of steepest descent and the stationary-phase method. For rigorous arguments, the interested reader may refer to the original paper [Daniels, 1954] or one of the excellent overview articles [Reid, 1988, Goutis and Casella, 1999, Huzurbazar, 1999]. Our aim is to give a general idea of the method for illustrative purposes.

We start with a well known theorem in probability stating that for a random variable A, if the MGF $M_A(iw)$ is integrable (with $i = \sqrt{-1}$), then it has a PDF given by

$$f_A(\alpha) = \frac{1}{2\pi} \int_{-\infty}^{\infty} M_A(iw)\, e^{-iw\alpha}\, dw.$$

This is the inverse Fourier transform of the MGF with a complex argument. By a change of variable, $t = iw$, the integral becomes

$$\begin{aligned} f_A(\alpha) &= \frac{1}{2\pi i} \int_{-i\infty}^{i\infty} M_A(t)\, e^{-t\alpha}\, dt \\ &= \frac{1}{2\pi i} \int_{-i\infty}^{i\infty} e^{\ln M_A(t) - t\alpha}\, dt. \end{aligned}$$

For notational simplicity, the *cumulant generating function* (CGF) is defined as $K_A(\alpha) = \ln M_A(\alpha)$. Assuming reasonable regularity conditions and by the Closed Curve Theorem, we can rewrite an equivalent integral with our contour translated by $\tau \in \mathbb{R}$ (to be fixed shortly) along the real axis such that

$$f_A(\alpha) = \frac{1}{2\pi i} \int_{\tau-i\infty}^{\tau+i\infty} e^{K_A(t) - t\alpha}\, dt.$$

To approximate the integral, the exponent is Taylor expanded about its maximum value on the contour. This agrees with intuition; the approximation will be accurate where the integral's mass lies. Points where the expansion is less accurate are effectively down-weighted through exponentiation. The quality of the approximation is contingent on the nature of $K_A(t) - t\alpha$, but is favorable for sums of random variables.

The integrand takes its maximum value along the contour at a critical point, that is, when $K'_A(t) - \alpha = 0$. It so happens that when α is in the support of the PDF for A, there is a unique

real root for the preceding equation [Daniels, 1954]. We call this point $t_0 \in \mathbb{R}$. Since the maximum occurs on a contour parallel to the imaginary axis, a minima occurs at the same point along the real axis because the exponent is analytic and must fulfill the Cauchy-Riemann equations, hence $K''_A(t_0) > 0$. Taylor expanding about t_0 and shifting the contour to pass through it, i.e., $\tau = t_0$, gives,

$$\begin{aligned} f_A(\alpha) &\approx \frac{1}{2\pi i}\int_{t_0-i\infty}^{t_0+i\infty} \exp\Big\{(K_A(t_0) - t_0\alpha) \\ &\quad + (K'_A(t_0) - \alpha)(t - t_0) + \frac{1}{2}K''_A(t_0)(t - t_0)^2\Big\}\, dt \\ &= \frac{1}{2\pi i} e^{K_A(t_0)-t_0\alpha}\int_{t_0-i\infty}^{t_0+i\infty} e^{\frac{1}{2}K''_A(t_0)(t-t_0)^2}\, dt \end{aligned}$$

Through successive substitutions, the integrand can be rewritten as a Gaussian function and integrated over $\mathbb{R}$ yielding an approximation of the PDF for A,

$$f_A(\alpha) \approx \tilde{f}_A(\alpha) = \sqrt{\frac{1}{2\pi K''_A(t_0)}}\; e^{K_A(t_0)-t_0\alpha} \tag{3.5}$$

where t_0 depends on α through the equation $K'_A(t_0) - \alpha = 0$. The function $\tilde{f}_A(\alpha)$ is known as the *saddle point approximation* for f_A at point α. Higher order approximations can be achieved by including more terms in the Taylor expansion, but we limit our discussion to quadratics.

3.2.2 Notation

Bold lower/upper case letters denote vectors/matrices, while unbolded lower-case letters are scalars. For a matrix $\mathbf{G}$, the i^{th} row and j^{th} column are given by $\mathbf{g}_i^T$ and $\mathbf{g}_j$, respectively. A vector $\mathbf{y}$ is composed of components y_i with the subscript denoting the index of an element. We write the joint CGF of a random variable $\mathbf{y} \in \mathbb{R}^m$ as $K_{\mathbf{y}}(\mathbf{t}) = [K_{y_1}(t_1), ..., K_{y_m}(t_m)]^T$ and its derivatives $K^{(p)}_{\mathbf{y}}(\mathbf{t}) = [\frac{\partial^p}{\partial t_1^p}K_{y_1}(t_1), ..., \frac{\partial^p}{\partial t_m^p}K_{y_m}(t_m)]^T$. Unless otherwise noted, the variables $\mathbf{t}$ and t_i are reserved to denote solutions to equation $K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) - \mathbf{y} = \mathbf{0}$ where t_i is the solution to the i^{th} component of the vector equation.

A scalar function $f : \mathbb{R} \to \mathbb{R}$ applied to a matrix $\mathbf{A}$ acts component-wise, i.e., $[f(\mathbf{A})]_{ij} = f(\mathbf{A}_{ij})$. The Hadamard (or element-wise) product/quotient is denoted by $\odot/\oslash$. For a vector

$\mathbf{y} \in \mathbb{R}^m$, the matrix $\text{diag}(\mathbf{y}) \in \mathbb{R}^{m\times m}$ has the components of $\mathbf{y}$ along its diagonal and zeros elsewhere. A vector of ones and zeros is denoted by $\mathbf{1}$ and $\mathbf{0}$, respectively.

3.3 Likelihood function

With an approximate density in hand, we can use (3.4) and (3.5) to form a likelihood function (LF). Begin by noting that the joint density for a random vector with independent components can be rewritten as a product,

$$\begin{aligned} p(\mathbf{y};\, \mathbf{x}, \mathcal{P}_{\mathbf{G}}, \mathcal{P}_{\boldsymbol{\eta}}) &= \prod_{i=1}^{m} f_{Y_i}\left(y_i;\, \mathbf{x}, \mathcal{P}_{\mathbf{G}}, \mathcal{P}_{\boldsymbol{\eta}}\right) \\ &= \prod_{i=1}^{m} f_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(y_i), \end{aligned} \tag{3.6}$$

where f_{Y_i} is the PDF for the i^{th} component. Since PDF construction is difficult for sums of random variables, we use the saddle point approximation and the LF from (3.4) such that

$$L(\mathbf{x}) = p(\mathbf{y};\, \mathbf{x}, \mathcal{P}_{\mathbf{G}}, \mathcal{P}_{\boldsymbol{\eta}}) \approx \prod_{i=1}^{m} \tilde{f}_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(y_i). \tag{3.7}$$

We now define our *approximate-likelihood function* (or approximate-LF) as

$$\begin{aligned} \mathcal{L}(\mathbf{x}) = \exp&\left\{\sum_{i=1}^{m} K_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i) - t_i y_i\right\} \\ &\cdot \left[\prod_{i=1}^{m}\left(2\pi\, K''_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i)\right)^{-1/2}\right] \end{aligned} \tag{3.8}$$

which assumes known measurements for y_i and distributional knowledge of $\mathbf{G}$ and $\boldsymbol{\eta}$. Recall that t_i solves $K'_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i) = y_i$.

As with any MLE problem, our goal is to find $\mathbf{x}$ that maximizes the LF. For numerical stability, we focus on the log-likelihood function (log-LF), i.e., $\ell(\mathbf{x}) = \ln \mathcal{L}(\mathbf{x})$, which has the same maximizer as (3.8) (we also omit the constant 2π since it doesn't impact the location of optima) and is given by

$$\ell(\mathbf{x}) = \sum_{i=1}^{m}\left[K_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i) - \frac{1}{2}\ln\left(K''_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i)\right) - t_i y_i\right]. \tag{3.9}$$

By exploiting the independence of components for both $\mathbf{G}$ and $\boldsymbol{\eta}$ and using properties of MGFs, we can write $K_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i) = K_{\eta_i}(t_i) + \sum_{j=1}^{n} K_{G_{ij}}(x_j t_i)$. Since both η_i and G_{ij} have known CGFs, explicit calculation of $\ell(\mathbf{x})$ is easy and the log-LF is given by

$$\ell(\mathbf{x}) = \sum_{i=1}^{m} \left[K_{\eta_i}(t_i) \; + \; \left(\sum_{j=1}^{n} K_{G_{ij}}(t_i x_j) \right) \right.$$
$$\left. - \; t_i y_i - \frac{1}{2} \ln \left(K''_{\eta_i}(t_i) + \sum_{j=1}^{n} K''_{G_{ij}}(t_i x_j) \right) \right]. \tag{3.10}$$

We remind the reader that each t_i is a function of y_i and $\mathbf{x}$ through the critical point equation although we leave the dependence out for notational simplicity. If functional forms of $K_{\mathbf{G}\mathbf{x}+\boldsymbol{\eta}}(\mathbf{t})$ and its derivatives are known, it is easier to work with the vectorized version, written as

$$\ell(\mathbf{x}) = \mathbf{1}^T \left(K_{\mathbf{G}\mathbf{x}+\boldsymbol{\eta}}(\mathbf{t}) - \frac{1}{2} \ln \left(K''_{\mathbf{G}\mathbf{x}+\boldsymbol{\eta}}(\mathbf{t}) \right) \right) - \mathbf{t}^T \mathbf{y}. \tag{3.11}$$

Functional forms for a particular problem are typically found by working directly with component-wise elements (Equation 3.10) first.

Under the framework presented here, the observed matrix $\mathbf{G}$ goes unused since all pertinent statistical information is embodied in the CGF. In practice, we envision the observed $\mathbf{G}$ as the sample mean with noise distributed about it, as will be illustrated in Section 3.5. If distributional parameters are unknown, such as variance for a Gaussian, we could treat them as variables in our log-LF. The actual formulation and gradient calculations would change only slightly, although the problem may have more spurious stationary points.

3.3.1 Log-likelihood gradient

Although we have a log-LF, we must still maximize it. Most optimization algorithms rely on gradients so we focus on that computation now. To complicate matters, our log-LF depends on $\mathbf{t}$ which is coupled to $\mathbf{x}$ and, in general, has no closed form solution. To deal with this, we proceed with implicit differentiation as is done in the adjoint state method by treating $\mathbf{t}$ as an independent

variable. We recast our MLE problem as a mathematical program

$$\begin{aligned} \underset{\mathbf{x},\mathbf{t}}{\text{argmax}} \quad & \ell(\mathbf{x},\mathbf{t}) \\ \text{subject to} \quad & K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) - \mathbf{y} = \mathbf{0}. \end{aligned} \tag{3.12}$$

It follows from the chain rule that

$$\nabla_{\mathbf{x}}\ell = \frac{\partial \ell}{\partial \mathbf{x}} + \left(\frac{\partial \ell}{\partial \mathbf{t}}\right)\left(\frac{\mathrm{d}\mathbf{t}}{\mathrm{d}\mathbf{x}}\right). \tag{3.13}$$

We follow the convention for Jacobians that the column indicates which variable is being differentiated and the row represents the component, i.e., $(\mathrm{d}\mathbf{t}/\mathrm{d}\mathbf{x})_{ij} = \partial t_i/\partial x_j$. By treating $\mathbf{x}$ or $\mathbf{t}$ as fixed, it is straightforward to calculate the partial derivatives of $\ell(\mathbf{x},\mathbf{t})$. It remains to find $(\mathrm{d}\mathbf{t}/\mathrm{d}\mathbf{x}) \in \mathbb{R}^{m\times n}$.

To ease notation, let $\mathbf{q}(\mathbf{x},\mathbf{t}) = K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) - \mathbf{y}$. Differentiating the constraint $\mathbf{q}(\mathbf{x},\mathbf{t}) = 0$ gives

$$\frac{\partial \mathbf{q}}{\partial \mathbf{x}} + \left(\frac{\partial \mathbf{q}}{\partial \mathbf{t}}\right)\left(\frac{\mathrm{d}\mathbf{t}}{\mathrm{d}\mathbf{x}}\right) = \mathbf{0} \iff \frac{\partial \mathbf{t}}{\partial \mathbf{x}} = -\left(\frac{\partial \mathbf{q}}{\partial \mathbf{t}}\right)^{-1}\left(\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right). \tag{3.14}$$

Combining (3.13) and (3.14) yields an expression for the gradient

$$\nabla_{\mathbf{x}}\ell = \frac{\partial \ell}{\partial \mathbf{x}} - \left(\frac{\partial \ell}{\partial \mathbf{t}}\right)\left(\frac{\partial \mathbf{q}}{\partial \mathbf{t}}\right)^{-1}\left(\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right). \tag{3.15}$$

Rewriting in terms of CGFs and their derivatives, we have

$$\frac{\partial \ell}{\partial \mathbf{x}} = \mathbf{1}^T\left(\frac{\partial}{\partial \mathbf{x}}K_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) - \frac{1}{2}\left\{\mathrm{diag}\left(K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t})\right)\right\}^{-1}\frac{\partial}{\partial \mathbf{x}}K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t})\right), \tag{3.16}$$

$$\frac{\partial \ell}{\partial \mathbf{t}} = \left(K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) - \frac{1}{2}\left(K'''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) \oslash K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t})\right) - \mathbf{y}\right)^T, \tag{3.17}$$

$$\frac{\partial \mathbf{q}}{\partial \mathbf{t}} = \mathrm{diag}\left(K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t})\right), \tag{3.18}$$

$$\frac{\partial \mathbf{q}}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}}K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}), \tag{3.19}$$

where $\mathbf{t}$ must be found for a given $\mathbf{x}$ by solving $\mathbf{q}(\mathbf{x},\mathbf{t}) = \mathbf{0}$. With enough patience, these derivatives can generally be calculated by hand. We present gradients for the example problems in Section 3.5 in the appendix.

3.4 Algorithms

So far, we proposed a method for constructing an approximate log-LF using noise from general distributions and discussed methods for calculating its gradient. Success rests on our ability to efficiently and accurately optimize ℓ as cast generically in (3.12). There are several computational challenges for maximizing the approximate log-LF.

First, most optimization algorithms require access to the objective's gradient. Although we present a method for obtaining a gradient when $\mathbf{t}$ must be determined numerically, it can be a challenging task. For those who prefer not to toil endlessly over a cruel calculus exercise, automatic differentiation software can be employed. ADiMat is a popular package for MATLAB [Bischof et al., 2002]. It supports reverse-mode differentiation (known as back-propogation in the context of training neural nets) which is desirable for scalar objectives of many variables. A list of packages for different languages can be found at `http://www.autodiff.org`.

The second challenge follows from the non-concavity of ℓ in $\mathbf{x}$, since most algorithms can only guarantee convergence to stationary points or, at best, local maximizers, and globalization strategies are either heuristics or computationally infeasible. This is a challenge for most MLE problems, not just our formulation. Experimental evidence using multiple initializations suggests that the non-concavity effect is quite mild, especially when initializing with a reasonable guess. We found that using the OLS estimator works well as an initial guess, as it is cheap to compute and more robust than the TLS estimator.

As the problem is unconstrained with a smooth objective function, we use the well-known quasi-Newton method L-BFGS [Nocedal, 1980] via the MATLAB package `minFunc` [Schmidt, 2012] because L-BFGS converges more quickly than gradient descent yet doesn't require the second-order derivative information nor matrix inversion of Newton's method. Since `minFunc` is a minimization routine, we minimize $-\ell(\mathbf{x})$ in practice rather than maximimizing $\ell(\mathbf{x})$. L-BFGS performs well on moderate or even large problems. Very large problems could be approached via stochastic gradient methods, using a subset of data to estimate a gradient, but such implementations are beyond the

scope of this paper.

3.5 Examples

The following examples consider the generative model, $\mathbf{y} = \mathbf{G}\mathbf{x} + \boldsymbol{\eta}$. Although we are free to use additive noise from other distributions depending on the problem, we elected to use Gaussian noise for clarity of exposition. In particular, $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. The corresponding CGFs and their derivatives are

$$K_{\eta_i}(t_i) = \sigma^2 t_i^2/2, \quad K'_{\eta_i}(t_i) = \sigma^2 t_i, \quad K''_{\eta_i}(t_i) = \sigma^2. \tag{3.20}$$

The log-LF for each example is derived and provided below. All the corresponding derivatives used for gradient calculations can be found in the appendix. We remind the reader that although analytic expressions for the gradient are possible, automatic differentiation eliminates the need for messy calculations.

3.5.1 Rounding error

Suppose that we observe $\mathbf{y}$ and a rounded version of $\mathbf{G}$, denoted by $\mathbf{H}$. For concreteness, assume all elements of $\mathbf{G}$ are rounded to the ones spot, e.g., $1.4 \to 1$. In this case, we can model $\mathbf{G}$ as a uniform random matrix of mean $\mathbf{H}$. Parameter values specifying the support of the uniform random variables can be inferred from the data. If elements are rounded to the ones spot, then the uncertainty parameter will be $\delta = 0.5$; if rounded to the tens, it will be $\delta = 5$, etc. Gaussian additive noise of known variance is assumed for $\boldsymbol{\eta}$.

To highlight the difficulties of forming an exact likelihood for this problem, we note that $\mathbf{y}$ is the joint density of linear combinations of uniform and Gaussian random variables. The Irwin-Hall distribution covers the case of i.i.d. uniform random variables on $(0, 1)$ which has a piece-wise polynomial PDF. Weighted sums of uniform random variables were studied in [Kamgar-Parsi et al., 1995] where the authors showed that the density can be written as a sum of polynomials through extensive use of the Heaviside function. Unfortunately, the number of terms in the expansion grows exponentially; a LF of 30 variables contains over a billion terms. The PDF for the sum of uniforms

must still be convolved with a Gaussian density to obtain the true density for a single component of $\mathbf{y}$. With $\mathbf{y} \in \mathbb{R}^m$, a single likelihood evaluation requires summing a billion terms m times! Needless to say, the problem is intractable using an exact likelihood.

The approximate likelihood presents a simple and appealing alternative. As shown in (3.10), we can focus on element-wise, univariate, random variables to form our approximate log-LF. The generating functions used to construct the LF can be found in most statistical texts. In particular, the CGF for $G_{ij} \sim \text{Uniform}(H_{ij} - \delta, H_{ij} + \delta)$ can be simplified to

$$K_{G_{ij}}(t_i x_j) = t_i x_j H_{ij} + \ln\left(\frac{\sinh(\delta t_i x_j)}{\delta t_i x_j}\right), \tag{3.21}$$

We also require the corresponding derivatives given by

$$\begin{aligned} K'_{G_{ij}}(t_i x_j) &= H_{ij} x_j - \frac{1}{t_i} + \delta x_j \coth(\delta t_i x_j), \\ K''_{G_{ij}}(t_i x_j) &= \frac{1}{t_i^2} - \delta^2 x_j^2 \text{csch}^2(\delta t_i x_j). \end{aligned} \tag{3.22}$$

Putting it together gives the log-LF

$$\begin{aligned} \ell(\mathbf{x}) = \sum_{i=1}^{m} \Bigg\{ & \frac{\sigma^2 t_i^2}{2} + \sum_{j=1}^{n} \left[H_{ij} t_i x_j \; + \; \ln\left(\frac{\sinh(\delta t_i x_j)}{\delta t_i x_j}\right)\right] \\ & - \; t_i y_i - \; \frac{1}{2} \ln\left[\sigma^2 + \sum_{j=1}^{n}\left(\frac{1}{t_i^2} - \delta x_j^2 \text{csch}^2(\delta t_i x_j)\right)\right] \Bigg\} \end{aligned} \tag{3.23}$$

and can be written in matrix/vector form as

$$\begin{aligned} \ell(\mathbf{x}) = & \mathbf{t}^T \left(\frac{\sigma^2}{2}\,\mathbf{t} + \mathbf{H}\,\mathbf{x} - \mathbf{y}\right) \\ & + \mathbf{1}^T \ln\left[\sinh\left(\delta \mathbf{t}\mathbf{x}^T\right) \oslash \left(\delta \mathbf{t}\mathbf{x}^T\right)\right] \mathbf{1} \\ & - \frac{1}{2}\mathbf{1}^T \ln\left[\sigma^2\,\mathbf{1} + n \oslash \mathbf{t}^2 - \delta^2 \,\text{csch}^2\left(\delta \mathbf{t}\mathbf{x}^T\right)\mathbf{x}^2\right], \end{aligned} \tag{3.24}$$

where $\mathbf{t}$ solves the equation $K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \mathbf{y}$.

The above example is principled and can be used in instances of rounding. The observed design matrix is used directly since it (along with δ) completely specifies the distribution from which our "true" design matrix was drawn. The distribution of $\mathbf{G}$ and all its parameters can be inferred by observing $\mathbf{H}$.

3.5.2 Floating point uncertainty

For the case of floating point uncertainty, all appearances of δ should be replaced by D_{ij} or $\mathbf{D}$ depending on whether it appears in an element-wise or matrix/vector equation, respectively. Specifically, by letting $\mathbf{D}$ be a non-negative matrix specifying the uncertainty based on the number of significant figures included. For example, if $H_{ij} = 1.7 \times 10^4$ then $D_{ij} = 0.05 \times 10^4$. Setting $\mathbf{M} = \mathbf{D} \odot (\mathbf{t}\mathbf{x}^T)$, we have

$$\begin{aligned}\ell(\mathbf{x}) =& \mathbf{t}^T \left(\frac{\sigma^2}{2} \mathbf{t} + \mathbf{H}\,\mathbf{x} - \mathbf{y} \right) + \mathbf{1}^T \ln\left[\sinh\left(\mathbf{M}\right) \oslash \mathbf{M}\right] \mathbf{1} \\ & - \frac{1}{2} \mathbf{1}^T \ln\left[\sigma^2\,\mathbf{1} + n \oslash \mathbf{t}^2 - \left(\mathbf{D}^2 \odot \operatorname{csch}^2\left(\mathbf{M}\right)\right) \mathbf{x}^2\right]. \end{aligned} \tag{3.25}$$

The derivatives provided in the appendix are for the floating-point uncertainty which reduces to fixed-point or rounding error by letting $\mathbf{D} = \delta \mathbf{1}_m \mathbf{1}_n^T$.

3.5.3 Exponential clipping

Suppose that the the elements of $\mathbf{G}$ are drawn from a double exponential distribution with rate λ and that entries with magnitudes greater than some threshold, γ say, are clipped. The clipped matrix is given by

$$\mathbf{H} = \operatorname{sign}(\mathbf{G}) \cdot \min\{|\mathbf{G}|, \gamma\},$$

with $\min\{\cdot, \gamma\}$ and the absolute values operating element-wise. In this case, uncertainty is only realized for clipped components. Let $\mathbf{S} = \operatorname{sign}(\mathbf{H})$ and $\mathbf{A}$ be defined by

$$A_{ij} = \begin{cases} 1, & |H_{ij}| = \gamma, \\ 0, & \text{else.} \end{cases}$$

By the memorylessness property of exponentials, the uncertainty of clipped entries will also be $\pm\text{Exponential}(\lambda)$. Using Gaussian additive noise, the remaining CGFs and corresponding deriva-

tives are given by

$$\begin{aligned} K_{G_{ij}}(t_i x_j) &= t_i H_{ij} x_j - A_{ij} \ln\left(1 - \frac{S_{ij} t_i x_j}{\lambda}\right), \\ K'_{G_{ij}}(t_i x_j) &= H_{ij} x_j + \frac{A_{ij} S_{ij} x_j}{\lambda - S_{ij} t_i x_j}, \\ K''_{G_{ij}}(t_i x_j) &= \frac{A_{ij} x_j^2}{(\lambda - S_{ij} t_i x_j)^2}. \end{aligned} \tag{3.26}$$

Using the CGFs above, the approximate log-LF is given by

$$\begin{aligned} \ell(\mathbf{x}) = \sum_{i=1}^{m} \Bigg\{ & \frac{\sigma^2 t_i^2}{2} + \sum_{j=1}^{n} \left[t_i H_{ij} x_j - A_{ij} \ln\left(1 - \frac{S_{ij} t_i x_j}{\lambda}\right) \right] \\ & - t_i y_i - \frac{1}{2} \ln \left[\sigma^2 - \sum_{j=1}^{n} \frac{A_{ij}\, x_j^2}{(\lambda - S_{ij} t_i x_j)^2} \right] \Bigg\}. \end{aligned} \tag{3.27}$$

Letting $\mathbf{\Lambda} = \lambda \mathbf{1}_m \mathbf{1}_n^T \in \mathbb{R}^{m \times n}$ and $\mathbf{C} = \mathbf{S} \odot (\mathbf{t}\mathbf{x}^T)$ the matrix/vector form is

$$\begin{aligned} \ell(\mathbf{x}) = & \mathbf{t}^T \left(\frac{\sigma^2 \mathbf{t}}{2} + \mathbf{H}\mathbf{x} - \mathbf{y} \right) + \mathbf{1}^T \left(\mathbf{A} \odot \ln\left[\mathbf{\Lambda} \oslash (\mathbf{\Lambda} - \mathbf{C}) \right] \right) \mathbf{1} \\ & - \frac{1}{2} \mathbf{1}^T \ln\left\{ \sigma^2 \mathbf{1} + \left[\mathbf{A} \oslash (\mathbf{\Lambda} - \mathbf{C})^2 \right] \mathbf{x}^2 \right\}. \end{aligned} \tag{3.28}$$

Derivatives for construction of the gradient can be found in the appendix.

It can be observed that the approximate log-LF above and its derivatives have singularities depending on component values of $\mathbf{x}$. Since root finding algorithms generally require continuity, complications arise when solving for the $\mathbf{t}$ that verifies $K'_{\mathbf{G}\mathbf{x}+\boldsymbol{\eta}}(\mathbf{t}) = \mathbf{y}$. Consequently, a modified root-finding algorithm must be employed that brackets a continuous interval about zero.

3.5.4 Gaussian uncertainty

For our final example, both the design matrix and measurement vector are subject to Gaussian noise. In this case, there exists an exact closed form likelihood that can be derived directly from the joint density and we show that our method recovers the exact log-LF. Suppose that $\mathbf{G}$ is Gaussian with mean $\mathbf{H}$ and element-wise variance ρ^2. Using the same additive noise of as the other examples, the remaining CGF is

$$K_{G_{ij}}(t_i x_j) = \mathbf{h}_i^T \mathbf{x}\, t_i + \frac{1}{2} \rho^2 t_i^2 x_j^2. \tag{3.29}$$

Recalling that t_i is the solution to $K'_{\mathbf{g}_i^T\mathbf{x}+\eta_i}(t_i) = y_i$, we can solve for t_i giving

$$t_i = \frac{y_i - \mathbf{h}_i^T\mathbf{x}}{\sigma^2 + \rho^2\|\mathbf{x}\|^2}. \tag{3.30}$$

Plugging (3.29) and (3.30) into (3.10) gives our "approximate" log-LF,

$$\ell(\mathbf{x}) = -\frac{1}{2}\left[\frac{\|\mathbf{y}-\mathbf{H}\mathbf{x}\|^2}{\sigma^2+\rho^2\|\mathbf{x}\|^2} + m\,\ln\left(\sigma^2+\rho^2\|\mathbf{x}\|^2\right)\right]. \tag{3.31}$$

By solving for $\mathbf{t}$ as a function of $\mathbf{x}$, the gradient is given directly by

$$\nabla_{\mathbf{x}}\ell(\mathbf{x}) = \frac{-1}{\sigma^2+\rho^2\|\mathbf{x}\|^2}\left[\mathbf{H}^T(\mathbf{H}\mathbf{x}-\mathbf{y}) - \rho^2\left(\frac{\|\mathbf{H}\mathbf{x}-\mathbf{y}\|^2}{\sigma^2+\rho^2\|\mathbf{x}\|^2} - m\right)\mathbf{x}\right]. \tag{3.32}$$

We note that $\ell(\mathbf{x})$ is the *exact* log-likelihood, up to a constant, as the one derived in [Wiesel et al., 2008]. This follows from the fact that a Gaussian is uniquely determined by its first and second cumulants, i.e., mean and variance. The truncated Taylor series used for the saddle point approximation is precise by virtue of all higher-derivatives being zero.

3.6 Numerical experiments

To validate the approximate likelihood estimator, we implemented our proposed method along with OLS and TLS on the model examples discussed in Section 3.5. Although it is possible to use a weighting matrix for TLS to account for non-uniform variance between the operator and measurement vector, we observed little benefit and opted to solve the unweighted version. In each case we used the generative model $\mathbf{y} = \mathbf{G}\mathbf{x}_{\text{TRU}} + \boldsymbol{\eta}$ with $\boldsymbol{\eta} \sim \mathcal{N}\left(\mathbf{0}, (0.1)^2\mathbf{I}\right)$. The variance on the additive Gaussian noise was chosen as to not overwhelm uncertainty in $\mathbf{G}$. The "true" solution vectors, denoted by $\mathbf{x}_{\text{TRU}}$, were drawn from a heavy-tailed Cauchy distribution; this distribution makes the use of prior information on the solution difficult. Although $\mathbf{x}_{\text{TRU}}$ is randomly drawn, it is fixed for each simulation and does not introduce uncertainty into the problem. The vector $\mathbf{y}$ is observed.

For each example, the matrices in question were created as follows:

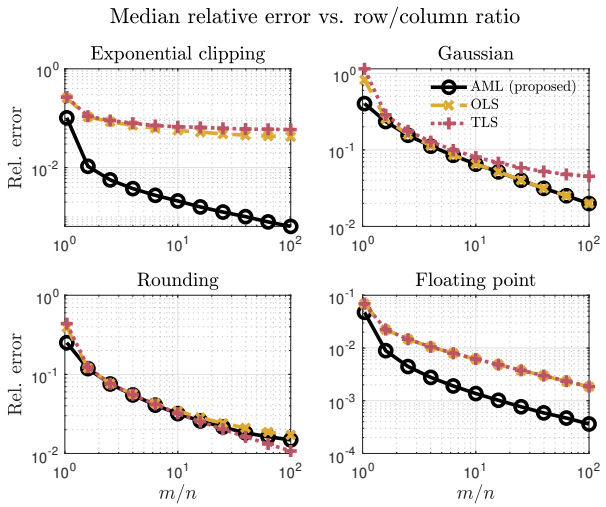

Figure 3.2: Median relative error $\left(\frac{\|\mathbf{x}_{\text{EST}}-\mathbf{x}_{\text{TRU}}\|}{\|\mathbf{x}_{\text{TRU}}\|}\right)$ over 1,000 simulations. Number of columns fixed at $n = 20$ while row count varied from $m = 21$ to $m = 2,000$ showing estimator performance with additional data.

- **Rounding**: The elements of $\mathbf{G}$ were drawn from a continuous uniform distribution supported on $(0, 10)$. The observed matrix $\mathbf{H}$ was constructed by rounding $\mathbf{G}$ to the ones spot, i.e., $\mathbf{H} = \text{Round}(\mathbf{G})$ with uncertainty parameter $\delta = 0.5$.

- **Floating point**: The elements of $\mathbf{G}$ are the product of a standard normal Gaussian and 10^k where k is uniformly chosen from $\{0, 1, 2, 3\}$. The observed matrix $\mathbf{H}$ is given by keeping two significant figures. The uncertainty for a particular element is scaled correspondingly, i.e., if $H_{ij} = -3.1 \times 10^2$ then $D_{ij} = 0.05 \times 10^2$.

- **Exponential clipping**: The elements of $\mathbf{G}$ were drawn from a double exponential or Laplace distribution with rate $\lambda = 2$. Fixing the clipping threshold at $\gamma = 2$, matrix $\mathbf{H} = \text{sign}(\mathbf{G}) \max\{|\mathbf{G}|, \gamma\}$ is observed. We note that for $\lambda = \gamma = 2$, approximately 2% of matrix elements are clipped.

- **Gaussian**: A mean observed matrix $\mathbf{H}$ was drawn randomly from a Gaussian of variance 100 (this merely fixes a mean and contributes no uncertainty to the problem). The elements of matrix $\mathbf{G}$ were then drawn from a Gaussian of mean $\mathbf{H}$ and variance $\rho^2 = 4$. The

Gaussian approximate MLE corresponds to the exact MLE.

For each listed problem, it is assumed that $\mathbf{H}$, $\mathbf{y}$, and the relevant distributional parameters are known. We conducted three simulations for each example instance:

(1) Fixed the number of columns at $n = 20$ then allowed the number of rows to increase from $m = 21$ to $m = 2000$ reflecting how the estimator responds to more data.

(2) Fixed the number of rows at $m = 100$ then allowed the number of columns to increase from $n = 1$ to $n = 99$ evaluating how the estimator responds to an increasing number of unknowns.

(3) For $m = 55$ and $n = 50$, we investigated the distribution of errors the approximate MLE error compares to that of OLS and TLS.

The approximate MLE, OLS, and TLS estimators are denoted by $\mathbf{x}_{\text{AML}}$, $\mathbf{x}_{\text{OLS}}$, and $\mathbf{x}_{\text{TLS}}$, respectively. A generic estimator is denoted by $\mathbf{x}_{\text{EST}}$.

The first two simulations summarized in Figs. 3.2 and 3.3 show that AML outperforms the other methods when the system is only slightly over determined. The method also excels in the highly over-determined regime for exponential clipping and floating point uncertainty. The relatively small extent of operator uncertainty as is typically encountered in the case of rounding error might explain why the proposed method fails to distinguish itself there. It is worth noting similar behavior for Gaussian uncertainty which corresponds to the **exact** MLE. As a result, we believe the modest performance stems from the limited scale of uncertainty rather than the method itself.

The large benefit of AML for limited data is further illustrated in simulation 3 as shown in Fig. 3.4. The figure gives box plots for each model and the error ratio or relative improvement of AML over OLS and TLS, i.e., $\frac{\|\mathbf{x}_{\text{AML}}-\mathbf{x}_{\text{TRU}}\|}{\|\mathbf{x}_{\text{OLS}}-\mathbf{x}_{\text{TRU}}\|}$ and $\frac{\|\mathbf{x}_{\text{AML}}-\mathbf{x}_{\text{TRU}}\|}{\|\mathbf{x}_{\text{TLS}}-\mathbf{x}_{\text{TRU}}\|}$, respectively. For the histogram, all values less than one (red line) are improvements over competing estimator for the same problem instance. Although AML does not always outperform the alternative methods, it usually does with

pronounced gains. The code use for simulations can be found at https://github.com/rclancyc/approximate_mle.

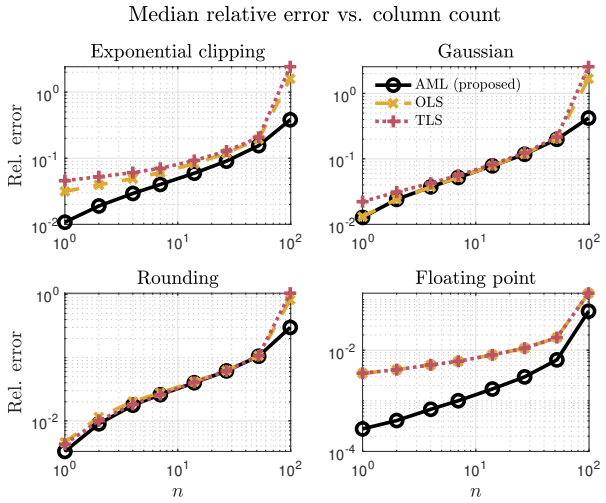

Figure 3.3: Median relative error $\left(\frac{\|\mathbf{x}_{\text{EST}}-\mathbf{x}_{\text{TRU}}\|}{\|\mathbf{x}_{\text{TRU}}\|}\right)$ over 1,000 simulations. Number of rows fixed at $m = 100$ while column count varied from $n = 1$ to $n = 99$ showing estimator performance for an increasing number of unknowns.

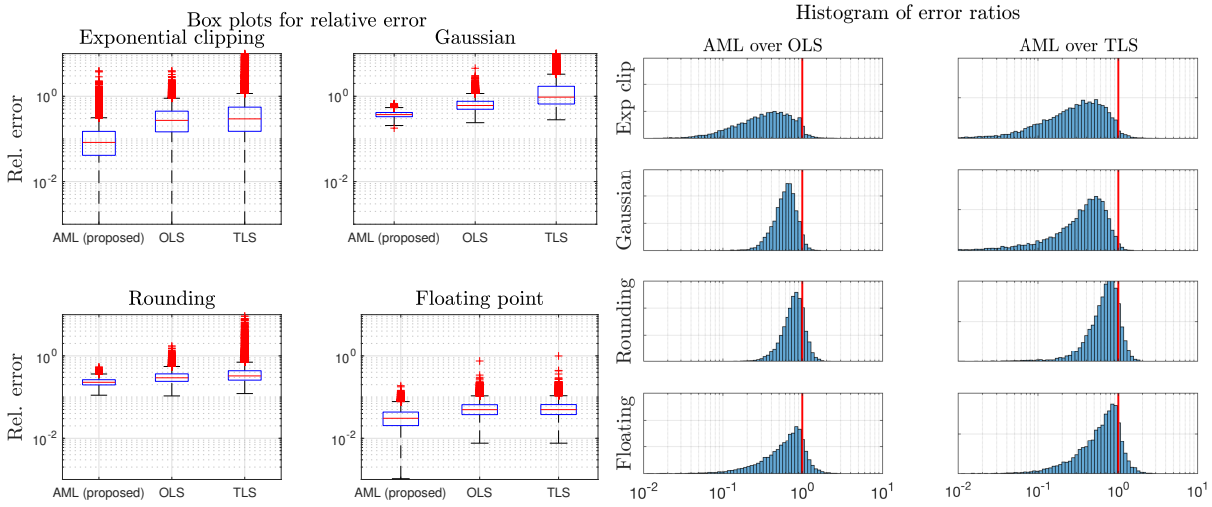

Figure 3.4: Error metrics for simulations $\mathbf{G} \in \mathbb{R}^{55\times 50}$ over $10,000$ simulations. Top: box-plots of relative error for different methods, i.e., $\frac{\|\mathbf{x}_{\text{TRU}}-\mathbf{x}_{\text{EST}}\|}{\|\mathbf{x}_{\text{TRU}}\|}$. Box indicates middle 50th quantile with interior line at the median. Bars denote extreme values with outliers indicated by "+". Bottom: histogram of error ratio $\frac{\|\mathbf{x}_{\text{AML}}-\mathbf{x}_{\text{TRU}}\|}{\|\mathbf{x}_{\text{OLS}}-\mathbf{x}_{\text{TRU}}\|}$ and $\frac{\|\mathbf{x}_{\text{AML}}-\mathbf{x}_{\text{TRU}}\|}{\|\mathbf{x}_{\text{TLS}}-\mathbf{x}_{\text{TRU}}\|}$. Values less than 1 (vertical line) indicate AML outperformed competing method for identical data.

3.7 Conclusion

In this work, we cast a regression problem in the maximum likelihood estimation framework. We discussed difficulties associated with forming a true and exact likelihood function for models with noise in the design matrix, then presented a method for generating an approximate log-LF based on the saddle point approximation. The proposed log-LF is easily constructed using component-wise moment generating functions and a simple root-finding algorithm. General gradient calculations were presented to help efficiently solve the resultant optimization problem. Quasi-Newton methods performed well in our numerical experiments.

Attention was paid primarily to the case of exponential clipping, Gaussian noise, rounding, and floating point error which motivated the authors initially and provide a principled example where the distributions involved were easily inferred. Despite our focus on several examples, the method remains widely useful, particularly when noise in the design matrix comes from unwieldy distributions with difficult to handle linear combinations. We envision the proposed method finding use anytime the design matrix is subject to ambiguity.

Chapter 4

Trust Region Methods Using Hermite Interpolation

4.1 Introduction

In this chapter, our goal is to solve the problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \; f(\mathbf{x}), \tag{4.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function. We focus on trust region (TR) methods which are numerical optimization schemes that iteratively solve non-convex problems such as (4.1).

An important consideration for a TR method is the choice of a model function. Linear and quadratic Taylor approximations are popular choices with easy to interpret error bounds and intuitive convergence results. Taylor approximations require access to derivatives, however, which in some cases are unavailable or too costly to compute. In addition, gradient based methods are greedy in the sense that they always step in the optimal **local** direction ignoring previous function values. Second order schemes such as Newton's Method use curvature as well, but still rely on the Hessian (another derivative). We introduce the salient feature of trust region methods in this chapter. Additional details can be found in the comprehensive text by Conn, Gould, and Toint [Conn et al., 2000]

When gradients are difficult to obtain, interpolating polynomials are often used. This is a popular choice in derivative free optimization (DFO) [Larson et al., 2019]. Interpolation involves solving a linear system to fit function values queried at previous iterates. By tracking objective values, the model is aware of the objective landscape, preventing the algorithm from making poor

decisions based solely on local information. Radial basis functions have been used for interpolation problems and show promise for computationally expensive objectives [Björkman and Holmström, 2000, Wild et al., 2008], but we focus on polynomial based interpolation models since the overwhelming majority of DFO solvers use them.

Like other DFO such as the Nelder-Mead Simplex Method [Nelder and Mead, 1965], polynomial based TR interpolation methods suffer from a number of drawbacks. First, they require, many function queries before a useful model is formed. For an n dimensional problem to be uniquely determined, a quadratic interpolation model requires $(1/2)(n^2+3n)$ function evaluations; one constraint for each component of the gradient and $n^2 + (n/2)$ for the degrees of freedom in the symmetric model Hessian (add one more if the model is the model is not centered). Second, solving for the coefficients of the interpolating polynomial in high dimension is costly since matrix inversion scales cubically with the problem dimension. Third, the system to be solved is often poorly conditioned resulting in solutions that are sensitive to rounding error. Perhaps the biggest limitation is that "on serial machines, it is usually not reasonable to try and optimize problems with more than a few tens of variables" [Conn et al., 2009, p. 5]. The authors go on to state that with recent advances, unconstrained problems of several hundred variables can be solved, but the limitation is apparent.

Our task is to devise methods to reduce the number of iterations required for convergence. Given the success of first order methods, it makes intuitive sense that interpolation is cast aside when derivatives are available. On the other hand, there are situations where gradients are available but expensive and can only be probed intermittently. Similarly, we may only have access to partial derivatives with respect to some but not all variables. Under these circumstances, we would like to incorporate derivative information without abandoning the simplicity of an interpolation framework.

Surprisingly, little has been said around Hermite interpolation for trust region methods despite their ubiquity in other branches of applied mathematics. They were incorporated into a hybrid evolutionary algorithm for computationally expensive adjoint solvers in [Ong et al., 2008], but beyond that, interpolation and gradient based trust region methods are mutually exclusive.

Our goal is to fill this gap.

We have two motivating problems in mind. First, we imagine a cost function which depends on two sets of variables. Suppose the objective has a simple dependence on the first set such that derivatives can be calculated analytically, but a prohibitively complex dependence on the second grouping. For illustrative purposes, we consider the forward model

$$\mathbf{f}(\mathbf{p},\mathbf{q}) = \begin{bmatrix} \hat{\mathbf{a}}^T\mathbf{L}(\mathbf{r}_1,\mathbf{p}) \\ \hat{\mathbf{a}}^T\mathbf{L}(\mathbf{r}_2,\mathbf{p}) \\ \vdots \\ \hat{\mathbf{a}}^T\mathbf{L}(\mathbf{r}_M,\mathbf{p}) \end{bmatrix} \mathbf{q} + \|\mathbf{a}\|\mathbf{1}$$

which is expounded on in Chapter 6 (see Eqs. 6.2, 6.3, and 6.5). Using the data measured by sensors located at $\{\mathbf{r}_1,\ \mathbf{r}_2,\ ...,\mathbf{r}_M\}$, we can form a residual sum of squares objective given by $\boldsymbol{\ell}(\mathbf{p},\mathbf{q}) = \|\mathbf{f}(\mathbf{p},\mathbf{q}) - \mathbf{y}\|^2$. The dependence on $\mathbf{q}$ is quadratic making $\nabla_{\mathbf{q}}\boldsymbol{\ell}(\mathbf{p},\mathbf{q})$ trivial to calculate. The dependence on $\mathbf{p}$, however, is very complicated making $\nabla_{\mathbf{p}}\boldsymbol{\ell}(\mathbf{p},\mathbf{q})$ difficult or impossible to compute. We can treat the objective as a black-box function in $\mathbf{p}$. Although we don't have a full gradient, this valuable information should not be ignored and can be incorporated into a trust region model.

In the second, we imagine gradients are only available periodically as might be encountered in a multi-fidelity framework (low fidelity is a function evaluation, high fidelity is a function and gradient evaluation). If a finite difference approximation or automatic differentiation is used, the cost is at best a constant factor more than the objective itself. As a result we we would like to query the gradient infrequently, perhaps once every 10 iterations, to reduce the cost while still including salient information from prior gradients in successive iterations. In this sense, gradients and partial derivatives are treated in a similar fashion as subgradients in bundle methods for convex problems [Mäkelä, 2002].

Despite the simplistic connection between Hermite interpolation and traditional interpolation methods used in TR methods, it does not appear that anyone has successfully employed derivative interpolating trust region methods. We do so in this chapter, the remainder of which introduces

trust region methods in more depth, discusses several models used in the trust region subproblem, and outlines the Hermite interpolating framework. We introduce the minimum norm solution for the TR subproblem along with several variants, discuss how to incorporate gradients/partial derivatives, then explain several issues with the formulation and methods to address them. Finally, we present an algorithm and provide numerical examples from the well known CUTEst test set [Gould et al., 2015] to illustrate the methods merit.

4.2 Background

In this section, we introduce several topics central to our algorithm. We first present trust region (TR) methods in generic terms followed by a discussion of interpolation based approximations and the limitations encountered.

4.2.1 Trust region methods

As stated prior, trust region (TR) methods are numerical optimization schemes that iteratively solve non-convex problems like the one found in (4.1). This is accomplished by forming a **model function** denoted by $m_k : \mathbb{R}^n \to \mathbb{R}$ which approximates the objective, f, in a neighborhood of the k^{th} iterate, $\mathbf{x}_k$. That is, $m_k(\mathbf{s}) \approx f(\mathbf{x}_k + \mathbf{s})$ when the norm of $\mathbf{s}$ is small. This approximation is optimized over a bounded set called the **trust region** on which the model is presumed to be "trust-worthy". TRs are usually chosen to be Euclidean norm-constrained balls with an associated **trust region radius**, δ_k, for the k^{th} iteration. By choosing a simple model, the **trust region subproblem**,

$$\mathbf{s}_k = \underset{\|\mathbf{s}\| \leq \delta_k}{\operatorname{argmin}} \; m_k(\mathbf{s}), \tag{4.2}$$

often reduces to an eigenvalue problem that can be solved efficiently with well known techniques [Conn et al., 2000]. Popular choices for m_k are linear and quadratic Taylor series approximation, i.e., $m_k(\mathbf{s}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{s}$ or $m_k(\mathbf{s}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \mathbf{s} + (1/2)\mathbf{s}^T \nabla^2 f(\mathbf{x}_k)\mathbf{s}$. Once (4.2) has been solved, the new prospective point $\mathbf{x}_+ = \mathbf{x}_k + \mathbf{s}_k$ can be queried. When the estimated decrease in the objective given by the model matches the actual decrease in the objective, we

set $\mathbf{x}_{k+1} = \mathbf{x}_+$, otherwise $\mathbf{x}_{k+1} = \mathbf{x}_k$ and the trust region radius is reduced to give a better approximation. Mathematically, we accept the prospective iterate when ρ defined by

$$\rho_k = \frac{\text{Actual reduction}}{\text{Predicted reduction}} = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_+)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}, \tag{4.3}$$

is larger than a user-specified threshold, i.e., $\rho_k > \eta_1 > 0$. We note that the denominator for $\rho_k \geq 0$ (if zero, no step is taken). Hence the prospective point is rejected if the objective realizes and increase, or too small of a decrease.

If ρ_k is larger than the user specified parameter η_2, we conclude that the model on the trust region approximates the objective well then expand the trust region radius. The standard trust region method is given in Alg. 1

To be useful and guarantee convergence under differentiability conditions, the models used must be *fully-linear* or *fully-quadratic* depending the model's polynomial degree. In essence, this ensures that the models are "about as good" as the corresponding Taylor expansion. The construction of a fully linear/quadratic model requires the maintenance of a *well-poised* geometric set that can be built in a deterministic way [Conn et al., 2008a, Conn et al., 2008b] but is challenging to maintain. Full definitions for the italicized terms can be found in [Conn et al., 2009]. An efficient algorithm to maintain this well-poised set was proposed in [Scheinberg and Toint, 2010]. In practice, users often relax requirements for the geometry improving phase which shows acceptable performance [Fasano et al., 2009].

Algorithm 1: Standard Trust Region

Initialize $0 < \eta_1 \leq \eta_2 < 1$, $\gamma_{\text{inc}} > 1$, $\gamma_{\text{dec}} \in (0, 1)$

Choose initial $\delta_1 > 0$, $\mathbf{x}_0 \in \mathbb{R}^n$.

$\mathbf{s}_k \leftarrow \mathbf{0}, k \leftarrow 0$

while some stopping criterion not satisfied do

Construct model m_k.

Solve $\mathbf{s}_k = \text{argmin}_{\|\mathbf{s}\| \leq \delta_k} m_k(\mathbf{s})$.

Compute $\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}$.

if $\rho_k > \eta_1$ **(successful iteration) then**

$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{s}_k$.

if $\rho_k > \eta_2$ **(very successful iteration) then**

$\delta_{k+1} \leftarrow \gamma_{\text{inc}} \delta_k$.

end

else

$\delta_{k+1} \leftarrow \gamma_{\text{dec}} \delta_k$.

$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$.

end

$k \leftarrow k + 1$.

end

4.2.2 Interpolation Based Models

In this chapter, we are interested in functions with difficult to compute derivatives within the trust region framework. We'd like to use past function values at different points to fit a model. To this end, we are given a set of vectors, $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(k)}\}$, in $\mathbb{R}^n$ and corresponding function values, $\{f(\mathbf{x}^{(1)}), f(\mathbf{x}^{(2)}), ..., f(\mathbf{x}^{(k)})\}$, we want to find a polynomial such that $m_k(\mathbf{x}^{(i)} - \mathbf{x}^{(k)}) = f(\mathbf{x}^{(i)})$ for all $i \in \{1, ..., k\}$. This is nothing more than a standard interpolation problem. After choosing a polynomial basis, one can form a Vandermonde matrix and solve a linear system of equations to find the polynomial coefficients that satisfy the interpolating conditions.

We focus on quadratic approximations using the monomial basis, i.e.,

$$\left\{1, x_1,\ x_2,\ ...,\ x_n,\ \frac{1}{2}x_1^2,\ \frac{1}{2}x_2^2,\ ...,\ \frac{1}{2}x_n^2,\ x_1x_2,\ ...,\ x_1x_n,\ x_2x_3, ...,\ x_3x_n, ...,\ x_{n-1}x_n\right\}, \tag{4.4}$$

for $\mathbf{x} = [x_1,\ x_2,\ , ...,\ x_n]^T$. Note that the basis elements can be split into constant, linear, and quadratic elements. The quadratic elements can be further refined into diagonal and cross terms. We note that the cross terms are not scaled by $1/2$ since we implicitly count them twice to account for the symmetry of the Hessian. Hence, any quadratic polynomial can be written

$$p(\mathbf{x}) = c_0 + \sum_{i=1}^{n}\left(g_i x_i + \frac{1}{2}h_{ii}x_i^2\right) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} h_{ij}x_i x_j. \tag{4.5}$$

For illustrative purposes, the Vandermonde system for an interpolating quadratic in $n = 2$ is

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \frac{1}{2}(x_1^{(1)})^2 & \frac{1}{2}(x_2^{(1)})^2 & x_1^{(1)}x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \frac{1}{2}(x_1^{(2)})^2 & \frac{1}{2}(x_2^{(2)})^2 & x_1^{(2)}x_2^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(6)} & x_2^{(6)} & \frac{1}{2}(x_1^{(6)})^2 & \frac{1}{2}(x_2^{(6)})^2 & x_1^{(6)}x_2^{(6)} \end{bmatrix} \begin{bmatrix} c_0 \\ g_1 \\ g_2 \\ h_{11} \\ h_{22} \\ h_{12} \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}^{(1)}) \\ f(\mathbf{x}^{(2)}) \\ \vdots \\ f(\mathbf{x}^{(6)}) \end{bmatrix}. \tag{4.6}$$

Since we know the matrix (by virtue of accessing the query points) and the right hand side composed of function values, we can invert the system to find the interpolating coefficients. The number of equations required to uniquely solve for a quadratic interpolating polynomial in $\mathbb{R}^n$ is $(1/2)(n^2+3n+2)$; 1 for a constant, n for linear terms, n for quadratic diagonal terms, and $(n^2-n)/2$ for cross quadratic terms due to symmetry in the Hessian. The interpolating polynomial can be written in vector/matrix notation as

$$p(\mathbf{x}) = c_0 + \mathbf{g}^{\mathrm{T}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x}, \quad \text{where} \quad \mathbf{g} = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \quad \text{and} \quad \mathbf{H} = \begin{pmatrix} h_{11} & h_{12} \\ h_{12} & h_{22} \end{pmatrix}. \tag{4.7}$$

To be useful and guarantee convergence under differentiability conditions, the models used must be *fully-linear* or *fully-quadratic* depending the model's polynomial degree. In essence, this

ensures that the models are "about as good" as the corresponding Taylor expansion. The construction of a fully linear/quadratic model requires the maintenance of a *well-poised* geometric set that can be built in a deterministic way [Conn et al., 2008a, Conn et al., 2008b] but is challenging to maintain. Full definitions for the italicized terms can be found in [Conn et al., 2009]. An efficient algorithm to maintain this well-poised set was proposed in [Scheinberg and Toint, 2010]. In practice, users often relax requirements for the geometry improving phase which shows acceptable performance [Fasano et al., 2009].

The strength of interpolation for model formation relies on its simplicity and ease of implementation. Furthermore, previous function values are naturally incorporated into the model allowing for approximations that capture some global behavior. Since basic interpolation methods ignore derivative information, the model ignores valuable local information when its available. By not exploiting all available information, convergence can suffer.

4.2.3 Hermite Interpolation

Hermite interpolation uses function and derivative values to fit a polynomial approximation. Working in one-dimension for the moment, suppose we have data $\{f(x^{(1)}), f'(x^{(1)}), f(x^{(2)}), f'(x^{(2)})\}$ and want to fit a cubic polynomial. To find the four unknowns in the interpolating polynomial, $p(x) = a + bx + cx^2 + dx^3$, we use the conditions $p(x^{(1)}) = f(x^{(1)})$, $p(x^{(2)}) = f(x^{(2)})$, $p'(x^{(1)}) = f'(x^{(1)})$, and $p'(x^{(2)}) = f'(x^{(2)})$. This gives rise to the general Vandemonde system

$$\begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & (x^{(1)})^3 \\ 1 & x^{(2)} & (x^{(2)})^2 & (x^{(2)})^3 \\ 0 & 1 & 2x^{(1)} & 3(x^{(1)})^2 \\ 0 & 1 & 2x^{(2)} & 3(x^{(2)})^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} f(x^{(1)}) \\ f(x^{(2)}) \\ f'(x^{(1)}) \\ f'(x^{(2)}). \end{pmatrix} \tag{4.8}$$

Similarly, we can fit derivatives for higher dimensions as well. For each partial derivative, we obtain one additional constraint. Hence, if we are interpolating a quadratic over n dimensions, interpolating a single gradient will constrain a total of n equations.

Notation: Throughout this chapter, lowercase (uppercase) bold-faced letters or symbols denote vectors (matrices), i.e., $\mathbf{x}$ ($\mathbf{X}$). Non-bolded symbols represent scalars. The i^{th} vector from a list of vectors will be noted by $\mathbf{x}^{(i)}$. The j^{th} element from a vector $\mathbf{v}$ is given by $\mathbf{v}_j$ or v_j when the meaning is clear. The ambient problem is n-dimensional and the number of points used for interpolation is m. The element in the i^{th} row and j^{th} column of matrix $\mathbf{M}$ is given by $\mathbf{M}_{ij}$. We borrow MATLAB style notation so that the k^{th} column of matrix $\mathbf{M}$ is denoted by $\mathbf{M}_{:,k}$ and $\mathbf{M}_{k,:}$ gives the k^{th} row.

The Hadamard or element-wise product between matrices is denoted by $\odot$. Similarly, $^{2.}$ denotes the element-wise square of a matrix, i.e., $\mathbf{M}^{2.} = \mathbf{M} \odot \mathbf{M}$. A length k vector of ones is denoted by $\mathbf{1}_k$.

A row from the Vandermonde matrix using point $\mathbf{x}$ in the monomial basis as given in (4.4) is denoted by $\boldsymbol{\phi}(\mathbf{x})$, in particular

$$\mathbf{V} = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{x}^{(1)})^{\mathrm{T}} \\ \boldsymbol{\phi}(\mathbf{x}^{(2)})^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}^{(n)})^{\mathrm{T}} \end{pmatrix} \tag{4.9}$$

From this point forward, we focus exclusively on polynomials **without a constant offset** which is equivalent to $c_0 = 0$ in (4.7). This can easily be accomplished at each step of our trust region algorithm by shifting the origin for the latest iterate (or best solution) to the origin. Hence, all distances are measured from our current location and the function values gives difference from best solution so far. We assume that all function values are centered to an appropriate value and only make note of it when necessary for clarity. It is sometimes helpful to split rows $\boldsymbol{\phi}(\mathbf{x})$ into linear and quadratic components (quadratic part denoted by $\boldsymbol{\psi}$). That is $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} = [\mathbf{x}^{\mathrm{T}}, \boldsymbol{\psi}(\mathbf{x})^{\mathrm{T}}]$ and

$$\mathbf{V} = \begin{pmatrix} (\mathbf{x}^{(1)})^{\mathrm{T}} & \boldsymbol{\psi}(\mathbf{x}^{(1)})^{\mathrm{T}} \\ (\mathbf{x}^{(2)})^{\mathrm{T}} & \boldsymbol{\psi}(\mathbf{x}^{(2)})^{\mathrm{T}} \\ \vdots & \vdots \\ (\mathbf{x}^{(n)})^{\mathrm{T}} & \boldsymbol{\psi}(\mathbf{x}^{(n)})^{\mathrm{T}} \end{pmatrix} = [\mathbf{L} \quad \mathbf{Q}] \tag{4.10}$$

where $\mathbf{L}$ and $\mathbf{Q}$ represent the linear and quadratic blocks of the Vandermonde. We may periodically use the shorthand $\mathbf{V} = \boldsymbol{\phi}(\mathbf{X})$ if $\mathbf{X}$ is composed of the points we use to form the Vandermonde, i.e., $\mathbf{X}^{\mathrm{T}} = [\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}]$. Since we are interested in using derivative based interpolation, we denote the k^{th} partial derivative of $\boldsymbol{\phi}(\mathbf{x})$ by $\boldsymbol{\phi}_k(\mathbf{x}) = (\partial/\partial x_k)\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^{(n^2+3n)/2}$. The matrix $\mathbf{D}_j = \nabla\boldsymbol{\psi}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^{(j)}}$ is the Jacobian of $\boldsymbol{\psi}(\mathbf{x}^{(j)})$,

$$\mathbf{D}_j = \begin{pmatrix} \boldsymbol{\psi}_1(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \boldsymbol{\psi}_2(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\psi}_n(\mathbf{x}^{(j)})^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{n\times(n^2+n)/2}. \tag{4.11}$$

Similarly, the Jacobian of the $\boldsymbol{\phi}(\mathbf{x})$ at $\mathbf{x}^{(j)}$ is given by

$$\begin{pmatrix} \mathbf{e}_1^{\mathrm{T}} & \boldsymbol{\psi}_1(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \mathbf{e}_2^{\mathrm{T}} & \boldsymbol{\psi}_2(\mathbf{x}^{(j)})^{\mathrm{T}} \\ \vdots & \vdots \\ \mathbf{e}_n^{\mathrm{T}} & \boldsymbol{\psi}_n(\mathbf{x}^{(j)})^{\mathrm{T}} \end{pmatrix} = [\mathbf{I} \quad \mathbf{D}_j] \in \mathbb{R}^{n\times(n^2+3n)/2}, \tag{4.12}$$

where $\mathbf{e}_k$ is the k^{th} standard basis element.

4.3 Interpolating High-Dimensional Polynomials with a Minimum Norm Solution

We considered cases where the gradient is costly or difficult to compute. It is often the case that the function itself is expensive to compute. Since standard interpolation based TR methods require $O(n^2)$ interpolating conditions to uniquely determine polynomial coefficients, a prohibitively large number of function evaluations are required before the first iteration is completed.

To allow the method to proceed in the underdetermined case, i.e., when $n < m < \frac{n^2+3n}{2}$, we turn our attention to a standard technique which is to find the **minimum norm solution** (MNS). The MNS penalizes the size of coefficients that compose the Hessian of the polynomial model. Mathematically, this is cast as

$$\begin{aligned} \min_{\mathbf{g},\mathbf{h}} \quad & \frac{1}{2}\|\mathbf{h}\|^2 \\ \text{subject to} \quad & \mathbf{L}\mathbf{g} + \mathbf{Q}\mathbf{h} = \mathbf{f}. \end{aligned} \tag{4.13}$$

For the point about which the data and polynomial have been centered and letting $\mathbf{g} = \nabla p(\mathbf{x})$ and $\mathbf{H} = \nabla^2 p(\mathbf{x})$, we have

$$p(\mathbf{x}) = \mathbf{g}^{\mathrm{T}}\mathbf{x} + (1/2)\mathbf{x}^{\mathrm{T}}\ \mathbf{H}\ \mathbf{x} \quad \Longleftrightarrow \quad p(\mathbf{x}) = \mathbf{g}^{\mathrm{T}}\mathbf{x} + \mathbf{h}^{\mathrm{T}}\boldsymbol{\psi}(\mathbf{x}). \tag{4.14}$$

The vector $\mathbf{h}$ encodes a permuted and vectorized Hessian for the quadratic polynomial model. To make this more precise and ease the exposition of proofs later, we let $r(k)$ and $c(k)$ be mappings from the k^{th} entry of $\mathbf{h}$ to the corresponding row and column of the polynomial Hessian. To illustrate for a 4d polynomial:

$$\begin{aligned} \mathbf{h} \ &= \ [H_{11},\ H_{22},\ H_{33},\ H_{44},\ H_{12},\ H_{13},\ H_{14},\ H_{23},\ H_{34}] \\ &\qquad\qquad \Updownarrow \\ \mathbf{H} \ &= \ \begin{pmatrix} h_1 & h_5 & h_6 & h_7 \\ h_5 & h_2 & h_8 & h_9 \\ h_6 & h_8 & h_3 & h_{10} \\ h_7 & h_9 & h_{10} & h_4 \end{pmatrix} \end{aligned}$$

Note that the factor of $1/2$ appearing in the left equality of (4.14) is absorbed in the vector $\boldsymbol{\psi}(\mathbf{x})$.

Hence, the MNS returns the model gradient and smallest Hessian that satisfy the interpolating conditions, i.e., the quadratic with the least curvature. Given a choice, the MNS return the model closest to linear as possible. This is intuitively appealing as linear models are simple and generally typically prefer simple models over complex ones when possible.

Practically, this problem can be solved by forming the KKT system. This is well known, but we prove it below since it is the basis for more complicated variants arising later.

Proposition 1. *The solution to* (4.13) *is given by solving the following system of equations:*

$$\begin{pmatrix} \mathbf{0} & \mathbf{L}^T \\ \mathbf{L} & \mathbf{Q}\mathbf{Q}^T \end{pmatrix} \begin{pmatrix} \mathbf{g} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{f} \end{pmatrix}, \tag{4.15}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ *are Lagrange multipliers and* $\mathbf{h} = \mathbf{Q}^T\boldsymbol{\lambda}$.

Proof. Begin by forming the Lagrangian:

$$\mathcal{L}(\boldsymbol{\lambda}, \mathbf{g}, \mathbf{h}) = \frac{1}{2}\mathbf{h}^{\mathrm{T}}\mathbf{h} + \boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{L}\mathbf{g} + \mathbf{Q}\mathbf{h} - \mathbf{f}).$$

Differentiating and equating to zero, we have

$$\nabla_{\boldsymbol{\ell}}\mathcal{L} = \mathbf{L}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{0}, \tag{4.16}$$

$$\nabla_{\mathbf{h}}\mathcal{L} = \mathbf{h} - \mathbf{Q}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{0}, \tag{4.17}$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \mathbf{L}\mathbf{g} + \mathbf{Q}\mathbf{h} - \mathbf{f} = \mathbf{0}. \tag{4.18}$$

Multiplying (4.17) by $\mathbf{Q}$, rearranging, and plugging into (4.18) gives the KKT system when stacked with (4.16). Since the problem is convex and has a differentiable objective and constraint, the solution to the KKT system is necessary and sufficient for optimality [Boyd et al., 2004]. □

Although it is useful for expository purposes, forming $\mathbf{Q}$ naively is computationally expensive task as is the construction of $\mathbf{Q}\mathbf{Q}^{\mathrm{T}}$. This can be avoided by using the following known result which we generalize later.

Lemma 1. *Let* $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}]^T$ *where* $\mathbf{x}^{(i)}$ *is the* i^{th} *interpolating point such that* $\mathbf{V} = \boldsymbol{\phi}(\mathbf{X})$. *Then*

$$\mathbf{Q}\mathbf{Q}^T = \frac{1}{2}(\mathbf{X}\mathbf{X}^T)^{2\cdot} - \frac{1}{4}\mathbf{X}^{2\cdot}(\mathbf{X}^{2\cdot})^T, \tag{4.19}$$

where $\mathbf{M}^{2\cdot}$ *denotes the element-wise square of a matrix* $\mathbf{M}$.

Proof. Consider the i, j element of $\mathbf{Q}\mathbf{Q}^{\mathrm{T}}$, i.e., $(\mathbf{Q}\mathbf{Q}^{\mathrm{T}})_{ij}$. We have

$$(\mathbf{Q}\mathbf{Q}^{\mathrm{T}})_{ij} = \sum_{k=1}^{n} \frac{1}{4}(x_k^{(i)})^2(x_k^{(j)})^2 + \sum_{k=n+1}^{(n^2+n)/2} x_{r(k)}^{(i)} x_{c(k)}^{(i)} x_{r(k)}^{(j)} x_{c(k)}^{(j)}, \tag{4.20}$$

where $r(k)$ and $c(k)$ are mappings from the columns $\mathbf{h}$ to the entries of $\mathbf{H}$. Note that the first summation reduces to

$$\sum_{k=1}^{n} \frac{1}{4}(x_k^{(i)})^2(x_k^{(j)})^2 = \frac{1}{4}\left((\mathbf{X}^{2\cdot})(\mathbf{X}^{2\cdot})^{\mathrm{T}}\right)_{ij}. \tag{4.21}$$

For the second sum, we group the $r(k)$ and $c(k)$ factors such that

$$\begin{aligned}
\sum_{k=n+1}^{(n^2+n)/2} x_{r(k)}^{(i)} x_{r(k)}^{(j)} x_{c(k)}^{(i)} x_{c(k)}^{(j)} &= \sum_{p=1}^{n} \sum_{\substack{q=1 \\ q\neq p}}^{n} x_p^{(i)} x_p^{(j)} x_q^{(i)} x_q^{(j)} \\
&= -\sum_{k=1}^{n} \frac{1}{2}(x_k^{(i)})^2(x_k^{(j)})^2 + \sum_{p=1}^{n}\sum_{q=1}^{n} x_p^{(i)} x_p^{(j)} x_q^{(i)} x_q^{(j)} \\
&= -\frac{1}{2}\left((\mathbf{X}^{2\cdot})(\mathbf{X}^{2\cdot})^{\mathrm{T}}\right)_{ij} + \sum_{p=1}^{n} x_p^{(i)} x_p^{(j)} \sum_{q=1}^{n} x_q^{(i)} x_q^{(j)} \qquad (4.22) \\
&= -\frac{1}{2}\left((\mathbf{X}^{2\cdot})(\mathbf{X}^{2\cdot})^{\mathrm{T}}\right)_{ij} + \frac{1}{2}(\mathbf{X}\mathbf{X}^{\mathrm{T}})_{ij}(\mathbf{X}\mathbf{X}^{\mathrm{T}})_{ij} \\
\text{(using (4.21))} \quad \Longrightarrow \quad (\mathbf{Q}\mathbf{Q}^{\mathrm{T}})_{ij} &= \frac{1}{2}(\mathbf{X}\mathbf{X}^{\mathrm{T}})_{ij}^2 - \frac{1}{4}\left((\mathbf{X}^{2\cdot})(\mathbf{X}^{2\cdot})^{\mathrm{T}}\right)_{ij}. \qquad (4.23)
\end{aligned}$$

Hence, $\mathbf{Q}\mathbf{Q}^{\mathrm{T}} = \frac{1}{2}(\mathbf{X}\mathbf{X}^{\mathrm{T}})^{2\cdot} - \frac{1}{4}\mathbf{X}^{2\cdot}(\mathbf{X}^{2\cdot})^{\mathrm{T}}$. □

This plays an important role in computation. Naively, the creation of $\mathbf{Q}$ is $O(mn^2)$ where m is the number of data points and n is the problem dimension. Once $\mathbf{Q}$ is complete, forming the Gram matrix requires $O(m^2n^2)$. Alternatively, the formation of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{X}^{2\cdot}(\mathbf{X}^{2\cdot})^{\mathrm{T}}$ are both $O(m^2n)$. The element-wise squaring is $O(mn)$ and therefore is dominated by Gram matrix formation. Proposition 1 reduces the computation by an order of $O(n)$.

Similarly, when it comes to forming $\mathbf{h}$, it is best best to avoid the $O(mn^2)$ operation of constructing $\mathbf{Q}$ then applying it to $\boldsymbol{\lambda}$. Instead, we recognize that our $\mathbf{h}$ encodes information for our model Hessian. With this in mind, we have the following result that reduces the complexity of forming $\mathbf{h}$ from $O(m^2n^2)$ to $O(mn^2)$ for constructing $\mathbf{H}$.

Lemma 2. *Suppose that* $p(\mathbf{x}) = \boldsymbol{\ell}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} = \mathbf{g}^T\mathbf{x} + \mathbf{h}^T\boldsymbol{\psi}(\mathbf{x})$ *where* $\mathbf{g}$ *and* $\mathbf{h}$ *solve* (4.13). *Then*

$$\mathbf{H} = \mathbf{M} - \frac{1}{2}\text{diag}(\text{diag}(\mathbf{M})), \tag{4.24}$$

where $\mathbf{M} = \mathbf{X}^T\text{diag}(\boldsymbol{\lambda})\mathbf{X}$.

Proof. We consider the k^{th} element of $\mathbf{h}$. Since k maps to a column and row of and square matrix via $c(k)$ and $r(k)$, respectively, then

$$\mathbf{H}_{r(k),c(k)} = q_k = (\mathbf{Q}_{:,k})^{\mathrm{T}}\boldsymbol{\lambda}.$$

When $k \leq n$, this corresponds to diagonal elements with the indices matching the entry, i.e.,

$$\mathbf{H}_{k,k} = \mathbf{h}_k = \frac{1}{2}\sum_{i=1}^{m}\lambda_i(x_k^{(i)})^2 = \frac{1}{2}(\mathbf{X}_{:,k})^{\mathrm{T}}\,\text{diag}(\boldsymbol{\lambda})\,\mathbf{X}_{:,k} \quad \text{for} \quad k \leq n.$$

Similarly, for cross terms, we have

$$\mathbf{H}_{r(k),c(k)} = \sum_{i=1}^{m}\lambda_i x_{r(k)}^{(i)} x_{c(k)}^{(i)} = (\mathbf{X}_{:,r(k)})^{\mathrm{T}}\,\text{diag}(\boldsymbol{\lambda})\,\mathbf{X}_{:,c(k)} \quad \text{for} \quad r(k) \neq c(k) \quad \text{for} \quad k > n.$$

Looping over k, we can construct the upper triangular portion of $\mathbf{H}$. By symmetry of the Hessian, we get the lower off diagonal triangular matrix. This can be written using the expression in (4.24). □

4.3.1 Other Regularization Schemes

The minimum norm solution penalizes the Frobenius norm of the Hessian. Given some prior knowledge of the Hessian, it is possible to penalize its distance from a model Hessian that exhibits known performance. Letting $\mathbf{h}_0$ be the vectorized encoding of $\mathbf{H}_0$, we solve

$$\begin{aligned} \min_{\mathbf{g},\mathbf{h}} \quad & \frac{1}{2}\|\mathbf{h} - \mathbf{h}_0\|^2 \\ \text{subject to} \quad & \mathbf{L}\mathbf{g} + \mathbf{Q}\mathbf{h} = \mathbf{f}. \end{aligned} \tag{4.25}$$

Corollary 2. *The solution to* (4.25) *is given by solving the following system of equations:*

$$\begin{pmatrix} \mathbf{0} & \mathbf{L}^T \\ \mathbf{L} & \mathbf{Q}\mathbf{Q}^T \end{pmatrix} \begin{pmatrix} \mathbf{g} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{f} - \mathbf{Q}\mathbf{h}_0 \end{pmatrix}, \tag{4.26}$$

where $\boldsymbol{\lambda}$ *are Lagrange multipliers and* $\mathbf{h} = \mathbf{h}_0 + \mathbf{Q}^T\boldsymbol{\lambda}$.

Proof. Follows as in Prop. 1 with $\mathbf{h} \leftarrow \mathbf{h} - \mathbf{h}_0$ and isolating constant terms in the KKT system. □

4.4 Interpolating Function and Derivative Values

In some cases, we have access to certain derivatives and would like to incorporate these values into our interpolating polynomial model. Supposing that we have derivatives at points $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(d)}\}$, we can write this as a minimum norm problem given by

$$\begin{aligned} \min_{\mathbf{g},\mathbf{h}} \quad & \frac{1}{2}\|\mathbf{h}\|^2 \\ \text{subject to} \quad & \begin{pmatrix} \mathbf{L} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \end{pmatrix} \mathbf{g} + \begin{pmatrix} \mathbf{Q} \\ \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_d \end{pmatrix} \mathbf{h} = \begin{pmatrix} \mathbf{f} \\ \nabla f_1 \\ \vdots \\ \nabla f_d \end{pmatrix}. \end{aligned} \tag{4.27}$$

where $\nabla f_j = \nabla f(\mathbf{x}^{(j)})$. Similar to Prop. 1 and Cor. 2, we have the following novel result

Proposition 2. *When feasible, the solution to* (4.27) *is given by solving the following system of equations:*

$$\begin{pmatrix} \mathbf{0} & \mathbf{L}^T & \mathbf{I}^T & \dots & \mathbf{I}^T \\ \mathbf{L} & \mathbf{Q}\mathbf{Q}^T & \mathbf{Q}\mathbf{D}_1^T & \cdots & \mathbf{Q}\mathbf{D}_d^T \\ \mathbf{I} & \mathbf{D}_1\mathbf{Q}^T & \mathbf{D}_1\mathbf{D}_1^T & \dots & \mathbf{D}_1\mathbf{D}_d^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{I} & \mathbf{D}_d\mathbf{Q}^T & \mathbf{D}_d\mathbf{D}_1^T & \cdots & \mathbf{D}_d\mathbf{D}_d^T \end{pmatrix} \begin{pmatrix} \mathbf{g} \\ \boldsymbol{\lambda}_0 \\ \boldsymbol{\lambda}_1 \\ \vdots \\ \boldsymbol{\lambda}_d \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{f} \\ \nabla f_1 \\ \vdots \\ \nabla f_d \end{pmatrix}, \tag{4.28}$$

where $\boldsymbol{\lambda}_0$ *are Lagrange multipliers associated with the function interpolating constraints and*

$\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_d$ *are the multipliers for their respective gradient conditions. Furthermore,*

$$\mathbf{h} = \begin{pmatrix} \mathbf{Q}^T & \mathbf{D}_1^T & \ldots & \mathbf{D}_d^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda}_0 \\ \boldsymbol{\lambda}_1 \\ \vdots \\ \boldsymbol{\lambda}_d \end{pmatrix} = \mathbf{Q}^T\boldsymbol{\lambda}_0 + \sum_{k=1}^{d} \mathbf{D}_k^T\boldsymbol{\lambda}_k. \tag{4.29}$$

Proof. Follows as in Prop. 1 with $\mathbf{L}^{\mathrm{T}} \leftarrow [\mathbf{L}^{\mathrm{T}}, \mathbf{I}, \ldots, \mathbf{I}]$, $\mathbf{Q}^{\mathrm{T}} \leftarrow [\mathbf{Q}^{\mathrm{T}}, \mathbf{D}_1^{\mathrm{T}}, \ldots, \mathbf{D}_d^{\mathrm{T}}]$, and $\mathbf{f}^{\mathrm{T}} \leftarrow [\mathbf{f}^{\mathrm{T}}, \nabla f_1^{\mathrm{T}}, \ldots, \nabla f_d^{\mathrm{T}}]$. □

We add the feasibility caveat to the proposition since there are many cases where the system is infeasible. We discuss more in later sections.

As before it is costly compute $\mathbf{Q}\mathbf{D}_j^{\mathrm{T}}$. Since $\mathbf{Q} \in \mathbb{R}^{m\times(n^2+n)/2}$ and $\mathbf{D}_j$ belongs to $\mathbb{R}^{n\times(n^2+n)/2}$, then mn inner products of cost $O(n^2)$ must be taken making it an $O(mn^3)$ operation. Using similar tricks as before, we can form the $\mathbf{Q}\mathbf{D}_j^{\mathrm{T}}$ and $\mathbf{D}_i\mathbf{D}_j^{\mathrm{T}}$ matrices for less. The following novel Lemmas show that $\mathbf{Q}\mathbf{D}_i^{\mathrm{T}}$ and $\mathbf{D}_i\mathbf{D}_j^{\mathrm{T}}$ can be formed for $O(mn)$ and $O(n^2)$, respectively.

Lemma 3. *Let* $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}]^T$ *where* $\mathbf{x}^{(i)}$ *is the* i^{th} *interpolating point and* $\mathbf{D}_p$ *is the Jacobian of* $\boldsymbol{\psi}(\mathbf{x})$ *at* $\mathbf{x}^{(p)}$*, then*

$$\mathbf{Q}\mathbf{D}_p^T = \mathbf{X} \odot \left(\mathbf{X}(\mathbf{x}^{(p)}\mathbf{1}_n^T)\right) - \frac{1}{2}\mathbf{X}^{2\cdot} \odot \left(\mathbf{1}_m(\mathbf{x}^{(p)})^T\right). \tag{4.30}$$

Proof. We consider the i, j element of $\mathbf{Q}\mathbf{D}_p^{\mathrm{T}}$ which gives

$$(\mathbf{Q}\mathbf{D}_p^{\mathrm{T}})_{ij} = \boldsymbol{\psi}(\mathbf{x}^{(i)})^{\mathrm{T}}\boldsymbol{\psi}_j(\mathbf{x}^{(p)}) = \frac{1}{2}(x_j^{(i)})^2 x_j^{(p)} + \sum_{\substack{k=1 \\ k\neq j}}^{n} x_k^{(i)} x_j^{(i)} x_k^{(p)} = -\frac{1}{2}(x_j^{(i)})^2 x_j^{(p)} + x_j^{(i)} \sum_{k=1}^{n} x_k^{(i)} x_k^{(p)}$$

since only elements with of $\boldsymbol{\psi}$ with the j^{th} component represented will appear. We note that the last summation is just the inner product between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(p)}$, hence

$$(\mathbf{Q}\mathbf{D}_p^{\mathrm{T}})_{ij} = -\frac{1}{2}(x_j^{(i)})^2 x_j^{(p)} + x_j^{(i)}(\mathbf{x}^{(i)})^{\mathrm{T}}\mathbf{x}^{(p)}.$$

This can be written in vector notation as

$$\mathbf{Q}\mathbf{D}_p^{\mathrm{T}} = \mathbf{X} \odot \mathbf{X}(\mathbf{x}^{(p)}\mathbf{1}_n^{\mathrm{T}}) - \frac{1}{2}\mathbf{X}^{2\cdot} \odot \mathbf{1}_m(\mathbf{x}^{(p)})^{\mathrm{T}},$$

which is the desire result. □

We also require a fast representation for constructing the matrices $\mathbf{D}_p\mathbf{D}_q$. The following Lemma reduces the complexity from $O(n^4)$ to $O(n^2)$.

Lemma 4. *Let* $\mathbf{D}_p$ *and* $\mathbf{D}_q$ *be the Jacobians of* $\boldsymbol{\psi}(\mathbf{x})$ *at* $\mathbf{x}^{(p)}$ *and* $\mathbf{x}^{(q)}$*, respectively. The equality holds*

$$\mathbf{D}_p\mathbf{D}_q^T = \mathbf{x}^{(q)}(\mathbf{x}^{(p)})^T + (\mathbf{x}^{(p)})^T\mathbf{x}^{(q)}\mathbf{I} - \mathrm{diag}(\mathbf{x}^{(p)} \odot \mathbf{x}^{(q)}). \tag{4.31}$$

Proof. Probing the i, j element gives

$$(\mathbf{D}_p\mathbf{D}_q^{\mathrm{T}})_{ij} = \boldsymbol{\psi}_i(\mathbf{x}^{(p)})^{\mathrm{T}}\boldsymbol{\psi}_j(\mathbf{x}^{(q)}),$$

where $\boldsymbol{\psi}_i$ denotes the i^{th} derivative of $\boldsymbol{\psi}$. Note that the only non-zero entries of $\boldsymbol{\psi}_i$ are the elements of $\boldsymbol{\psi}$ where $\mathbf{x}_i$ is present. When $i = j$, we have

$$(\mathbf{D}_p\mathbf{D}_q^{\mathrm{T}})_{ii} = \sum_{k=1}^{n} x_k^{(p)}x_k^{(q)} = (\mathbf{x}^{(q)})^{\mathrm{T}}\mathbf{x}^{(p)}.$$

When $i \neq j$, the only element that is non-zero for both $\boldsymbol{\psi}_i(\mathbf{x}^{(p)})$ and $\boldsymbol{\psi}_j(\mathbf{x}^{(q)})$ is the ij cross-term. Hence

$$(\mathbf{D}_p\mathbf{D}_q^{\mathrm{T}})_{ij} = x_j^{(p)}x_i^{(q)}.$$

Putting the cases together yields

$$(\mathbf{D}_p\mathbf{D}_q^{\mathrm{T}})_{ij} = \begin{cases} x_i^{(q)}x_j^{(p)} & \text{if } i \neq j, \\ (\mathbf{x}^{(p)})^{\mathrm{T}}\mathbf{x}^{(q)} & \text{if } i = j. \end{cases}$$

Our final result is given when we rewrite in vector notation. We note that subtracting the diagonal term is to offset double counting that occurs along the diagonal for the first two terms. □

Having derived a computationally efficient method for computing the blocks of the KKT matrix, all that remains is to find a inexpensive way to compute the Hessian.

Theorem 2. *Suppose that* $p(\mathbf{x}) = \mathbf{g}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} = \mathbf{g}^T\mathbf{x} + \mathbf{h}^T\boldsymbol{\psi}(\mathbf{x})$ *where* $\mathbf{g}$ *and* $\mathbf{h}$ *solve* (4.27) *and* $\boldsymbol{\lambda}_i$ *for* $i \in \{0, ..., d\}$ *are the corresponding multipliers. Then*

$$\mathbf{H} = \mathbf{M} - \frac{1}{2}\mathrm{diag}(\mathrm{diag}(\mathbf{M})) + \sum_{k=1}^{d}\left\{\boldsymbol{\lambda}_k(\mathbf{x}^{(k)})^T + \mathbf{x}^{(k)}\boldsymbol{\lambda}_k^T - \mathrm{diag}(\boldsymbol{\lambda}_k \odot \mathbf{x}^{(k)})\right\}, \tag{4.32}$$

where $\mathbf{M} = \mathbf{X}^T\mathrm{diag}(\boldsymbol{\lambda}_0)\mathbf{X}$.

Proof. Using (4.29), we see that $\mathbf{h} = \mathbf{Q}^{\mathrm{T}}\boldsymbol{\lambda}_0 + \sum_{i=1}^{d} \mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i$. From Lemma 2, we know $\mathbf{H}_0 = \mathbf{Q}^{\mathrm{T}}\boldsymbol{\lambda}_0 = \mathbf{M} - \frac{1}{2}\mathrm{diag}(\mathrm{diag}(\mathbf{M}))$. We now focus on $\mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i$. This can be written as

$$\mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i = \sum_{j=1}^{n} \boldsymbol{\psi}_j(\mathbf{x}^{(i)})(\boldsymbol{\lambda}_i)_j.$$

Focusing on the k^{th} element of $\mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i$, we know that if $r(k) = c(k)$, i.e., a diagonal element, then $(\mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i)_k = (\boldsymbol{\lambda}_i)_{r(k)} x_{r(k)}^{(i)}$. If $r(k) \neq c(k)$, then by direction computation $(\mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i)_k = (\boldsymbol{\lambda}_i)_{c(k)} x_{r(k)}^{(i)} + (\boldsymbol{\lambda}_i)_{r(k)} x_{c(k)}^{(i)}$. Forming the symmetric matrix $\mathbf{H}_i$ from the vector $\mathbf{D}_i^{\mathrm{T}}\boldsymbol{\lambda}_i$, we have

$$\mathbf{H}_i = \begin{pmatrix} (\boldsymbol{\lambda}_i)_1 x_1^{(i)} & (\boldsymbol{\lambda}_i)_1 x_2^{(i)} + (\boldsymbol{\lambda}_i)_2 x_1^{(i)} & \cdots & (\boldsymbol{\lambda}_i)_1 x_n^{(i)} + (\boldsymbol{\lambda}_i)_n x_1^{(i)} \\ (\boldsymbol{\lambda}_i)_1 x_2^{(i)} + (\boldsymbol{\lambda}_i)_2 x_1^{(i)} & (\boldsymbol{\lambda}_i)_2 x_2^{(i)} & \cdots & (\boldsymbol{\lambda}_i)_2 x_n^{(i)} + (\boldsymbol{\lambda}_i)_n x_2^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ (\boldsymbol{\lambda}_i)_1 x_n^{(i)} + (\boldsymbol{\lambda}_i)_n x_1^{(i)} & (\boldsymbol{\lambda}_i)_2 x_n^{(i)} + (\boldsymbol{\lambda}_i)_n x_2^{(i)} & \cdots & (\boldsymbol{\lambda}_i)_n x_n^{(i)} \end{pmatrix}.$$

This can be rewritten

$$\mathbf{H}_i = \mathbf{x}^{(i)}\boldsymbol{\lambda}_i^{\mathrm{T}} + \boldsymbol{\lambda}_i(\mathbf{x}^{(i)})^{\mathrm{T}} - \mathrm{diag}(\boldsymbol{\lambda}_i \odot \mathbf{x}^{(i)}). \tag{4.33}$$

Summing over all l terms gives the result. □

Once the KKT system has been solved for $\mathbf{g}$ and the multipliers, the complexity to form $\mathbf{H}$ is $O(mn^2)$ as opposed to $O(m^2n^2)$ giving a substantial reduction in cost. We summarize the naive and reduced complexity using Lemmas 1, 3, and 4.

Matrix	Naive	Reduced
$\mathbf{Q}\mathbf{Q}^{\mathrm{T}}$	$O(m^2n^2)$	$O(mn^2)$
$\mathbf{Q}\mathbf{D}_i^{\mathrm{T}}$	$O(mn^3)$	$O(mn)$
$\mathbf{D}_i\mathbf{D}_j^{\mathrm{T}}$	$O(n^4)$	$O(n^2)$
$\mathbf{H}$	$O(m^2n^2)$	$O(mn^2)$

Table 4.1: Summary of complexities to form matrices naively vs. construction via Lemmas 1, 3, and 4. In addition, there is no need to store the $\mathbf{Q}$ or $\mathbf{D}_i$ matrices since they can be constructed directly from $\mathbf{X}$.

4.4.1 Different Regularization Schemes

We can penalize the distance from an approximate Hessian in vector form $\mathbf{h}_0$ with

$$\min_{\mathbf{g},\mathbf{h}} \quad \frac{1}{2}\|\mathbf{h}-\mathbf{h}_0\|^2 \tag{4.34}$$
$$\text{subject to} \quad \begin{pmatrix}\mathbf{L}\\ \mathbf{I}\\ \vdots\\ \mathbf{I}\end{pmatrix}\mathbf{g}+\begin{pmatrix}\mathbf{Q}\\ \mathbf{D}_1\\ \vdots\\ \mathbf{D}_d\end{pmatrix}\mathbf{h}=\begin{pmatrix}\mathbf{f}\\ \nabla f_1\\ \vdots\\ \nabla f_d\end{pmatrix}.$$

The only difference between the solution to (4.25) and (4.34) is for the right hand side.

Proposition 3. *The solution to* (4.34) *is given by solving the following system of equations:*

$$\begin{pmatrix}\mathbf{0} & \mathbf{L}^T & \mathbf{I}^T & \cdots & \mathbf{I}^T\\ \mathbf{L} & \mathbf{Q}\mathbf{Q}^T & \mathbf{Q}\mathbf{D}_1^T & \cdots & \mathbf{Q}\mathbf{D}_d^T\\ \mathbf{I} & \mathbf{D}_1\mathbf{Q}^T & \mathbf{D}_1\mathbf{D}_1^T & \cdots & \mathbf{D}_1\mathbf{D}_d^T\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{I} & \mathbf{D}_d\mathbf{Q}^T & \mathbf{D}_d\mathbf{D}_1^T & \cdots & \mathbf{D}_d\mathbf{D}_d^T\end{pmatrix}\begin{pmatrix}\mathbf{g}\\ \boldsymbol{\lambda}_0\\ \boldsymbol{\lambda}_1\\ \vdots\\ \boldsymbol{\lambda}_d\end{pmatrix}=\begin{pmatrix}\mathbf{0}\\ \mathbf{f}-\mathbf{Q}\mathbf{h}_0\\ \nabla f_1-\mathbf{D}_1\mathbf{h}_0\\ \vdots\\ \nabla f_d-\mathbf{D}_d\mathbf{h}_0\end{pmatrix}, \tag{4.35}$$

where $\boldsymbol{\lambda}_0$ *are Lagrange multipliers associated with the function interpolating constraints and* $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_d$ *are the multipliers for their respective gradient conditions. Furthermore,*

$$\mathbf{h}=\mathbf{h}_0+\mathbf{Q}^T\boldsymbol{\lambda}_0+\sum_{k=1}^{d}\mathbf{D}_k^T\boldsymbol{\lambda}_k. \tag{4.36}$$

Proof. The result follows from Corollary 2 and Proposition 2. □

Once again, we would prefer to work without ever explicitly forming the matrices $\mathbf{Q}$ or $\mathbf{D}_i$. We have the following results that ease the computation of the right hand side.

Proposition 4. *Let* $\mathbf{h}_0$ *be the vector encoding (with a factor of 1/2 multiplied to the diagonal components) of a symmetric matrix* $\mathbf{H}_0$ *(Hessian in our case). Then the* k^{th} *element of* $\mathbf{Q}\mathbf{h}_0$ *is given by*

$$(\mathbf{Q}\mathbf{h}_0)_k=\frac{1}{2}(\mathbf{x}^{(k)})^T\mathbf{H}_0\mathbf{x}^{(k)} \qquad \textit{and} \qquad \mathbf{D}_i\mathbf{h}_0=\mathbf{H}_0\mathbf{x}^{(i)}. \tag{4.37}$$

Proof. Starting with the k^{th} row gives

$$(\mathbf{Q}\mathbf{h}_0)_k = \boldsymbol{\psi}(\mathbf{x}^{(k)})^{\mathrm{T}}\mathbf{h}_0 = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2}x_i^{(k)}x_j^{(k)}(\mathbf{H}_0)_{ij} = \frac{1}{2}(\mathbf{x}^{(k)})^{\mathrm{T}}\mathbf{H}_0\mathbf{x}^{(k)}.$$

□

4.5 Linear Dependence

Despite intuition, it turns out that there are limits around the number of gradients that can be interpolated for the quadratic model. There is redundant information after two gradients and their corresponding function values are used for interpolation. Similarly, we have dependent rows in the matrix when 3 or more gradients are interpolated. To see this, we consider the constructive theorem.

Theorem 3. *Suppose that gradients are calculated and interpolated at* $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, *and* $\mathbf{x}^{(3)}$. *Then the matrix*

$$\mathbf{G} = \begin{pmatrix} \mathbf{I} & \mathbf{D}_1 \\ \mathbf{I} & \mathbf{D}_2 \\ \mathbf{I} & \mathbf{D}_3 \end{pmatrix} \tag{4.38}$$

is rank deficient where $\mathbf{D}_i$ *is the Jacobian of* $\boldsymbol{\psi}(\mathbf{x}^{(i)})$.

Proof. We assume that the points $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, and $\mathbf{x}^{(3)}$ are distinct and span a three-dimensional subspace otherwise the result is trivially satisfied. Note the that rows of $\mathbf{G}$ are given by the partial derivatives of the vector $\boldsymbol{\phi}$, i.e.,

$$\mathbf{G}_{i,:} = \frac{\partial}{\partial x_{i \bmod n}}\boldsymbol{\phi}(\mathbf{x}^{(\lfloor i/n \rfloor + 1)}).$$

For notational simplicity, we let $\boldsymbol{\phi}_i = (\partial/\partial x_i)$. Via Gaussian elimination, we see that the linear combination of rows equals zero,

$$\sum_{i=1}^{n}(x_i^{(1)} - x_i^{(2)})\left[\boldsymbol{\phi}_i(\mathbf{x}^{(3)}) - \boldsymbol{\phi}_i(\mathbf{x}^{(1)})\right] - (x_i^{(1)} - x_i^{(3)})\left[\boldsymbol{\phi}_i(\mathbf{x}^{(2)}) - \boldsymbol{\phi}_i(\mathbf{x}^{(1)})\right].$$

□

The same can be shown for two gradients with their corresponding function values although we omit the proof here.

Empirically, we observe the rank deficiency increases in a predictable way. We expect the following conjecture is simple to prove be have not had a chance to complete it.

Conjecture 1. *For a general Vandermonde matrix constructed with d points that span a d-dimensional subspace and their corresponding gradient blocks, i.e.,*

$$\mathbf{V} = \begin{pmatrix} \mathbf{L} & \mathbf{Q} \\ \mathbf{I} & \mathbf{D}_1 \\ \vdots & \vdots \\ \mathbf{I} & \mathbf{D}_d \end{pmatrix}, \tag{4.39}$$

the matrix $\mathbf{V}$ has exactly $\frac{d^2-d}{2}$ linearly dependent rows.

This linear dependence results in an inconsistent system of constraints. We use sketching to form a full rank approximation by eliminating the required number of rows as dictated by the conjecture above. We can view sketching here as a random weighted average of the constraints which prevents us from discarding any single (possibly significant row).

4.6 Dimension Reduction to Preserve Interpolation Feasibility

As mentioned in the previous section, linear dependence becomes a concern when more than one gradient is incorporated into the model. One method to address the issue is by discarding the number of dependent rows. Although simple, this approach is coarse as it eliminates all information from a particular interpolation condition. In addition, it is possible to choose an adversarial arrangement for dropped rows that still results in a singular KKT matrix. We'd like a robust method to accomplish this task.

We find inspiration in the field of sketching which uses certain classes of randomized linear transformations to preserve distances between points when embedded into a lower dimensional subspace with high probability [Martinsson and Tropp, 2020]. Suppose we want to find the projection

of the columns of $\mathbf{V}$ onto the subspace spanned by the range of an orthonormal matrix $\mathbf{S}$. The i^{th} column of $\mathbf{B} = \mathbf{S}\mathbf{S}^{\mathrm{T}}\mathbf{V}$ gives the projection of the i^{th} column of $\mathbf{V}$ ($\mathbf{v}_i$) onto the range of $\mathbf{S}$.

Accordingly, the i^{th} column of $\mathbf{S}^{\mathrm{T}}\mathbf{V}$ gives the coordinates of the projection of $\mathbf{v}_i$ in the basis col($\mathbf{B}$). The Johnson–Lindenstrauss lemma [Johnson and Lindenstrauss, 1984] shows that if $\mathbf{S}$ has enough rows (independent of the dimension of $\mathbf{v}$) there exists a projection into a lower dimensional subspace that preserves relative distance between points. This work was extended to show there exist easily computed and fast to apply transformations that preserve distances with high probability [Ailon and Chazelle, 2009]. Although further discussion of sketching is beyond the scope of this work, we borrow their ideas by projecting the interpolating constraints into a random subspace of the appropriate dimension to yield a feasible system. The reason for random projections is to avoid adversarial instances where the "wrong" rows are dropped yielding a smaller but still inconsistent system. In what follows, it may be helpful to think of $\mathbf{S}$ as Gaussian which is one of the simplest forms of a sketching matrix.

To illustrate, consider a full rank matrix in $\mathbb{R}^{m\times n}$ with a row of zeros appended to it. In this extreme example, we must drop the zero row to have a full rank matrix. Dropping any other row will result in a matrix with one less row that is still rank deficient. To avoid problematic instances such as these, we sketch it randomly which in essence combines all the constraints, then drop one row. For a Gaussian sketch we will have a full rank system with the probability 1.

Returning to problem (4.27) where $m + dn$ interpolation conditions must be satisfied, we use a sketching matrix $\mathbf{S} \in \mathbb{R}^{(m+dn)\times r}$ where r is the desired reduced dimension which is less than or equal to the rank of the KKT system. We can now work with the coordinates in the columns space of $\mathbf{S}$.

$$\mathbf{S}^{\mathrm{T}}\left(\begin{bmatrix}\mathbf{L}\\ \mathbf{I}\\ \vdots\\ \mathbf{I}\end{bmatrix}\mathbf{g} + \begin{bmatrix}\mathbf{Q}\\ \mathbf{D}_1\\ \vdots\\ \mathbf{D}_d\end{bmatrix}\mathbf{h}\right) = \mathbf{S}^{\mathrm{T}}\begin{bmatrix}\mathbf{f}\\ \nabla f_1\\ \vdots\\ \nabla f_d\end{bmatrix}. \tag{4.40}$$

For notational simplicity, let $\mathbf{S}^{\mathrm{T}} = [\mathbf{S}_0^{\mathrm{T}}, \mathbf{S}_1^{\mathrm{T}}, \ldots, \mathbf{S}_d^{\mathrm{T}}]$ with $\mathbf{S}_0 \in \mathbb{R}^{m\times r}$ and $\mathbf{S}_i \in \mathbb{R}^{n\times r}$ for $i > 0$ and

$$\begin{pmatrix} \mathbf{L}_0 \\ \mathbf{L}_1 \\ \vdots \\ \mathbf{L}_d \end{pmatrix} = \begin{pmatrix} \mathbf{L} \\ \mathbf{I} \\ \vdots \\ \mathbf{I} \end{pmatrix}, \quad \begin{pmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_1 \\ \vdots \\ \mathbf{Q}_d \end{pmatrix} = \begin{pmatrix} \mathbf{Q} \\ \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_d \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_d \end{pmatrix} = \begin{pmatrix} f \\ \nabla f_1 \\ \vdots \\ \nabla f_d \end{pmatrix}. \tag{4.41}$$

We can rewrite (4.40) as

$$\underbrace{\left(\sum_{i=0}^{d} \mathbf{S}_i^{\mathrm{T}} \mathbf{L}_i\right)}_{=\mathbf{A}} \mathbf{g} + \underbrace{\left(\sum_{i=0}^{d} \mathbf{S}_i^{\mathrm{T}} \mathbf{Q}_i\right)}_{=\mathbf{B}} \mathbf{h} = \underbrace{\sum_{i=0}^{d} \mathbf{S}_i^{\mathrm{T}} \mathbf{v}_i}_{=\mathbf{c}}. \tag{4.42}$$

The resultant problem is familiar

$$\begin{aligned} \min_{\mathbf{g},\mathbf{h}} \quad & \frac{1}{2}\|\mathbf{h}\|^2 \\ \text{subject to} \quad & \mathbf{A}\mathbf{g} + \mathbf{B}\mathbf{h} = \mathbf{c}. \end{aligned} \tag{4.43}$$

The same procedure as before gives the KKT system

$$\begin{pmatrix} \mathbf{0} & \mathbf{A}^{\mathrm{T}} \\ \mathbf{A} & \mathbf{B}\mathbf{B}^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \mathbf{g} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{c} \end{pmatrix}. \tag{4.44}$$

The $\mathbf{A}$ blocks are easy to compute since there is no cost associated with forming $\mathbf{L}_i$. An appropriately chosen sketch can be applied quickly such as partial a Fourier or Hadamard transform. The same is true for the right hand side. Turning our attention to $\mathbf{B}\mathbf{B}^{\mathrm{T}}$, we see that

$$\mathbf{B}\mathbf{B}^{\mathrm{T}} = \sum_{i=0}^{d}\sum_{j=0}^{d} \mathbf{S}_i^{\mathrm{T}} \mathbf{Q}_i \mathbf{Q}_j^{\mathrm{T}} \mathbf{S}_j.$$

Fortunately, as shown in Section 4.4, the $\mathbf{Q}_i\mathbf{Q}_j^{\mathrm{T}}$ are computed at a cost $O(m^2 n)$ for $\mathbf{Q}_0\mathbf{Q}_0^{\mathrm{T}}$ or $O(mn^2)$ for $\mathbf{D}_i\mathbf{D}_j^{\mathrm{T}}$. Importantly, we apply the sketching matrices to $\mathbf{Q}_i\mathbf{Q}_j^{T}$ and not the $\mathbf{Q}$'s directly. The cost to apply any sketch matrix is at most $O(\max\{m^2 r, n^2 r, mnr\})$. Since there are d matrices in the summation, the complexity of applying the sketching and summing is $O(r\ \max\{m^2, dn^2\})$.

Once we've solved for $\mathbf{g}$ and the multipliers, we can construct $\mathbf{h}$ easily with

$$\mathbf{h} = \mathbf{B}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{Q}^{\mathrm{T}}(\mathbf{S}_0\boldsymbol{\lambda}) + \sum_{i=1}^{d} \mathbf{D}_i^{\mathrm{T}}(\mathbf{S}_i\boldsymbol{\lambda}). \tag{4.45}$$

The sketch matrices can be applied cheaply to $\boldsymbol{\lambda}_i$. Letting $\tilde{\boldsymbol{\lambda}}_i = \mathbf{S}_i\boldsymbol{\lambda}$ gives the vector representation of the Hessian

$$\mathbf{h} = \mathbf{Q}^{\mathrm{T}}\tilde{\boldsymbol{\lambda}}_0 + \sum_{i=1}^{d} \mathbf{D}_i^{\mathrm{T}}\tilde{\boldsymbol{\lambda}}_i. \tag{4.46}$$

Corollary 3. *Suppose that* $p(\mathbf{x}) = \mathbf{g}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} = \mathbf{g}^T\mathbf{x} + \mathbf{h}^T\boldsymbol{\psi}(\mathbf{x})$ *where* $\mathbf{g}$ *and* $\mathbf{h}$ *solve* (4.40) *and* $\boldsymbol{\lambda}$ *is the corresponding multiplier. Defining* $\tilde{\boldsymbol{\lambda}}_i = \mathbf{S}_i\boldsymbol{\lambda}$ *gives the representation for the model Hessian*

$$\mathbf{H} = \mathbf{M} - \frac{1}{2}\mathrm{diag}(\mathrm{diag}(\mathbf{M})) + \sum_{k=1}^{d}\left\{\tilde{\boldsymbol{\lambda}}_k(\mathbf{x}^{(k)})^T + \mathbf{x}^{(k)}\tilde{\boldsymbol{\lambda}}_k^T - \mathrm{diag}(\tilde{\boldsymbol{\lambda}}_k \odot \mathbf{x}^{(k)})\right\}, \tag{4.47}$$

where $\mathbf{M} = \mathbf{X}^T\mathrm{diag}(\tilde{\boldsymbol{\lambda}}_0)\mathbf{X}$.

Proof. Replacing $\boldsymbol{\lambda}_i$ in Theorem 2 with $\tilde{\boldsymbol{\lambda}}_i$ immediately yields the result. □

Since inversion of the matrix is the dominating computational cost, we don't worry about the complexity of applying the sketch.

4.7 Algorithm

We present our algorithm below. The main difference between the generic trust region algorithm and the one proposed here is the maintenance and improvement of an interpolating set and our formulation of the subroutine `HermiteInterpolant` which uses derivative information to form a quadratic model. As we began writing our results, it became clear that there were many variants worthy of pursuit. The most notable omission is the approximate Hessian. We opted to focus on the

Algorithm 2: `HermiteInterpolant`

Input: Current iterate $\mathbf{x}$, current function value f, current gradient (or null value) $\mathbf{g}$, interpolant data $\mathbf{X}, \mathbf{f}$, and $\mathbf{G}$.
Output: Gradient $\mathbf{g}$ and Hessian $\mathbf{H}$ for model interpolating polynomial.

Center data, $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{x}$ and $\mathbf{f} \leftarrow \mathbf{f} - f$.
Determine gradients in $\mathbf{G}$ and corresponding points to interpolate.
Construct the KKT system in (4.44).
Solve for $\mathbf{g}$ and $\boldsymbol{\lambda}$.
Use (4.45), (4.46), and (4.47) to form $\mathbf{H}$.

minimum norm solution here for simplicity's sake. Beyond ease of exposition, it is unclear whether SR1 or BFGS type updates make sense when curvatures pairs are computed periodically rather than every step. We compare the MNS to a BFGS penalized version in our numerical examples below, but before using with confidence, additional analysis should be performed.

Algorithm 3: `HermiteTrustRegionMethod`

Input: $\mathbf{x}_0 \in \mathbb{R}^n$, max # gradients, min # gradients, $0 < \eta_{\text{good}} \leq \eta_{\text{great}} < 1$, $\gamma_{\text{inc}} > 1$, $\gamma_{\text{dec}} \in (0,1)$, $p \geq 0$, $\delta_0 > 0$.
Output: Optimal $\mathbf{x}^*$ solving (4.1)

Sample $n-1$ points around $\mathbf{x}_0$ and store as rows of $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n]^T$
Evaluate $f(\mathbf{x})$ at n points interpolating conditions (sample n points or interpolate gradient)
while Not converged do
  $\mathbf{g}, \mathbf{H} \leftarrow$ `HermiteInterpolant`$(\mathbf{x}_i,\ f_i, \mathbf{g}_i, \mathbf{X}, \mathbf{f})$.
  Construct model $m_k(\mathbf{s}) = \mathbf{g}^{\mathrm{T}}\mathbf{s} + (1/2)\,\mathbf{s}^{\mathrm{T}}\mathbf{H}\mathbf{s}$.
  Solve (4.2) to obtain $\mathbf{s}_k$.
  $\mathbf{x}^+ \leftarrow \mathbf{x}_k + \mathbf{s}_k$.
  if k **mod** $p = 0$ **then**
    $\mathbf{g} \leftarrow \nabla f(\mathbf{x}^+)$.
  else
    $\mathbf{g} \leftarrow$ Null
  end
  Compute ρ_k as in (4.3).
  if $\rho_k > \eta_{\textbf{good}}$ **(successful iteration) then**
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}^+$
    if $\rho_k > \eta_{\textbf{great}}$ **and** $\|\mathbf{s}_k\| \geq 0.9\,\delta_k$ **(very successful iteration) then**
      $\delta_{k+1} \leftarrow \gamma_{\text{inc}}\delta_k$.
  else
    $\delta_{k+1} \leftarrow \gamma_{\text{dec}}\delta_k$.
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$.
  end
  Perform geometry improving step to to update $\mathbf{X}$, $\mathbf{f}, \mathbf{G}$ (see [Conn et al., 2009] for details).
  $k \leftarrow k+1$.
end

4.8 Numerical Examples

For our numerical experiments, we ran our algorithm in Python on the unconstrained CUTEst problems [Gould et al., 2015] of dimension 10 to 150. Our mean and median problem dimension was 65. The CUTEst test suite is a collection of differentiable but non-convex optimization problems

of varying size and difficulty used in the field of optimization to test solver performance. Although the CUTEst problems are written in Fortran, we used the PyCUTEst [Fowkes and Roberts, 2018] package to port the problems into Python. We ran each solver until the model gradient fell below 10^{-6}, the trust region shrank below machine precision, or until 1000 iterations had been completed. We compared our Hermite interpolation based TR algorithm to one using standard interpolation which was accomplished by specifying that exactly zero gradients should be used to construct an interpolation model. We plotted log-log plots of the scaled function values by iteration. The scaled function value for solver s at iteration i is

$$f_{s,i,\text{scaled}} = \frac{f_{s,i} - F_{\min}}{F_{\max} - F_{\min}} + \epsilon, \tag{4.48}$$

where $F_{\min}$ and $F_{\max}$ are the minimum and maximum function values encountered over all iterations across all solvers for a particular problem. The offset ϵ is used to allow plotting on a log-scale. Although our choice of ϵ will impact the appearance of different curves for each solvers in the last few iterations, we are concerned primarily with the qualitative behavior over time. For our purposes here, we set $\epsilon = 10^{-6}$.

We focused on interpolating 1, 2, or 3 gradients with new gradients calculated every 10 iterations. We have included four representative examples in this chapter to avoid overwhelming the reader. Additional plots for other CUTEst problems can be found in Appendix Figure 4.1 shows the performance of the Hermite interpolation model using the minimum norm solution. Figure 4.2 shows the same, but with the interpolated model regularized to the BFGS approximate Hessian. Finally, Figure 4.3 compares the BFGS to the minimum norm interpolants fitting two gradients with an update every 10 iterations. The horizontal axis gives the iteration count. In all cases, performance of the standard interpolation method is provided for reference. We only show four problems in these figures to avoid overwhelming the reader, but a comprehensive set can be found in Appendix B.

The Hermite based models do not universally outperform the standard function interpolation models, but do better in general. It is also clear that using more gradients doesn't always result

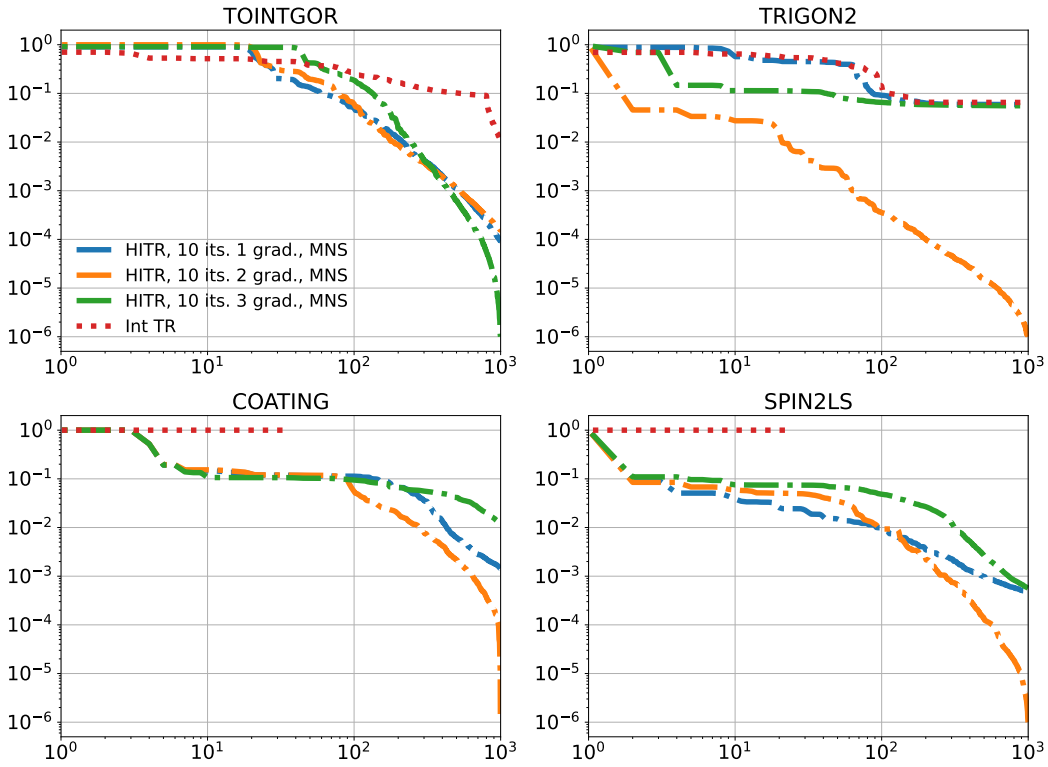

Figure 4.1: Scaled function values (y-axis) at each iteration (x-axis) using minimum norm solution (MNS) for Hermite TR model with 1, 2, or 3 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.

in improved performance. This might be the result of infrequently updated gradients becoming "stale" and preserving information from a different basin of the objective's domain. In such non-convex cases, the quadratic may not be expressive enough. It is not clear that using a BFGS update matrix for regularization purposes gives a noticeable benefit. Since curvature pairs are typically constructed via back-to-back iterations where a secant line reasonably approximates a directional derivative, infrequently updated gradients might be too crude to form useful approximation. Further investigation is warranted but is beyond the scope of this chapter. In general, the results are promising. Even infrequently updated gradients appear to improve the behavior of trust region methods compared to standard interpolation.

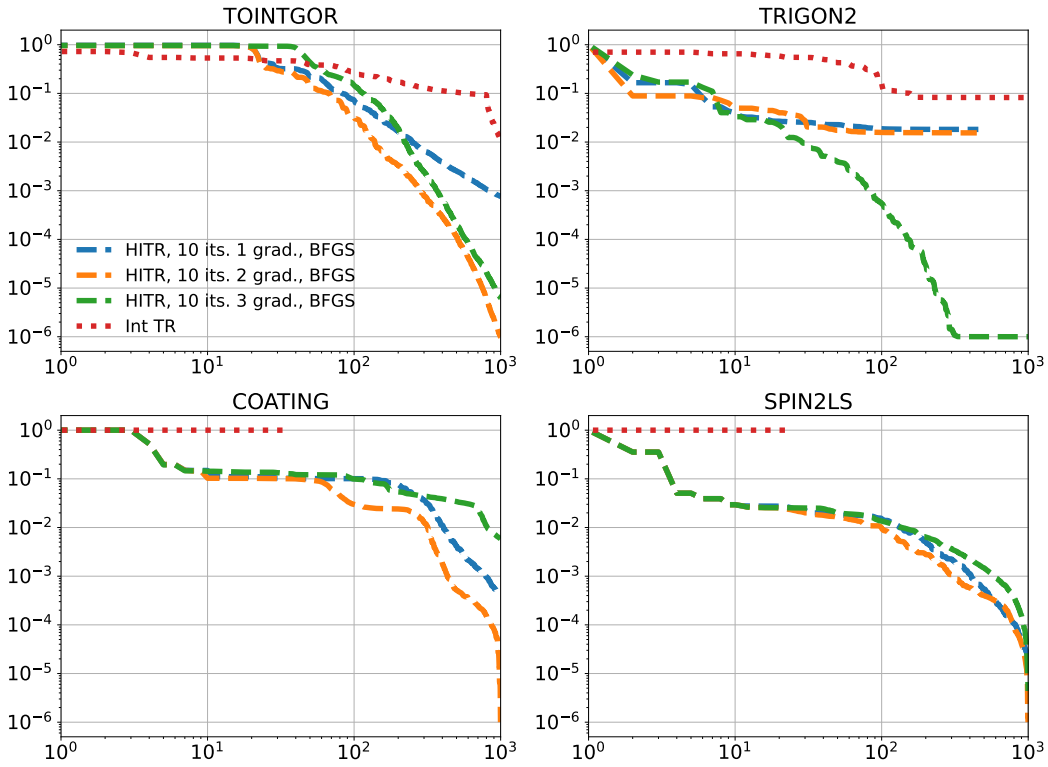

Figure 4.2: Scaled function values (y-axis) at each iteration (x-axis) regularized to BFGS approximate Hessian for Hermite TR model with 1, 2, or 3 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.

4.9 Conclusion

We introduced a method for interpolating gradients or partial derivatives for a quadratic trust region model in higher dimension. Importantly, our algorithm relies primarily on function values and can be iterated even when derivatives are unknown. We provided expressions for computing blocks matrices necessary to solve the linear system for finding polynomial coefficients in the TR model that drastically reduced both the computational complexity and memory requirements compared to a naive implementation.

We also showed that the problem can be easily modified to accommodate an approximate Hessian and penalize models that have disparate second derivatives. Complications associated with

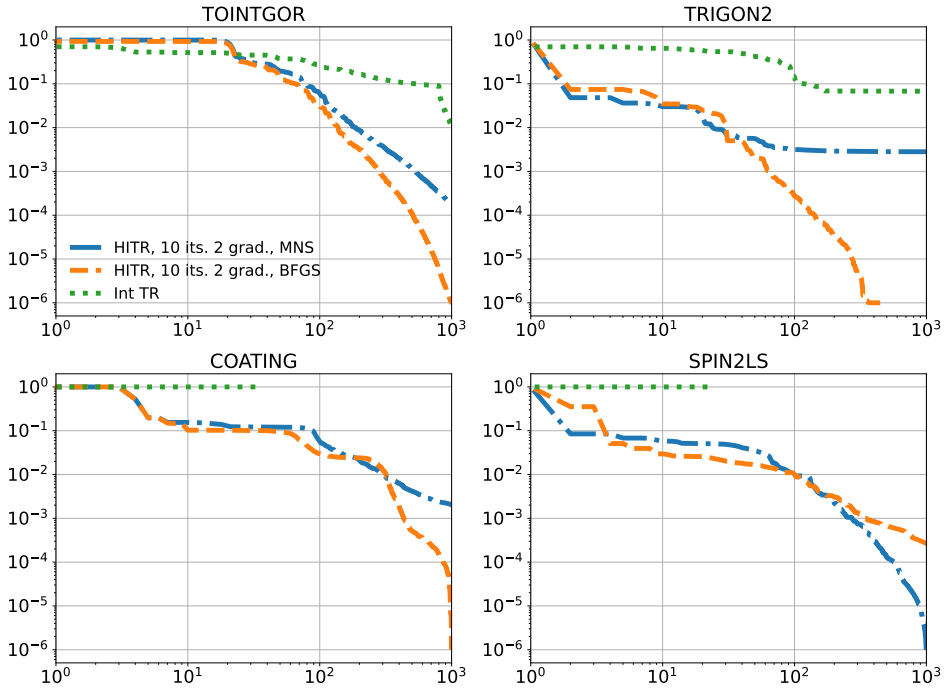

Figure 4.3: Scaled function values (y-axis) at each iteration (x-axis) comparing BFGS Hessian regularized model to MNS model for Hermite TR with 2 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.

linear dependence were discussed as were methods for handling this redundancy when multiple gradients are interpolated. Given the cost of solving the KKT system, it is worth investigating how low dimensional embeddings might be used to reduce the cost of finding a model for each iteration. Other future work includes establishing convergence rates for Hermite interpolation and investigating how gradients might be used to improve the complexity of geometry improving subroutines that were absent from our treatment here.

Chapter 5

TROPHY: Trust Region Optimization using a Precision HierarchY

5.1 Introduction

Optimization methods are used in many applications, including engineering, science, and machine learning. The memory requirements and run time for different methods have been studied extensively and determine the problem sizes that can be run on existing hardware. Similarly, the energy consumption of each method determines its cost and carbon footprint, which is a growing concern [Hao, 2019].

With the desire to incorporate more data into models and ever-increasing computational power, problem scales have grown as well. To improve efficiency, modern computers tightly integrate graphical processing units (GPUs) and other accelerators. Many of these units natively support data types of differing precision to lessen the storage and computational load. Previous work has found significant differences in the overall energy consumption for double- and single-precision computations [Molka et al., 2010, Kestor et al., 2013]. Server-level products such as NVIDIA Tensor cores in V100 GPUs show 16× improvement over traditional double precision [Abdelfattah et al., 2021].

Such gains come at a cost, however. Classical algorithms such as the Gram–Schmidt process are well known to suffer from loss of orthogonality and numerical instability due to limited precision [Golub and Van Loan, 1996]. In an effort to ameliorate algorithmic issues with accuracy and stability, there has been a flurry of activity using mixed precision. These methods utilize multiple data types in a principled fashion to reduce the computational burden without sacrificing accu-

racy. A few of the many applications are tomographic reconstruction [Doucet et al., 2019], seismic modeling [Ichimura et al., 2018], and neural network training [Jia et al., 2018, Micikevicius et al., 2018, Wang et al., 2018, Carmichael et al., 2019]. Mixed-precision methods have been used generically to minimize the cost of linear algebra methods [Abdelfattah et al., 2021], iterative schemes, and improved finite element solvers [Strzodka and Göddeke, 2006, Göddeke et al., 2007].

There are a variety of auto-tuning algorithms that attempt to identify variables within a program that can safely be cast in a lower precision, while satisfying some accuracy constraint [Chiang et al., 2017a] [Graillat et al., 2019, Guo and Rubio-González, 2018, Menon et al., 2018, Rubio-González et al., 2013]. Our method proposed does not attempt to identify low precision candidates nor do we try satisfying accuracy constraints. All computation within the objective/gradients are performed in the lowest precision possible and only increase after it is deemed necessary for the solver to proceed.

A recent paper by Gratton and Toint [Gratton and Toint, 2020] illustrates potential savings in an optimization setting via variable-precision trust region (TR) methods. We investigate the ideas proposed in their work but with an important difference. In particular, their algorithm (TR1DA) requires access to an approximate objective, $\bar{f}(\mathbf{x}_k, \omega_{f,k})$, and gradient, $\bar{\mathbf{g}}(\mathbf{x}_k, \omega_{g,k})$, where $\omega_{f,k}$ and $\omega_{g,k}$ are uncertainty parameters (for the kth iterate $\mathbf{x}_k$) that satisfy

$$|\bar{f}(\mathbf{x}_k, \omega_{f,k}) - f(\mathbf{x}_k)| \leq \omega_{f,k} \qquad \text{and} \qquad \frac{\|\bar{\mathbf{g}}(\mathbf{x}_k, \omega_{g,k}) - \mathbf{g}(\mathbf{x}_k)\|}{\|\bar{\mathbf{g}}(\mathbf{x}_k, \omega_{g,k})\|} \leq \omega_{g,k}.$$

Their error model requires user specified absolute error bounds on function and gradient values; such bounds are difficult to realize in practice as computational complexity grows for reasons such as catastrophic cancellation and accumulated round-off error. Our focus here is on designing an algorithm that performs well without assumptions on the output error when using lower precision.

In this paper, we introduce TROPHY (**T**rust **R**egion **O**ptimization using a **P**recision **H**ierarch**Y**), a mixed-precision TR method for unconstrained optimization. We provide practically verifiable conditions intended to determine whether the error related to a current precision level may be interfering with the dynamics of the TR algorithm. If the conditions are not satisfied, we

increase the precision level until they are. Our goal is to lighten the computational load without sacrificing accuracy of the final solution. By using a limited-memory, symmetric rank-1 update (L-SR1) to the approximate Hessian, the method is suitable for large-scale, high-dimensional problems. We compare the method with a standard TR method—supplied with access to either a single- or double-precision evaluation of the function and gradient—on the Constrained and Unconstrained Testing Environment with safe threads (CUTEst) test problem collection [Gould et al., 2015] and on a large-scale weather model based on the PyDDA software package [Jackson et al., 2020].

Since computational, storage, and communication savings are based on hardware implementations of different precision types rather than assumed theoretical values, our primary metric for comparison will be adjusted function evaluations rather than time. Simply put, adjusted function evaluations discount computations performed in lower-precision levels. The goal here is to provide a proof of concept for computational gains attainable by exploiting variable precision in TR methods. In practice, improvements in energy consumption, time, communication, and memory must be realized through optimized hardware which is beyond the scope of this paper.

5.2 Background

Consider the unconstrained minimization of a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$,

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}). \tag{5.1}$$

We are motivated by problems where the objective and its derivatives are expensive to calculate as is typical for large-scale computing. In this paper we focus on the TR framework, but could have studied line-search methods instead such as L-BFGS, which is a popular quasi-Newton method distributed in SciPy [Virtanen et al., 2020]. However, it is remarkably simpler to illustrate the effect of error on the quality of models within a TR method; that is likely the reason TR methods were employed in [Gratton and Toint, 2020]. In the following subsections we give an overview of the general framework for TR methods and describe the model function used in our algorithm.

5.2.1 Trust Region Methods

Trust region methods are iterative algorithms used for numerical optimization. At each iteration (with the counter denoted by k), a model function $m_k : \mathbb{R}^n \to \mathbb{R}$ is built around the incumbent point or iterate, $\mathbf{x}_k$, such that $m_k(\mathbf{0}) = f(\mathbf{x}_k)$ and $m_k(\mathbf{s}) \approx f(\mathbf{x}_k + \mathbf{s})$. The model, m_k, is intended to be a "good" local model of f on the **trust region**, $\{\mathbf{s} \in \mathbb{R}^n : \|\mathbf{s}\| \leq \delta_k\}$ for $\delta_k > 0$. We refer to δ_k as the **trust region radius**. A **trial step**, $\mathbf{s}_k$, is then computed via a(n approximate) solution to the **trust region subproblem**,

$$\mathbf{s}_k = \underset{\|\mathbf{s}\| \leq \delta_k}{\operatorname{argmin}} \; m_k(\mathbf{s}), \tag{5.2}$$

for $\mathbf{s} \in \mathbb{R}^n$. By an approximate solution to the TR subproblem (5.2), we mean that one requires the **Cauchy decrease condition** to be satisfied:

$$f(\mathbf{x}_k) - m_k(\mathbf{s}_k) \geq \frac{\mu}{2} \min \left\{ \delta_k, \frac{\|\mathbf{g}_k\|}{C} \right\}, \tag{5.3}$$

where μ and C are constants and $\mathbf{g}_k = \nabla m(\mathbf{x}_k)$. A common choice for m_k is a quadratic Taylor expansion, namely, $m_k(\mathbf{s}) = f(\mathbf{x}_k) + \mathbf{g}_k^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \nabla^2 f(\mathbf{x}_k) \mathbf{s}$. In practice, $\nabla^2 f(\mathbf{x}_k)$ is typically replaced with a (quasi-Newton) approximation.

Having computed $\mathbf{s}_k$, the standard TR method then compares the true decrease in the function value, $f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)$, with the decrease predicted by the model, $m_k(\mathbf{0}) - m_k(\mathbf{s}_k)$. In particular, one computes the quantity

$$\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}. \tag{5.4}$$

If ρ_k is sufficiently positive ($\rho_k > \eta_{\text{good}}$ for fixed $\eta_{\text{good}} > 0$), then the algorithm accepts $\mathbf{x}_k + \mathbf{s}_k$ as the incumbent point $\mathbf{x}_{k+1}$ and may possibly increase the TR radius $\delta_k < \delta_{k+1}$ (if $\rho_k > \eta_{\text{great}}$ for fixed $\eta_{\text{great}} \geq \eta_{\text{good}}$). This scenario is called a **successful iteration**. On the other hand, if ρ_k is not sufficiently positive or is negative ($\rho_k < \eta_{\text{good}}$), then the incumbent point stays the same, $\mathbf{x}_{k+1} = \mathbf{x}_k$, and we set $\delta_{k+1} < \delta_k$. For the experiments below, we chose $\eta_{\text{good}} = 10^{-5}$ and $\eta_{\text{great}} = 0.10$. This process is iterated until a stopping criterion is met, e.g., when the gradient

norm $\|\nabla f(\mathbf{x}_k)\|$ is below a given tolerance. Under mild assumptions, TR methods asymptotically converge to stationary points of $f(\mathbf{x})$ [Conn et al., 2000].

5.2.2 Model Function

The model function, m_k, must be specified for a TR algorithm. Popular choices include linear or quadratic approximations of the objective using Taylor series or interpolation methods; the latter are often employed in derivative-free optimization [Larson et al., 2019]. Since many applications of interest are high-dimensional or have costly objective and derivative functions, it is difficult if not impossible to compute and/or store the Hessian matrix for use in quadratic TR models with memory requirement scaling as $\mathcal{O}(n^2)$. A common technique that exploits derivative information while keeping the cost low is to use *curvature pairs* given by $\mathbf{s}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_k+\mathbf{s}_k)-\nabla f(\mathbf{x}_k)$. After each successful iteration, the curvature pairs are used to update the current approximate Hessian denoted by $\mathbf{H}_k$. These updates employ secant approximations of second derivatives. Common update rules include BFGS, DFP, and SR1 [Nocedal and Wright, 2006].

In this work we use a limited-memory symmetric rank-1 update (L-SR1) to the approximate Hessian. This update rule requires the user to set a memory parameter that specifies a number of secant pairs to use in the approximate Hessian. Since we require only a matrix-vector product and not the explicit Hessian, we can implement a matrix-free version reducing the storage cost to $\mathcal{O}(n)$. Thus, our TR subproblem is

$$\mathbf{s}_k = \underset{\|\mathbf{s}\|\leq\delta_k}{\operatorname{argmin}} \ \mathbf{s}^T\nabla f(\mathbf{x}_k) + \frac{1}{2}\mathbf{s}^T\mathbf{H}_k\mathbf{s}, \tag{5.5}$$

which we recast and approximately solve using the Steihaug conjugate gradient method implemented in [Berahas et al., 2019][Appendix B.4].

In the next section we describe the dynamic precision framework and present criteria for when precision should switch. We then are prepared to give a formal statement of TROPHY. In Section 5.4 we describe the problems on which we have tested TROPHY, and in Section 5.5 we discuss the results of our experiments.

5.3 Method

We assume access to a hierarchy of arithmetic precisions for the evaluation of both $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$, but the direct (infinite-precision) evaluation of $f(\mathbf{x}), \nabla f(\mathbf{x})$ is unavailable. We formalize this slightly by supposing we are given oracles that compute $f^p(\mathbf{x}), \nabla f^p(\mathbf{x})$ for $p \in \{0, \ldots, P\}$. With very high probability, given a uniform distribution on all possible inputs $\mathbf{x}$, the oracles satisfy the inequalities

$$|f^p(\mathbf{x}) - f(\mathbf{x})| > |f^{p+1}(\mathbf{x}) - f(\mathbf{x})|, \quad \|\nabla f^p(\mathbf{x}) - \nabla f(\mathbf{x})\| > \|\nabla f^{p+1}(\mathbf{x}) - \nabla f(\mathbf{x})\|.$$

For a tangible example, if intermediate calculations involved in the computation of $f(\mathbf{x})$ can be done in half, single, or double precision, then we can denote $f^0(\mathbf{x})$, $f^1(\mathbf{x})$, and $f^2(\mathbf{x})$ as the oracles using only half, single, or double, respectively.

To build on the generic TR method described in Section 5.2, we must specify when and how to switch precision. We can identify two additional difficulties presented in the multiple-precision setting. First, it is currently unclear how to compute ρ_k in (5.4) since our error model assumes we have no access to an oracle that directly computes $f(\cdot)$. Second, because models m_k typically use function and gradient information provided by $f(\cdot)$ and $\nabla f(\cdot)$, we must specify how to construct models using lower precision oracles.

For the first of these two issues, we make a practical assumption that **the highest level of precision available to us should be treated as if it were infinite precision.** Although this is a theoretically poor assumption, virtually all computational optimization makes it implicitly; algorithms are analyzed over the real numbers but are typically implemented using floating point arithmetic (often double). Thus, in the notation we have developed, the optimization problem we actually aim to solve is not (5.1) but

$$\min_{\mathbf{x} \in \mathbb{R}^n} f^P(\mathbf{x}), \tag{5.6}$$

so that the ρ-test in (5.4) is replaced with

$$\rho_k = \frac{f^P(\mathbf{x}_k) - f^P(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)} = \frac{\text{ared}_k}{\text{pred}_k}. \tag{5.7}$$

The values ared and pred were introduced to denote "actual reduction" and "predicted reduction", respectively. We note that computing (5.7) still entails two evaluations of the highest-precision oracle, $f^P(\cdot)$, which is exactly what we hoped to avoid by using mixed-precision. Our algorithm avoids the cost of full-precision evaluations by dynamically adjusting the precision level $p_k \in \{0, \ldots, P\}$ between iterations so that in the kth iteration, ρ_k is approximated by

$$\tilde{\rho}_k = \frac{f^{p_k}(\mathbf{x}_k) - f^{p_k}(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)} = \frac{\text{ered}_k}{\text{pred}_k}, \tag{5.8}$$

introducing ered to denote "estimated reduction". To update p_k, we are motivated by a strategy similar to one employed in [Heinkenschloss and Vicente, 2002] and [Kouri et al., 2014]. We introduce a variable θ_k that is not initialized until the end of the first unsuccessful iteration and set $p_0 = 0$. When the first unsuccessful iteration is encountered, we set

$$\theta_k \leftarrow |\text{ared}_k - \text{ered}_k| \,. \tag{5.9}$$

Notice that we must incur the cost of two evaluations of $f^P(\cdot)$ following the first unsuccessful iteration in order to compute ared_k. From that point on, θ_k is involved in a test triggered on every unsuccessful iteration (in which the TR radius is sufficiently small) to determine whether the precision level, p_k, should be increased. We compute θ_k and test for precision when $\delta_k < \Delta_{\text{prec}}$. The value Δ_{prec} is set to be a length scale where numerical imprecision is a concern.

Introducing a predetermined **forcing sequence** $\{r_k\}$ satisfying $r_k \in [0, \infty)$ for all k and $\lim_{k\to\infty} r_k = 0$, and fixing a parameter $\omega \in (0, 1)$, we check on any unsuccessful iteration whether

$$\theta_k^{\omega} \leq \eta \min\{\text{pred}_k, r_k\} \,, \tag{5.10}$$

where $\eta = \min\{\eta_{\text{good}}, 1 - \eta_{\text{great}}\}$. If (5.10) does not hold, then we increase $p_{k+1} = p_k + 1$ and again update the value of θ_k according to (5.9) (thus incurring two more evaluations of $f^P(\cdot)$). The reasoning behind the test in (5.10) is that if (the unknown) ρ_k in (5.7) satisfies $\rho_k \geq \eta$, then

$$\eta \leq \rho_k = \frac{\text{ared}_k}{\text{pred}_k} \leq \frac{|\text{ared}_k - \text{ered}_k| + \text{ered}_k}{\text{pred}_k} \approx \frac{\theta_k + \text{ered}_k}{\text{pred}_k} = \frac{\theta_k}{\text{pred}_k} + \tilde{\rho}_k. \tag{5.11}$$

Thus, for the practical test (5.8) to be meaningful, we need to ensure that $\theta^k/\text{pred}_k < \eta$, which is what (5.10) attempts to enforce. The use of ω and the forcing sequence in (5.10) is designed to

ensure that we eventually do not tolerate error, since (5.11) involves an approximation due to the estimate θ_k. The forcing sequence would likely be necessary to guarantee convergence for theoretical analysis, but is not critical to the performance of a practical algorithm, and was not employed in our implementation. For concreteness, if a forcing sequence were employed, one might consider a slowly decaying sequence such as $r_k = 1/\sqrt{k}$.

It remains to describe how we deal with our second identified difficulty, the construction of m_k in the absence of evaluations of $f(\cdot)$ and $\nabla f(\cdot)$. As is frequently done in trust region methods, we will employ quadratic models of the form

$$m_k(\mathbf{s}) = f_k + \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top \mathbf{H}_k \mathbf{s}. \tag{5.12}$$

Having already defined rules for the update of p_k through the test (5.10), we take in the kth iteration $f_k = f^{p_k}(\mathbf{x})$ and $\mathbf{g}_k = \nabla f^{p_k}(\mathbf{x})$. In theory, we require $\mathbf{H}_k$ to be any Hessian approximation with a spectrum bounded above and below uniformly for all k. In practice, we update $\mathbf{H}_k$ via L-SR1 updates [Byrd et al., 1994]. By implementing a reduced-memory version, we need not store an explicit approximate Hessian, thus greatly reducing the memory cost and significantly accelerating the matrix-vector products in our model. Pseudocode for TROPHY is provided in Algorithm 4.

5.4 Test Problems and Implementations

Our initial implementation of TROPHY is written in Python. To validate the algorithm, we focus on a well-known optimization test suite and a problem relating to climate modeling. In all cases, the algorithms terminate when one of the following conditions are met: (1) the first-order condition is satisfied, namely, $\|\nabla f^P(\mathbf{x}_k)\| < \epsilon_{\text{tol}}$; (2) the TR radius is smaller than machine precision, namely, $\delta_k < \epsilon_{\text{machine}}$; or (3) the first two conditions have not been met after some maximum number of iterations. Condition 1 is a success whereas conditions 2 and 3 are failed attempts. We describe the problem setup and implementation considerations in the current section and then discuss results in Section 5.5.

Algorithm 4: TROPHY

Initialize $0 < \eta_{\text{good}} \leq \eta_{\text{great}} < 1$, $\omega \in (0,1)$, $\gamma_{\text{inc}} > 1$, $\gamma_{\text{dec}} \in (0,1)$, $\Delta_{\text{prec}} \in (0,1)$, forcing seq. $\{r_k\}$.
Choose initial $\delta_0 > 0$, $\mathbf{x}_0 \in \mathbb{R}^n$.
$\theta_0 \leftarrow 0, p_k \leftarrow 0, k \leftarrow 0,$ failed $\leftarrow$ FALSE
while some stopping criterion not satisfied do
  Construct model m_k.
  (Approximately solve) (5.2) to obtain $\mathbf{s}_k$.
  Compute $\tilde{\rho}_k$ as in (5.8).
  if $\tilde{\rho}_k > \eta_{\text{good}}$ **(successful iteration) then**
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{s}_k$.
    if $\tilde{\rho}_k > \eta_{\text{great}}$ **(very successful iteration) then**
      $\delta_{k+1} \leftarrow \gamma_{\text{inc}} \delta_k$.
    end
  else
    if not failed **then**
      Compute θ_k as in (5.9).
      failed $\leftarrow$ TRUE.
    end
    if (5.10) **holds or** $\delta_k \geq \Delta_{\text{prec}}$ **then**
      $\delta_{k+1} \leftarrow \gamma_{\text{dec}} \delta_k$.
    else
      $p_{k+1} \leftarrow p_k + 1$.
      Compute θ_k as in (5.9).
      $\delta_{k+1} \leftarrow \delta_k$.
    end
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$.
  end
  $k \leftarrow k + 1$.
end

5.4.1 CUTEst

Our first example used the CUTEst set [Gould et al., 2015], which is well known within the optimization community and offers a variety of problems that are challenging to solve. Each problem is given in a Standard Input Format [Conn et al., 2013] file that is passed to a decoder from which Fortran subroutines are generated. The problems can be built directly by using single or double precision, making the set useful for mixed-precision comparison.

5.4.1.1 Python Implementation of CUTEst:

The PyCUTEst package [Fowkes and Roberts, 2018] serves as an interface between Python and CUTEst's Fortran source code. The problems are compiled via the interface; then Python scripts are generated and cached for subsequent function calls. Although CUTEst natively supports both single- and double-precision evaluations, at the time of writing, the single-precision implementations are concealed from the PyCUTEst API.

To access single-precision evaluations, we used PREDUCER [Hückelheim, 2019], a Python script written to compare the effect of round-off errors in scientific computing. PREDUCER parses Fortran source code and downcasts double data types to single. To allow for its use in existing code, all single data types are recast to double after function/gradient evaluation but before returning to the calling program. Because of the overhead associated with casting operations, we do not expect improvements in computational time. However, performance gains in terms of both accuracy and a reduction in the number of adjusted function calls for an iterative algorithm should be realized. We built the double-precision functions, too, and wrapped both functions to pass as a unified handle to TROPHY.

For the subset of unconstrained problems with dimension less than or equal to 100, we ran TROPHY with single/double switching along with the same TR method using only single or double precision. Our first-order stopping criterion was $\|\nabla f^P(\mathbf{x}_k)\| < 10^{-5}$, and the maximum number of allowable iterations was 5,000. We show results for problems solved by at least one TR in Sec. 5.5.

5.4.1.2 Julia Implementation of CUTEst:

The Julia programming language supports variable-precision floating-point data types. More precisely, it allows users to specify the number of bits used in the mantissa and expands memory for the exponent as necessary to avoid overflow. This is in contrast to the IEEE 754 standard that uses 11 (5), 24 (8), and 53 (11) bits for the mantissa (exponent) of half-, single-, and double-precision floats, respectively. In practice, one can assign enough bits for the exponent to avoid dynamic

reallocation.

To exploit variable precision, we hand-coded several of the unconstrained CUTEst objectives in Julia and then computed gradients with forward-mode automatic differentiation (AD) using the ForwardDiff.jl package [Revels et al., 2016]. We wrote a Julia port that allows us to call this code from Python for use in TROPHY. We opted to use forward-mode AD for its ease of implementation. In all cases used, the hand-coded Julia objective and AD gradient were compared against the Fortran implementation and found to be accurate.

We compared TROPHY with TR methods using a fixed precision of half, single, and double precision (11, 24, and 53 bits, respectively). We used the same first-order condition of $\|\nabla f(\mathbf{x}_k)\| < 10^{-5}$ but allowed this implementation to run only for 1,000 iterations. For TROPHY, we used several precision-switching sets: {24, 53}, {11, 24, 53}, {8, 11, 17, 24, 53}, and {8, 13, 18, 23, 28, 33, 38, 43, 48, 53}. The third set of precisions was motivated by the number of mantissa bits in bfloat16, fp16, fp24, fp32, and fp64, respectively. The last set increased the number of bits in increments of 5 up to double-precision.

5.4.2 Multiple Doppler Radar Wind Retrieval:

We also looked at a data assimilation problem for retrieving wind fields for convective storms from Doppler radar returns. Shapiro and Potvin [Shapiro et al., 2009, Potvin et al., 2012] proposed a method for doing so that optimizes a cost functional based on vertical vorticity, mass continuity, field smoothness, and data fidelity, among others. Although the function calls are fairly simple, the wind field must be reconciled on a 3-D grid over space, each with an x, y, and z component. For a $39 \times 121 \times 121$ grid, there are $1,712,997$ variables. Therefore, reducing computational, storage, and communication costs where possible is paramount.

Our work centered on the PyDDA package [Jackson et al., 2020], which was written to solve the aforementioned problem. We amended the code in two significant ways. First, to improve efficiency, we rewrote portions of the code to use JAX, an automatic differentiation package using XLA that exploits efficient computation on GPUs [Bradbury et al., 2018]. Since JAX natively

supports single precision on CPUs and can be recast to half and double as desired, it nicely serves as a proof of concept on a real application. Second, we modified the solver to use TROPHY rather than the SciPy implementation of L-BFGS.

Once again, we compared TROPHY against single- and double-precision TR methods. TROPHY switched among half, single, and double precision. To avoid overflow initially for half-precision, we warm started the algorithm by providing it with the tenth iterate from the double-precision TR method, i.e., $\mathbf{x}_{10}$. We perturbed this initial iterate 10 times and used the perturbed vectors as the initial guess for each algorithm (including double TR). We measured the average performance when solving each problem to different first-order conditions: $\|\nabla f(\mathbf{x}_k)\| < 10^{-3}$ and $\|\nabla f(\mathbf{x}_k)\| < 10^{-6}$. The maximum number of allowable iterations was 10,000.

5.5 Experimental Results

We display results across the CUTEst set using data and performance profiles [Dolan and Moré, 2002, Moré and Wild, 2009]. For a given metric, performance profiles help determine how a set of solvers, $\mathcal{S}$, performs over a set of problems, $\mathcal{P}$. The value $v_{ij} > 0$ denotes a particular metric (say, the final gradient norm) of the jth solver on problem i. We can then consider the performance of each solver in relation to the solver that performed best, that is, the one that achieved the smallest gradient norm. The **performance ratio** is defined as

$$r_{ij} = \frac{v_{ij}}{\min_j\{v_{ij}\}}. \tag{5.13}$$

Smaller values of r_{ij} are better since they are closer to optimal. The performance ratio was set to ∞ if the solver failed to solve the problem. We can evaluate the performance of a solver by asking what percentage of the problems are solved within a fraction of the best. This is given by the **performance profile**,

$$h_j(\tau) = \frac{\sum_{i=1}^{N} \mathcal{I}_{\{r_{ij} \leq \tau\}}}{N},$$

where $N = |\mathcal{P}|$ (the cardinality of $\mathcal{P}$) and $\mathcal{I}_{\{A\}}$ is the indicator function such that $\mathcal{I}_{\{A\}} = 1$ if A is true and 0 otherwise. Hence, better solvers have profiles that are above and to the left of the

others.

Motivated by the computational models in [Molka et al., 2010] and [Kestor et al., 2013], we assume that the energy efficiency of single precision is between 2 and 3.6 times higher than double precision [Fagan et al., 2016, Galal and Horowitz, 2011]. The storage and communication have less optimistic savings since we expect the cost of both to scale linearly with the number of bits used in the mantissa. Accordingly, we focus primarily on the model where half- and single-precision evaluations cost 1/4 and 1/2 that of a double evaluation, respectively. This gives a conservative estimate for energy cost and a favorable one for execution time. For a given problem and solver, we define **adjusted calls**:

$$\text{Adj. calls} = \sum_{p \in \{0,1,\ldots,P\}} \frac{(\#\text{ bits for prec. } p) \times (\#\text{ func. calls at prec. } p)}{\#\text{ bits in prec. } P}. \tag{5.14}$$

Figures 5.1 and 5.2 show performance profiles for the Python and Julia implementations of CUTEst, respectively. All CUTEst problems had their first-order tolerance set to 10^{-5}. Working from right to left in both images, we can see that the first-order condition is steady across methods provided that double-precision evaluations are ultimately available to the solver. When limited to half (11 bits) or single (24 bits), the performance suffers, and a number of problems cannot be solved. For the number of iterations in Python, we see that TROPHY and the double TR method perform comparably. The Julia implementation shows that the iterations count suffers when using low precision or TROPHY with many precision levels available for switching. This behavior is expected for low precision since the solver may never achieve the desired accuracy and hence runs longer, and for TROPHY since each precision switch requires a full iteration. For example, if 10 precision levels are available, TROPHY will take at least 10 iterations to complete. This limits the usefulness of the method on small to medium problems and problems where the initial iterate is close to the final solution. As anticipated, TROPHY shows a distinct advantage for adjusted calls. The one exception is when there are many precision levels to cycle through, for the same reason as above. Although the initial iterate might be close to optimal, the algorithm must still visit all precision levels before breaking. The fact is made worse since each time the precision switches, two evaluations at the highest precision are required. Using two or three widely spaces precision levels

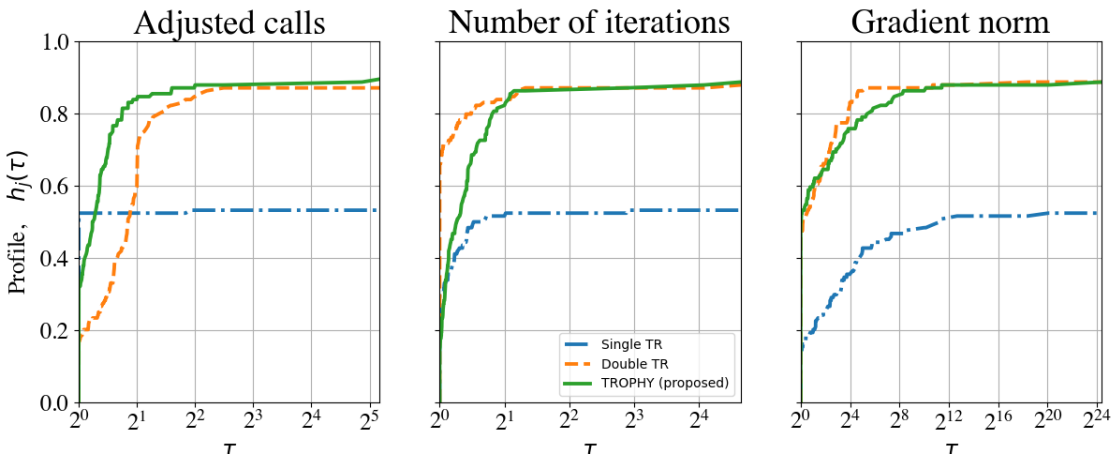

Figure 5.1: Performance profiles for Python implementation of unconstrained CUTEst problems of dimension < 100 solved to first-order tolerance of 10^{-5}. Standard single and double TR methods compared against TROPHY using single/double switching.

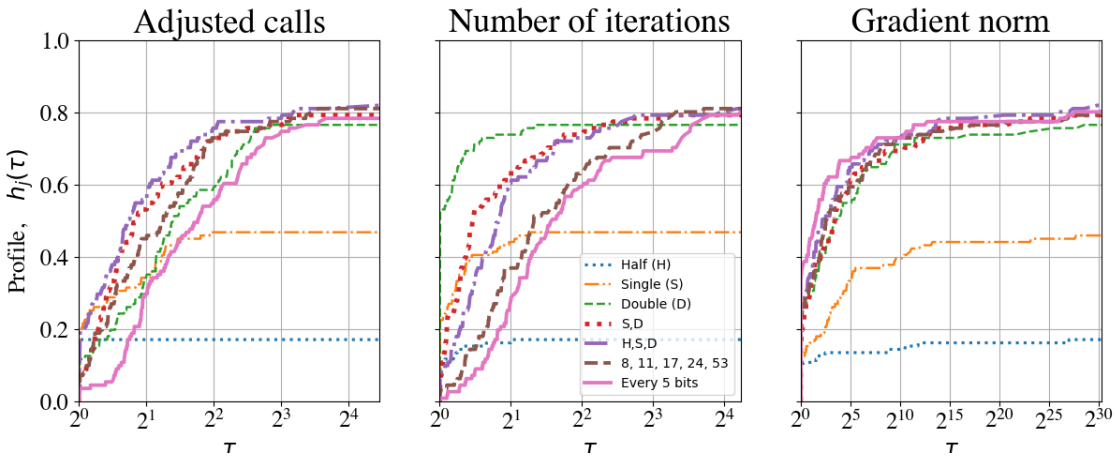

Figure 5.2: Performance profiles for Julia implementation of unconstrained CUTEst problems of dimension < 100 solved to first-order tolerance of 10^{-5}. Half, single, and double are standard TR methods using corresponding precision. "S,D", "H,S,D", "8,11,17,24,53", and "Every 5 bits" are TROPHY implementations using different precision regimes. "Every 5 bits" starts at 8 bits and increases to 53 bits in increments of 5 bits. A finer precision hierarchy does not imply better performance.

yields strong results for the CUTEst set.

The wind retrieval example shows similar results. The switching criteria used here differs slightly from the one presented in (5.10). Specifically, a baseline θ_{p_k} is set after the first failed iteration at the current precision level. The model predicted reduction (pred_k) is compared to the baseline θ_{p_k} for successive failures. If pred_k is small compared to the baseline θ_{p_k}, then the precision

Table 5.1: Average performance over ten initializations for single-precision TR, double-precision TR, and TROPHY on PyDDA wind retrieval example. Adjusted calls indicate improved computational efficiency. Half, single, and double costs are 1/4, 1/2, and 1 for linear and 1/16, 1/4, and 1 for quadratic adjustments, respectively. Problem solved to $\|\nabla f(x_{\text{final}})\|_2 < 10^{-3}$ above.

Tolerance $\|\nabla f\| < 10^{-3}$	Half calls	Single calls	Double calls	**Adj. calls (linear)**	**Adj. calls (quad.)**	f_{final}	$\|\nabla f_{\text{final}}\|$
Single	-	3411	-	**1706**	**853**	5.3×10^{-3}	9.5×10^{-4}
Double	-	-	1877	**1877**	**1877**	4.5×10^{-3}	9.1×10^{-4}
TROPHY	465	1898	6	**1071**	**510**	4.7×10^{-3}	9.4×10^{-4}

is increased. We do not expect this to significantly change the qualitative results or behavior of the method for this test case. We included two "adjusted call" columns: one for a linear decay adjustment (memory and communication as above) and the other for quadratic decay (reduction in energy consumption). We originally iterated until $\|\nabla f(\mathbf{x}_k)\| < 10^{-6}$ but observed that the single TR method failed to converge. Consequently, we loosened the stopping criterion as far as possible while maintaining the correct qualitative behavior of the solution. TROPHY outperformed the standard (double-precision TR) method in all cases, reducing the number of adjusted calls by 17% to 73%.

Our results show a promising reduction in the relative cost over naive single or double TR solvers. We expect that for many problems where function evaluations dominate linear algebra costs for the TR subproblem, our time to solve will greatly benefit from the method.

5.6 Conclusion and Future Work

In this paper we introduced TROPHY, a TR method that exploits variable-precision data types to lighten the computational burden of expensive function/gradient evaluations. We illustrated proof of concept for the algorithm by implementing it on the CUTEst set and PyDDA. The full benefit of our work has not yet been realized. We look forward to implementing simi-

Table 5.2: Problem solved to $\|\nabla f(x_{\text{final}})\|_2 < 10^{-6}$ accuracy. The single-precision TR method failed to converge with the TR radius falling below machine precision.

Tolerance $\|\nabla f\| < 10^{-6}$	Half calls	Single calls	Double calls	**Adj. calls (linear)**	**Adj. calls (quad.)**	f_{final}	$\|\nabla f_{\text{final}}\|$
Single	-	∞	-	**FAIL**	**FAIL**	9.9×10^{-7}	3.9×10^{-6}
Double	-	-	5283	**5283**	**5283**	1.8×10^{-7}	9.6×10^{-7}
TROPHY	465	7334	601	**4384**	**2464**	1.9×10^{-7}	9.7×10^{-8}

.

lar tests on hardware that can realize the full benefit of lower energy consumption and reduced memory/communication costs and ultimately shorten the time to solution. This will be especially beneficial for large scale climate models.

We would also like to incorporate mixed precision into line-search methods given their popularity in quasi-Newton solvers. By incorporating the same ideas into highly optimized algorithms such as the SciPy implementation of L-BFGS, we could easily deploy mixed precision to a wide population, dramatically reducing computational loads. Although TR methods are, computationally speaking, more appropriate for expensive-to-evaluate objectives, there is no reason the same ideas cannot be extended if practitioners prefer them.

Chapter 6

A Study of Scalar OPMs for use in Magnetoencephalography without Shielding

6.1 Introduction

Magnetoencephalography (MEG) is a non-invasive method to image brain function with high spatial and temporal resolution [Supek and Aine, 2016]. Electrical currents in the brain give rise to magnetic fields that are detectable on the exterior of the head. Exploring these MEG recordings can provide insight into the functionality and disorders of the brain.

Since the early 1990s, superconducting quantum interference devices (SQUIDs) have been stalwarts in the world of MEG [Cohen, 1972]. While the technology is mature and well understood, several practical considerations are incentivizing alternate technologies. To start, SQUIDs must be cooled requiring a bath of liquid helium which is expensive and subject to commodity shortages. Also, the Dewar wall thickness sets a limit on the proximity of sensors to the scalp. Since the magnetic fields induced by neural currents in the brain are remarkably weak (hundreds of femto-Tesla at the surface of the scalp), any additional separation between the source and the sensor adversely impacts the signal-to-noise ratio (SNR) and the ability to localize brain activity. Furthermore, to fit all users, the rigid Dewars are sized to fit large human heads, severely limiting their usefulness on young children [Baillet, 2017].

Optically-pumped magnetometers (OPMs) are atomic sensors that, under certain conditions, are capable of matching sensitivities observed with SQUIDs [Dang et al., 2010]. They operate without cryogenics, eliminating the need for bulky Dewars, and can be placed within millimeters of the scalp. OPMs open the door for wearable MEG systems that conform to a subject's head and

allow for free and natural movement during scanning [Boto et al., 2018]. To rival the sensitivities of SQUIDS, OPMs must operate in the spin-exchange relaxation-free (SERF) regime [Kominis et al., 2003, Allred et al., 2002]. However, the narrow linewidths of SERF OPMs [Allred et al., 2002, Happer and Tang, 1977] limit the operating dynamic range to a few nano-Tesla or near zero-field background. This means magnetically shielded rooms, and often additional large coils, are necessary to cancel any ambient fields and reduce interference that can severely limit usefulness in hospital environments.

An alternative to zero-field OPMs are total-field or scalar optically-pumped magnetometers, which are being developed in microfabricated packages [Schwindt et al., 2004] or with high sensitivities [Sheng et al., 2013]. While they were first developed in the 1950s [Kastler, 1950] and used for biomagnetic measurements since the 1970s [Livanov et al., 1978], only recently have they reached noise floors sufficient for MEG applications and been demonstrated in small packaged sensors [Limes et al., 2020c]. We investigate the use of scalar OPMs for MEG in this paper.

For total field magnetometers, the sensor orientation does not affect the measured field value but only the achieved sensitivity. For that reason, total field magnetometers realize higher common-mode rejection ratios than their vector counterparts [Limes et al., 2020b]. This allows for the detection of very small signals in the presence of large background fields, making them well suited for use in unshielded environments [Zhang et al., 2020a, Perry et al., 2020]. Since scalar OPMs are less sensitive to their orientation, they are commonly used on moving or vibrating platforms for unexploded ordinance detection (UXO), geophysical surveying, and exploration [Du et al., 2017, David et al., 2004, Nabighian et al., 2005]. Work has begun on detecting brain signals in unshielded environments [Limes et al., 2020c, Zhang et al., 2020b]. Recently, an exciting proof-of-concept portable MEG system was constructed for use in Earth's field with scalar magnetometers [Limes et al., 2020a, Limes et al., 2020b], exhibiting their suitability for biomagnetic applications.

In this paper, we provide numerical characterizations of localization accuracy based on the variability in sensor count, sensitivity, and the fidelity of the forward model using a single dipolar source in a conducting sphere and large ambient field. We also investigate the influence of back-

ground (or bias) field angle on the localization accuracy. We illustrate the validity of our method by localizing a current dipole using a dry phantom and a scalar gradiometer array.

6.2 Models and algorithms

Following the general MEG method for dipole localization in a conducting sphere [Sarvas, 1987], the sources are modeled as *current dipoles* [Cohen and Hosaka, 1976] specified by their position, orientation, and magnitude of directional current flow. Current dipoles give rise to magnetic field patterns but are distinct from those of *magnetic dipoles* which can be thought of as small bar magnets. We note that fields from current dipoles decay with the distance squared in contrast to magnetic dipoles that decay with the distance cubed. The dipolar source is constrained to a depth consistent with the cerebral cortex and represents a neural bundle firing in unison. The terms ambient, background, and bias field are used synonymously.

In this paper, we focus on static analysis but similar methods can be used for time series measurements through averaging, peak finding, Fourier transforms, and other processing tools. The aim of localization is to determine the dipole location and moment that best conforms to the measured data. In what follows, we describe a forward model, explain how it relates to the optimization problem we use for localization, and then outline an algorithm to find sources of neuronal activity.

6.2.1 Forward model

A forward model characterizes the magnetic field of a known source throughout space. We consider the conducting sphere model of radius 9.1 cm provided in [Sarvas, 1987] and elaborated on in [Mosher et al., 1999]. The authors show that the magnetic field at point $\mathbf{r}$ due to a dipole located at point $\mathbf{p}$ can be written as the product of a *solution kernel*, $\mathbf{L}(\mathbf{r}, \mathbf{p})$, and the dipole moment, $\mathbf{q}$. That is, for a dipole $(\mathbf{p}, \mathbf{q})$, the magnetic field at point $\mathbf{r}$ is

$$\mathbf{B}(\mathbf{r}, \mathbf{p}, \mathbf{q}) = \mathbf{L}(\mathbf{r}, \mathbf{p})\mathbf{q}. \tag{6.1}$$

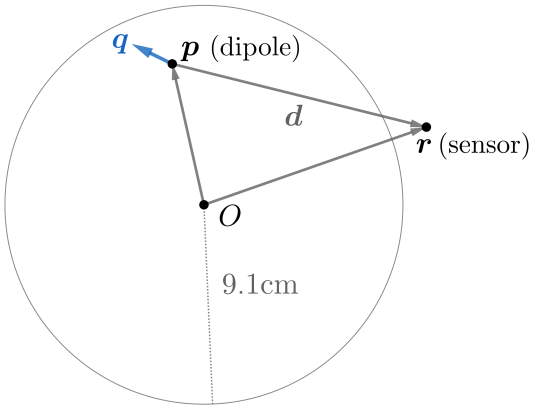

Figure 6.1: A current dipole at point $\mathbf{p}$ with moment $\mathbf{q}$ will generate a field at sensor location $\mathbf{r}$ according to Eq. (6.1) and (6.2). The displacement vector is given by $\mathbf{d} = \mathbf{r} - \mathbf{p}$.

The solution kernel, $\mathbf{L}$, is given by

$$\mathbf{L}(\mathbf{r}, \mathbf{p}) = \frac{\mu_0}{4\pi} \left[\frac{\nabla_{\mathbf{p}} \Phi \mathbf{r}^T - \Phi \mathbf{I}}{\Phi^2} \mathbf{C}_{\mathbf{p}} \right] \qquad \text{with} \qquad \mathbf{C}_{\mathbf{p}} = \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix}, \tag{6.2}$$

where p_x, p_y, and p_z are the components of $\mathbf{p}$, and $\mathbf{I}$ is the identity matrix. Note that $\Phi \in \mathbb{R}$, $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$, $\mathbf{B}$, $\nabla_{\mathbf{p}} \Phi \in \mathbb{R}^3$, and $\mathbf{C}_{\mathbf{p}}$, $\mathbf{I}$, $\mathbf{L}(\mathbf{r}, \mathbf{p}) \in \mathbb{R}^{3\times 3}$. Defining relative position as $\mathbf{d} = \mathbf{r} - \mathbf{p}$, the scalar function Φ and its gradient with respect to $\mathbf{p}$ are given by

$$\begin{aligned} \Phi(\mathbf{r}, \mathbf{p}) &= \|\mathbf{d}\| \left(\|\mathbf{d}\| \, \|\mathbf{r}\| + \|\mathbf{r}\|^2 - \mathbf{p}^T \mathbf{r} \right) \\ \nabla_{\mathbf{p}} \Phi(\mathbf{r}, \mathbf{p}) &= \left(\frac{\|\mathbf{d}\|^2}{\|\mathbf{r}\|} + \frac{\mathbf{d}^T \mathbf{r}}{\|\mathbf{d}\|} + 2\|\mathbf{d}\| + 2\|\mathbf{r}\| \right) \mathbf{r} - \left(\|\mathbf{d}\| + 2\|\mathbf{r}\| + \frac{\mathbf{d}^T \mathbf{r}}{\|\mathbf{d}\|} \right) \mathbf{p}. \end{aligned} \tag{6.3}$$

In the absence of fields from other magnetic sources, the measured signal, y_i, of a scalar sensor at location $\mathbf{r}_i$ will be the norm of the superposition of the ambient field, $\mathbf{a}$, and the dipolar field, $\mathbf{B}$, plus noise, η_i. That is, $y_i = \|\mathbf{B}(\mathbf{r}_i, \mathbf{p}, \mathbf{q}) + \mathbf{a}\| + \eta_i$. We consider cases where the ambient field is sufficiently large such that y_i is always positive. Since the background fields we consider are many orders of magnitude larger than the biomagnetic signals we aim to measure, i.e., micro vs. femto-Telsa, we can treat scalar sensor readings as a superposition of the dipolar and ambient field,

$\mathbf{B}(\mathbf{r}_i, \mathbf{p}, \mathbf{q}) + \mathbf{a}$, projected onto the ambient field's direction. That is, letting $\hat{\mathbf{a}}$ be the ambient field's unit vector, writing $\mathbf{b}_i = \mathbf{B}(\mathbf{r}_i, \mathbf{p}, \mathbf{q})$ and assuming $\|\mathbf{b}_i\| \ll \|\mathbf{a}\|$, the noiseless scalar magnetic field at point $\mathbf{r}_i$ is

$$\begin{aligned}
\|\mathbf{a} + \mathbf{b}_i\| &= \sqrt{\|\mathbf{a}\|^2 + 2\mathbf{a}^T\mathbf{b}_i + \|\mathbf{b}_i\|^2} \\
&\approx \sqrt{\|\mathbf{a}\|^2 + 2\mathbf{a}^T\mathbf{b}_i} \\
&= \|\mathbf{a}\|\sqrt{1 + 2\hat{\mathbf{a}}^T\mathbf{b}_i/\|\mathbf{a}\|} \\
&\approx \|\mathbf{a}\|\left(1 + \hat{\mathbf{a}}^T\mathbf{b}_i/\|\mathbf{a}\|\right) \quad \text{(Taylor series)} \\
&= \|\mathbf{a}\| + \hat{\mathbf{a}}^T\mathbf{B}(\mathbf{r}_i, \mathbf{p}, \mathbf{q}) \stackrel{\text{def}}{=} f_i. \qquad (6.4)
\end{aligned}$$

Thus for an MEG array with M sensors at points $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_M$, the forward model is

$$\mathbf{f} = \begin{bmatrix} \hat{\mathbf{a}}^T\mathbf{L}(\mathbf{r}_1, \mathbf{p}) \\ \hat{\mathbf{a}}^T\mathbf{L}(\mathbf{r}_2, \mathbf{p}) \\ \vdots \\ \hat{\mathbf{a}}^T\mathbf{L}(\mathbf{r}_M, \mathbf{p}) \end{bmatrix} \mathbf{q} + \|\mathbf{a}\|\mathbf{1} \qquad (6.5)$$

where $\mathbf{1}$ is the vector of ones. This approximation is an affine function of $\mathbf{q}$ paving the way for a simplified localization algorithm. Upon testing, we found that the difference between our approximation and the precise forward model, i.e., $\left|f_i - \|\mathbf{a} + \mathbf{b}_i\|\right|$, was below the noise floor used and therefore negligible.

6.2.2 Gradiometry

Since we are interested in detecting weak brain signals dominated by large background fields, filtering is paramount. Outside of laboratory settings, we expect the ambient field to vary over space and time for many reasons including solar winds, passing vehicles, proximity to ferrous materials, etc. The presence of magnetic fields from other unrelated sources necessitates gradiometry.

For simulations and experiment, we used two scalar magnetometers oriented radially as our gradiometers. A *primary* sensor at $\mathbf{r}^P$ was mounted on the surface of the conducting sphere and was

intended to measure brain magnetic fields. The *secondary* sensor at $\mathbf{r}^S$ was employed to sense the background field but should, in principle, be far enough away to not detect biomagnetic signals. The superscripts are not exponents. The distance or baseline between primary and secondary sensors is 4 cm in the radial direction. For this gradiometer arrangement, the forward model corresponding to measured difference data, $y^P - y^S$, is

$$\begin{aligned} f^P - f^S &= (\hat{\mathbf{a}}^T \mathbf{L}(\mathbf{r}^P, \mathbf{p})\mathbf{q} + \|\mathbf{a}\|) - (\hat{\mathbf{a}}^T \mathbf{L}(\mathbf{r}^S, \mathbf{p})\mathbf{q} + \|\mathbf{a}\|) \quad (6.6) \\ &= \hat{\mathbf{a}}^T \big(\mathbf{L}(\mathbf{r}^P, \mathbf{p}) - \mathbf{L}(\mathbf{r}^S, \mathbf{p})\big)\mathbf{q}, \quad (6.7) \end{aligned}$$

which has no dependence on the magnitude of the bias field. The full gradiometer forward model can be written as

$$\mathbf{f}^G(\mathbf{p}, \mathbf{q}) = \begin{bmatrix} \hat{\mathbf{a}}^T \left(\mathbf{L}(\mathbf{r}_1^P, \mathbf{p}) - \mathbf{L}(\mathbf{r}_1^S, \mathbf{p})\right) \\ \hat{\mathbf{a}}^T \left(\mathbf{L}(\mathbf{r}_2^P, \mathbf{p}) - \mathbf{L}(\mathbf{r}_2^S, \mathbf{p})\right) \\ \vdots \\ \hat{\mathbf{a}}^T \left(\mathbf{L}(\mathbf{r}_M^P, \mathbf{p}) - \mathbf{L}(\mathbf{r}_M^S, \mathbf{p})\right) \end{bmatrix} \mathbf{q} = \mathbf{A}(\mathbf{p})\,\mathbf{q}. \quad (6.8)$$

The superscript denotes the gradiometer arrangement forward model. For the matrix $\mathbf{A}(\mathbf{p})$, we treat the sensor locations as fixed henceforth. For primary and secondary sensor readings $\mathbf{y}^P$ and $\mathbf{y}^S$, respectively, the corresponding gradiometer measurement vector is given by

$$\mathbf{y}^G = \mathbf{y}^P - \mathbf{y}^S. \quad (6.9)$$

With a forward model and data at our disposal, we can now formulate the optimization problem to localize a dipole.

6.2.3 Optimization problem and algorithms

Given the forward model $\mathbf{f}^G$ from (6.8), the goal is to find a dipole $(\mathbf{p}, \mathbf{q})$ that best explains observed MEG data $\mathbf{y}^G$. We formulate this mathematically with the nonlinear least squares problem

$$\min_{\mathbf{p},\mathbf{q}} \|\mathbf{f}^G(\mathbf{p}, \mathbf{q}) - \mathbf{y}^G\|^2 = \min_{\mathbf{p},\mathbf{q}} \|\mathbf{A}(\mathbf{p})\,\mathbf{q} - \mathbf{y}^G\|^2 \quad (6.10)$$

which is convex in $\mathbf{q}$ but not in $\mathbf{p}$. To address non-convexity, we break the problem into two parts as outlined in [Ilmoniemi and Sarvas, 2019]. First, we find a point that is in the global optimum's basin of attraction as a warm start, then employ an iterative nonlinear solver.

For the warm start, we grid the variable $\mathbf{p}$ over the sphere's interior at K locations. Spacing of 2 cm is typically sufficient although we used 1.1 cm here. For each grid value of $\mathbf{p}$, the optimal value of $\mathbf{q}$ is easily found by solving an ordinary least squares problem, so $\mathbf{q}$ does not need to be discretized. In particular, at a fixed grid point $\mathbf{p} = \mathbf{p}_k$ with $k \in \{1, \ldots, K\}$, the optimal solution $\mathbf{q}_k^*$ to problem (6.10) is

$$\mathbf{q}_k^* = \left(\mathbf{A}(\mathbf{p}_k)^T \mathbf{A}(\mathbf{p}_k)\right)^{-1} \mathbf{A}(\mathbf{p}_k)^T \mathbf{y}^G. \tag{6.11}$$

After finding location/moment least square pairs, $(\mathbf{p}_k, \mathbf{q}_k^*)$, for all K discrete locations, the optimal index o is given by

$$o = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmin}} \; \|\mathbf{A}(\mathbf{p}_k)\, \mathbf{q}_k^* - \mathbf{y}^G\|^2. \tag{6.12}$$

Hence, we use pair $(\mathbf{p}_o, \mathbf{q}_o^*)$ as a warm start for a continuous optimization algorithm. In our simulations, we used L-BFGS [Liu and Nocedal, 1989] as the continuous optimization algorithm, which is a quasi-Newton method, although others could be used. L-BFGS iterates through the space of possible values for $(\mathbf{p}, \mathbf{q})$ to find a fit minimizing residual error.

6.3 Numerical Experiments

We ran a variety of simulations to understand limitations and requirements for scalar OPM use in MEG systems with strong ambient fields. In all simulations we use gradiometry. As such, the sensor count reflects the number of gradiometers, not the number of sensors (each gradiometer has a primary and secondary sensor). We provide multiple curves in each experiment indicating performance for different *relative dipole strengths* (RDS) defined by

$$\text{RDS} = \frac{\text{Dipole strength (nAm)}}{\left(\text{Sensor sensitivity (fT}/\sqrt{\text{Hz}}\,)\right)\left(\sqrt{\text{Bandwidth (Hz)}}\right)}. \tag{6.13}$$

This figure of merit is chosen to easily relate the results to a variety of magnetometers and experiments with different noise, bandwidth, and dipole source strengths. Throughout this paper,

RDS		**Sensor sensitivity ($\mathbf{fT/\sqrt{Hz}}$)**			
		0.1	1.0	10.0	100.0
Bandwidth (Hz)	1	100.000	10.000	1.000	0.100
	5	44.721	4.472	0.447	0.045
	10	31.623	3.162	0.316	0.032
	50	14.142	1.414	0.141	0.014
	100	10.000	1.000	0.100	0.010
	500	4.472	0.447	0.045	0.004
	1000	3.162	0.316	0.032	0.003

Table 6.1: Relative dipole strength (RDS) in units of nAm/fT for a physiological neuronal current dipole of magnitude 10 nAm over different bandwidths and sensitivities.

bandwidth refers to the signal's frequency content and the corresponding filter used for signal extraction rather than sensor bandwidth which is related to the magnetometer's specifications. Signals with narrow bandwidths are easily recovered with band-pass filters. On the other hand, a wide-band signal requires an extensive filter that includes noise across a broader spectrum effectively lowering the SNR. Although we focus on bandwidth, noise reduction is also accomplished by averaging; taking the mean of 100 measurements is equivalent to reducing the bandwidth by a factor of 10. For clarity, we focus primarily on bandwidth but understand that averaging can be used instead. A typical physiological current dipole generated by synchronous neural activity is 10 nAm [Hämäläinen et al., 1993]. In our experiments, we used a scalar OPM sensor with a noise floor of 70 fT/$\sqrt{\text{Hz}}$ [Gerginov et al., 2020] and measurement bandwidth of 100 Hz. For these OPM parameters, we then varied the dipole strength for simplicity in our simulation. A 10 nAm dipole gives an RDS of 0.014 nAm/fT. See Table 6.1 for conversions between bandwidths and sensitivities.

Each experiment consists of 10,000 dipoles randomly drawn from a specified volume. All dipoles in this paper have a random orientation that is tangential to the spherical surface. To generate scalar sensor data for a "true" simulated dipole $(\mathbf{p}_{\text{T}}, \mathbf{q}_{\text{T}})$, we first calculate the dipolar field from Equations 6.1, 6.2, and 6.3 then add it to the ambient field. We take the magnetic field norm at all sensor locations (both primary and secondary) to give noiseless sensor readings. A random Gaussian vector with zero mean and standard deviation of 700 fT (70 fT/$\sqrt{\text{Hz}}$ times $\sqrt{100\text{Hz}}$) is drawn and added to the noiseless measurement vector. Finally, we subtract the secondary from the

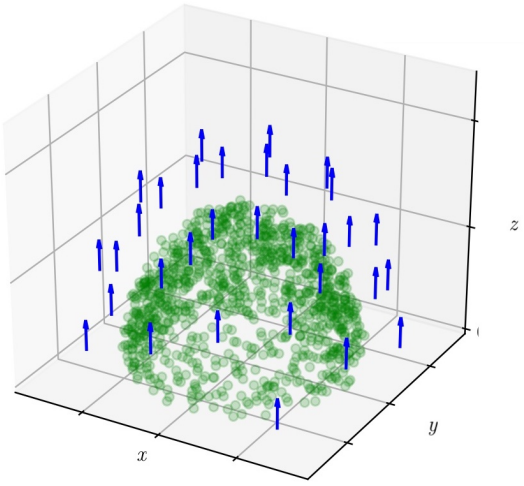

Figure 6.2: Typical 32 sensor arrangement using Fibonacci spiral with bias field along z-axis. Green dots are locations of simulated dipoles drawn randomly with surface depth of $2 - 3.5$ cm. Gridline spacing is 5 cm.

primary sensor readings to give the gradiometer measurement vector in (6.9). We can then employ the algorithm outlined in Section 6.2.3 with the generated gradiometer measurement vector as our input.

For each simulation, the algorithm returns an estimated dipole, $(\tilde{\mathbf{p}}, \tilde{\mathbf{q}})$, which is compared with the true dipole through localization error $\|\mathbf{p}_{\mathrm{T}} - \tilde{\mathbf{p}}\|$. Although we must estimate $\mathbf{q}$, we are not concerned with it and focus on dipole location error alone. We use median and other quantile based error over 10,000 simulations to evaluate accuracy.

Except for bias field orientation simulations, sensors were placed on the upper half of a hemisphere along a Fibonacci spiral [Vogel, 1979, González, 2010] which is a parametric curve giving nearly uniform coverage over a spherical surface. The lower boundary of the hemisphere is the xy-plane. We outline our numerical simulations in detail below. A typical sensor arrangement is shown in Figure 6.2. Since we assume scalar sensors, the orientation of the OPM is not important and the blue arrows denote the direction of the large bias field assumed at the location of the sensor.

6.3.1 Vector vs. scalar sensor comparison

Usually, MEG is recorded with directional sensors, measuring a single component of the magnetic field vector. We will denote them as "vector" sensors for simplicity, even though the full vector field is not measured. In contrast, scalar sensors measure the total magnitude of the field independent of orientation.

To give the reader a better idea of how scalar and vector sensors compare, Figure 6.3 shows the median and middle 50th quantile error for vector and scalar sensors, as well as their gradiometry counterparts. We varied the number of sensors and RDS. For each simulation, the same dipole-generated fields measured by all four sensor varieties. Vector sensors were oriented normal to the conducting sphere surface, detecting radial components of the magnetic field. For both scalar sensor varieties, we assume that the orientation of the background field is provided to the forward model. Dipoles were drawn uniformly at random within the upper half of the hemisphere at a depth of 2 – 3.5 cm to mimic activity on the cortex and with random tangential orientations.

Both vector sensor arrangements perform better than their scalar versions, independent of sensor count and dipole strength. Nevertheless, it can be seen that the difference is about a factor of two and that sensor count and SNR have a much larger effect on the localization error than the type of sensor.

6.3.2 Dependence on sensor count

Increasing the number of sensors ostensibly improves localization accuracy. Simulations were conducted to quantify the impact of increased sensor count. Using the same setup as detailed in Section 6.3.1, we varied the number of scalar gradiometers from 16 to 512 in powers of 2 and recorded the localization error for each dipole simulated.

Figure 6.4 illustrates localization accuracy as a function of sensor count. As intuition suggests, more sensors improve localization accuracy. It can be observed, however, that increasing the number of gradiometers does not improve performance for the 0.01 nAm/fT curve over the sensor counts

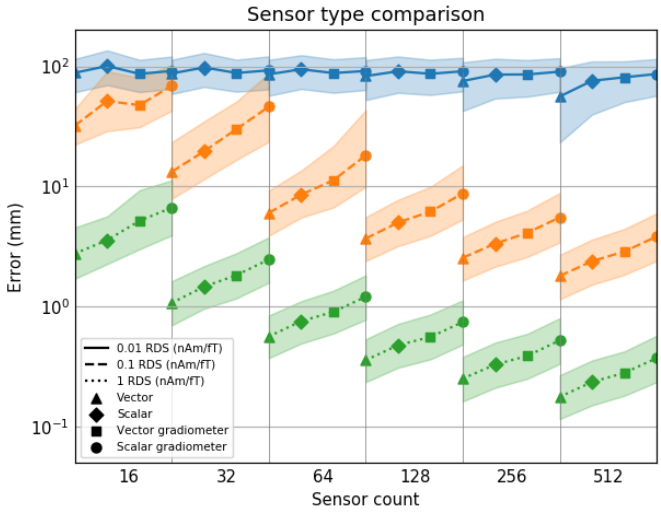

Figure 6.3: Each curve shows progression of error by sensor type. From left to right, sensor types are vector, scalar, vector gradiometer, and scalar gradiometer. Median localization error and middle 50th quantile error (shaded region) vs. sensor types. Each curve corresponds to a different relative dipole strength (RDS) of given sensor type and varies from a 16 to 512 sensor arrangement.

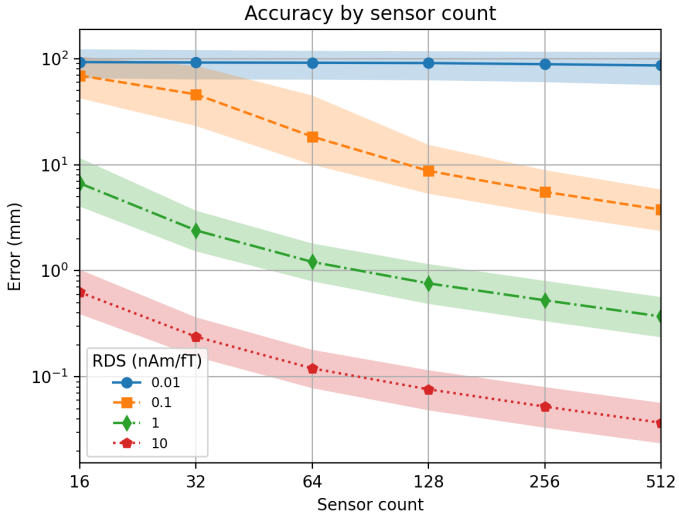

Figure 6.4: Median localization error (curve) and middle 50th quantile error of scalar gradiometers (shaded region) vs. sensor count for different RDS. Bias field oriented in positive z-direction.

considered. It is important to achieve a baseline sensitivity before adding more sensors since they will not give the desired improvement. Assuming a 10 nAm dipole and bandwidth of 10 Hz, a suitable sensitivity is approximately $80\,\mathrm{fT}/\sqrt{\mathrm{Hz}}$ and is currently attainable as demonstrated in Section 6.4. **We estimate that localization error of 1 cm can be reached for a 10 nAm dipole using 128 sensors with a noise floor of 10 $\mathrm{fT}/\sqrt{\mathrm{Hz}}$ and a bandwidth of 100 Hz. Likewise, 1 mm localization accuracy is achievable with the same noise floor and dipole strength by reducing the filter bandwidth to approximately 4 Hz**. We note that similar accuracy for a given dipole can be achieved by reducing bandwidth, increasing sensitivity, averaging measurements, or increasing sensor count. For example, 80 sensors at 1 Hz or 128 sensors at 40 Hz averaged over 100 measurements with the same noise floor will also achieve 1 mm accuracy. While these simulations omit many practical considerations, they give an idea of the system complexity needed to achieve reasonable localization results.

6.3.3 Bias field angle dependence

We now consider the case where a subject is undergoing MEG measurement in a spatially homogeneous and static bias field. Suppose the subject is free to rotate and move their head throughout the measurement process. By fixing a reference frame to the subjects head, it will appear as though the bias field changes its orientation over time. Does our ability to localize a dipole (fixed with respect to the subject's head) depend on its orientation relative to the bias field? Are certain orientations better or worse for localization in general? How does do uniformly oriented (all in the same direction) sensors arrays considered in this study compare to radially mounted arrays as found in SQUID setups?

We make a few simplifying assumptions. First, the forward model is correctly specified and accurate; this implies precise knowledge of sensor locations and orientation of the ambient field with respect to the subject's frame of reference at all points and times. Second, we assume the sensors are isotropically sensitive, that is, their sensitivities are independent of orientation with respect to the background field. In practice, sensors have dead-zones where the response of the magnetometer

drops to zero, but we do not consider such cases at present.

We also assume that the magnetometers are not sensitive enough to detect a signal from brain regions on the opposite side of the head. We therefore arrange a dense sensor array over a localized area of the head. Sixteen gradiometers are arranged on the hemisphere's surface in two concentric circles about the z-axis. The circles have azimuthal angles of 7.5° and 15° with gradiometers spaced every 45° in the polar direction. Dipoles were drawn from a region roughly below the sensor array at a mean depth of 2.5 cm. A typical sensor arrangement with a subset of simulated dipole locations is shown in Figure 6.5. For a simulated dipole, we attempted localization for different bias field orientations varied from 0° (along the z-axis) to 90° (along the x-axis) in 10° increments. For comparison to typical radial vector sensor arrangements, we also simulated radially mounted sensors in the same locations. All non-radial bias field orientations varied in the xz-plane without a y component. Figure 6.6 shows accuracy as a function of the bias field.

Based on our simulations, performance deteriorates slightly as the bias field differs from the axis of symmetry around which dipoles are drawn, at least when dipoles are tightly bunched. It is also worth noting that radial sensors perform slightly worse than 0° orientation. We believe this is due in part to the fact that dipoles close to the z-axis generate smaller projections on average along the radial direction and hence have lower RDS. We conclude that it will be possible to image with any bias field orientation and that the localization accuracy degrades by less than a factor of two between radial and tangential bias field.

6.3.4 Sensitivity to uncertainty in forward model: sensor locations and orientations

Localization error depends heavily on the accurate specification of a forward model. In practice, it is challenging to determine sensor location and orientation precisely, especially since an advantage of uncooled sensors is that they can be placed in conformal geometries individual to every subject. Accordingly, we'd like to understand how robust localization is to model misspecification. To accomplish this, we fixed a presumptive sensor array for use in the forward model. We then perturbed sensor locations and bias field orientation by adding Gaussian noise to give a *true* sensor

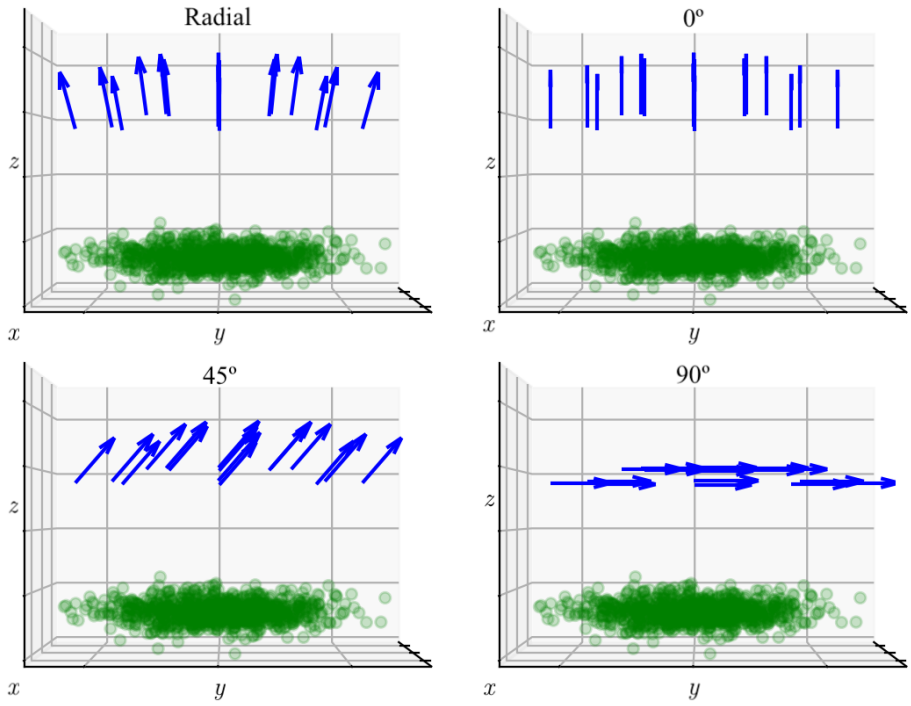

Figure 6.5: Several different bias field orientations. Radial sensors point outward. Other arrangements vary bias field in the azimuthal direction. Blue arrows indicate location of sensors and orientation, green dots mark realizations of randomly drawn dipoles used to generate data. All moment oriented randomly, but tangential to spherical surface. Gridline spacing is 1 cm in z-direction and 2 cm in x and y.

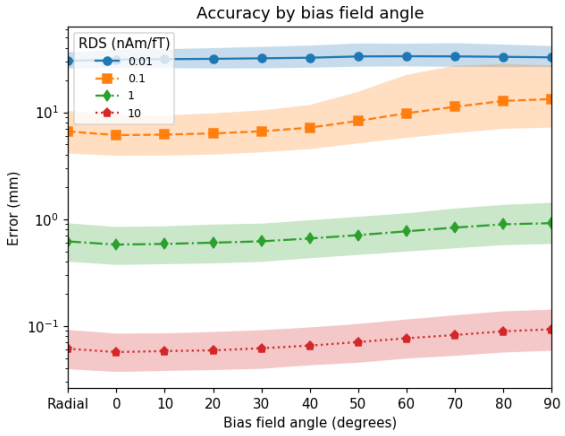

Figure 6.6: Median (curve) and middle 50th quantile (shaded region) localization error as a function of bias or ambient field for different RDS. Radial arrangement measures radial component of field at each sensor. Accuracy deteriorates slightly as bias field angle increases.

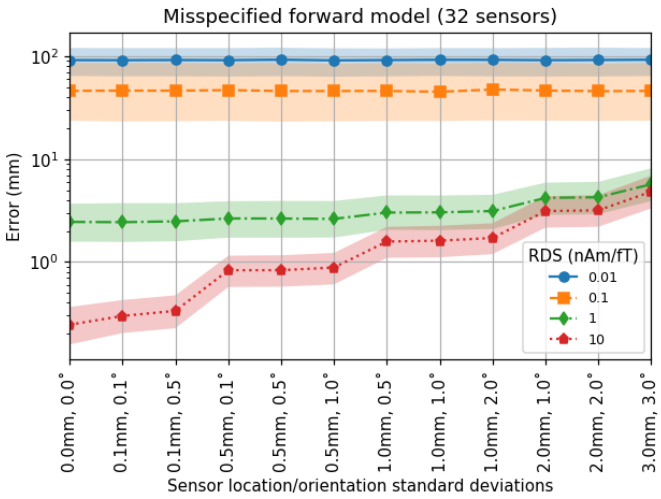

Figure 6.7: Median localization accuracy (curve) and middle 50th quantile error for a 32 gradiometers with varied perturbations to the sensor array used in forward model specification.

array. The standard deviations used varied from 0.1° to 3° for orientation and 0.1 mm to 3 mm for sensor location. Perturbations to orientation were common across all sensors; this noise model reflects uncertainty in the direction of a subject's head in a uniform bias field rather than variations of the bias field over space. Dipoles were drawn as in Section 6.3.1. Finally, we generated true data by using a forward model based on the perturbed sensors and the randomly drawn dipole. For each simulation, we calculated localization error using the presumptive forward model.

Localization error as a function of perturbation level for an array with 32 gradiometers is shown in Figure 6.7. These results are consistent across changes in array size. Perturbations impact the accuracy of sensitive arrays more. Small RDS values of 0.01 and 0.1 perform poorly even with correctly specified forward models, hence, there is little accuracy to lose as perturbation levels increase.

Correctly specifying sensor locations within 0.1 mm seems challenging for a wearable scalar OPM MEG system. Commercially available optical scanners can localize sensors to 0.5 mm, but the question of how well sensors remain in place during a recording persists, especially if the subject can move their head. Simulations show that scalar OPMs are fairly robust to perturbations in bias

field orientation, but more sensitive to the uncertainty in sensor location. Sensitivity to sensor uncertainty is impacted by sensor count and dipole strength as well. In the case of 32 sensors (Fig. 6.7), increased forward model accuracy has little impact on dipoles with low RDS. It is therefore imperative to increase sensitivity to a point before adding more sensors to improve localization accuracy.

6.4 Phantom experiment

6.4.1 Experimental setup and data collection

To validate our method, we constructed a dry phantom to mimic a current dipole in a conducting sphere [Ilmoniemi et al., 1985, Oyama et al., 2015, Uehara et al., 2008]. We constructed a virtual sensor array based on a single scalar gradiometer using a scalar magnetometer concept described in detail in Ref. [Gerginov et al., 2020]. The experiment was carried out inside a three layer magnetically shielded chamber. The gradiometer had a fixed base distance and was used to record many dipolar sources spread over a volume consecutively. The phantom setup is shown in Figure 6.8. The gradiometer was based on two pulsed OPMs featuring $3 \times 4.5 \times 5$ mm^3 internal volume vapor cells filled with ^{87}Rb and 600 Torr Nitrogen N_2 gas. The pulsed OPM concept is described in detail in [Gerginov et al., 2020]; it is based on optically-driven spin precession [Bell and Bloom, 1957] using amplitude-modulated pump light. A full description of the OPM is beyond the scope of the present work but we point out that the only substantial difference in the sensors used in this work are the larger, glass-blown vapor cell and the orthogonal pump and probe light geometry compared to the parallel geometry in [Gerginov et al., 2020]. Each magnetometer had a white-noise floor of 70 fT/$\sqrt{\text{Hz}}$ in a 500 Hz bandwidth. The distance between magnetometers (or baseline) of the gradiometer was 15 mm. The dipolar sources were approximated by currents flowing along the boundary of arcs with small subtended angle [Ilmoniemi et al., 1985] fabricated on a Electroless nickel immersion gold (ENIG) printed circuit board (PCB). The dipoles each had a length of 1 mm. They were placed on a circle of radius 70 mm and connected with thin twisted

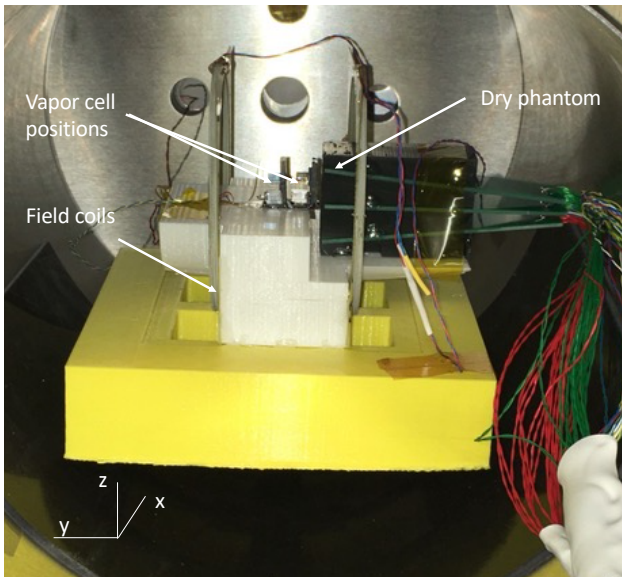

Figure 6.8: Phantom experiment setup.

wires at the center point of the circle. Neighboring dipoles on each PCB were separated by 1.42 mm center to center. They were activated independently with a current source. Three identical PCBs were stacked together to place the dipoles on a sphere with radius 100 mm, each PCB separated by 12.8 mm at the location of the dipoles. The phantom was placed such that the central dipole was 11.7 mm from the closer vapor cell of the scalar gradiometer. A 150 Hz sinusoidal signal of 25 mV across 1200 Ω was applied to each dipole to mimic neuronal activity. This equates to a current of 20.8 µA over 1 mm arc length and yielded a dipole strength of 20.8 nAm. The resultant field from each dipolar source was recorded for 100 seconds from which we created a virtual MEG sensor array. We picked a grid of 9 out of the 3 x 17 dipoles in our array. To mimic a realistic geometry, the nine dipoles were spaced by roughly 12.5 mm in x- and z-directions. Data from the 9 dipole or "virtual" sensors were used for localization.

6.4.2 Data processing

The virtual array exploits the fact that magnetic fields induced by a current dipole at a point in space depends on relative and not absolute position. As such, we can relocate each dipole

to a single point in space and vary sensor locations to mimic coverage over a subjects head. For example, with a sensor at point $\mathbf{s}$ and dipoles at locations $\mathbf{r}_1$ and $\mathbf{r}_2$, displacements are given by $\mathbf{d}_1 = \mathbf{s} - \mathbf{r}_1$ and $\mathbf{d}_2 = \mathbf{s} - \mathbf{r}_2$. If we treat $\mathbf{r}_1$ as the true dipole location then we can write $\mathbf{d}_2 = \mathbf{s} - \mathbf{r}_2 + (\mathbf{r}_1 - \mathbf{r}_1) = (\mathbf{s} + \mathbf{r}_1 - \mathbf{r}_2) - \mathbf{r}_1 = \tilde{\mathbf{s}} - \mathbf{r}_1$, where $\tilde{\mathbf{s}}$ is the location of the virtual sensor. Geometrically, the two are equivalent but in the latter, we treat the dipole as fixed rather than the sensor.

It should be noted that our simulations in Section 6.3 focused on arrays with sensors placed on a spherical surface with uniform orientation, i.e., all pointing along the bias field. For the phantom setup, treating the spatially varying dipole as fixed results in an equivalent array with sensors orientated radially (pointing directly away from source). Since bias field orientation has little impact on localization accuracy as shown in Section 6.3.3, our simulation results should serve as a good proxy for the phantom. To be certain, we compare our experimental results to simulations for an identical setup in Section 6.4.3.

Although the 150 Hz periodic signal was common to all dipoles, the phase differed for each virtual sensor since data collection started at different points in the cycle, i.e., not all recordings began when the signal had peak amplitude. To localize, the data from each virtual sensor had to be aligned in time (peak signal strengths must occur simultaneously). To accomplish this, we took the Fourier Transform of the time series data at each sensor to recover amplitude and phase. That is, for time series $x(t)$, we took $\hat{x}(\omega) = \mathcal{F}\{x(t)\}$. We found the phase, ϕ, of the 150 Hz signal then shifted *all* Fourier coefficients by $-\phi$, i.e., $\hat{x}_S(t) = \hat{x}(t)e^{-i\phi}$. Once equipped with phase-aligned data, we reconstructed our phase-shifted time series with $x_S(t) = \mathcal{F}^{-1}\{\hat{x}_S(\omega)\}$. Performing this same procedure for all sensors in the virtual array, we aligned our time series data for localization.

Since the true dipole strength and magnetometer sensitivity were fixed in the experiment, we varied filter bandwidth to change our figure of merit, relative dipole strength or RDS (see Eq. 6.13). We used three band-pass filters centered at 150 Hz with bandwidths of 1 Hz, 10 Hz, and 100 Hz, respectively, to process the phase-aligned time series data. Since the signal of interest lies at 150 Hz, decreasing filter bandwidth allows for rejection of noise at other frequencies thereby

increasing the SNR. Noise reduction can be accomplished through averaging as well but we limit our discussion to filter bandwidth here.

To process the data for localization, we considered our phase aligned times-series as a data matrix, $M \in \mathbb{R}^{9\times T}$, where T is the total number of time measurements. $M_{i,j}$ denotes the entry in the ith row and jth column of M. We split the 100 second time-series into roughly single period/cycle subsets. Since we sampled at 1,000 measurements per second, a single period of a 150 Hz signal contains approximately seven time measurements. Each row represents time-series data for a fixed sensor and each column is a measurement vector across sensors at a fixed time. Letting $M^{(t)} \in \mathbb{R}^{9\times 7}$ be the measurement sub-matrix for the t^{th} subset and defining

$$k^* = \underset{j\in\{1,\ldots,7\}}{\operatorname{argmax}} \left[\sum_{i=1}^{9} \left(M_{i,j}^{(t)}\right)^2\right], \tag{6.14}$$

we used the k^* column of $M^{(t)}$ for the t^{th} sensor reading. In principle, the max power reading corresponds to peaks or troughs in the signal where the SNR is maximized. We used these "maximum power" columns to localize with for each subset.

6.4.3 Localization of experimental data and comparison to simulations

After discarding the first and last 2 seconds of each data set to to avoid artifacts at the beginning and end of each recording, we were left with 9,600 sensor readings. For a dipole at point $(0,\ 7,\ 0)$ cm oriented in the x-direction with RDS of 2.98, 0.951, and 0.298, the median localization errors were 5.0 mm, 5.5 mm, and 11.5 mm for measured data. respectively. The left panel of Figure 6.9 shows elevation views of estimated dipole locations for the data measured using the phantom virtual array.

We compared our experimental results to simulations for an identical setup. We estimated that each dipole location was known to within 0.75 mm for two reasons. First, our dipoles are not true dipoles; they are wires with a spatial extent of 1 mm. We allow for 0.5 mm uncertainty due to this "smearing". Second, we were limited to 0.25 mm accuracy for spatial measurements. In future work, we would like to change our physical model to account for the spatial extent of the

"dipole" but, as a first-step, account for it now through uncertainty. We estimated that the bias field is known within 3° ; the variability here stems from slight imperfections in the construction of the phantom more so than ambiguity in the direction of the bias field. Fixing the simulated

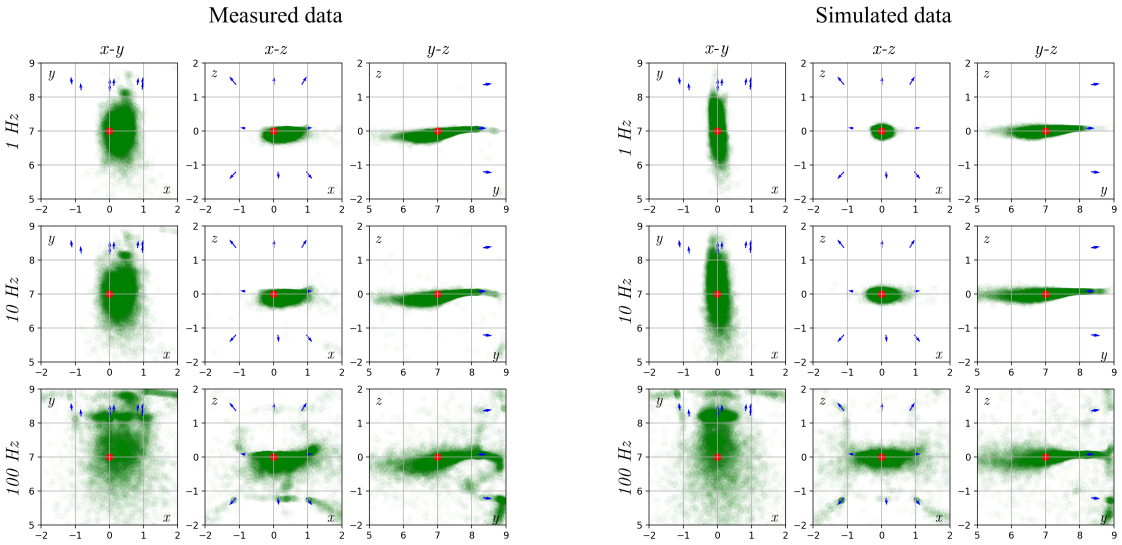

Figure 6.9: Elevation views of localization results for phantom array. Heavy red dots indicate true dipole location, light green dots show dipole location estimates for 10,000 field measurements. Blue arrows show sensor locations and orientations. Scale in centimeters. Left: localization results for data measured using phantom virtual array. Right: localization results for simulated data under identical conditions. Gridline spacing is 1 cm

dipole at the same point in space, we generated 10,000 sensor readings using the method detailed in Section 6.3.4. With the simulated sensor readings, we estimated the dipole's location and compared it to the known coordinates. The median localization errors were 3.3 mm, 4.6 mm, and 11.8 mm for RDS of 2.98, 0.951, and 0.298, respectively. The right panel of Figure 6.9 shows elevation views of estimated dipole locations for simulated data. We note that both simulated and measured data exhibit the same qualitative behavior. Although the localization errors for each respective RDS differ, their orders of magnitude all agree and are well within a standard deviation of each other. We note that the mean estimated location for measured data shows a bias; this indicates a misspecified forward model as expected. Since the model errors are common to all localization runs, this behavior is expected and of no concern.

6.5 Conclusion

In this study, we considered single-source dipole localization with scalar OPM arrays in large uniform ambient fields. We investigated how localization depends on dipole strength/sensor noise, sensor count, bias field orientation, and uncertainty in forward models. We provided results of numerical simulation as general guidelines for future work on scalar OPM MEG arrays in ambient magnetic fields. Numerical results were validated experimentally by localizing a dipolar source on a phantom using a virtual scalar gradiometer array.

Given current sensitivities of our scalar OPMs (around 70 fT/$\sqrt{\text{Hz}}$), localization of single neuronal dipoles at the 10 nAm level with a 100 Hz bandwidth under optimal conditions is unlikely unless the signal is averaged at least 100 times based on our simulations. With this level of averaging and an array of 128 sensors, localization accuracy around 1 cm is predicted. Increasing sensitivity by a factor of 10 to 7 fT/$\sqrt{\text{Hz}}$ would allow localization accuracy of 1 mm with a 100 sensor array and 100 averages, provided that the positions of the sensors can be determined with an accuracy of 0.5 mm. While this does not sound impossible, it is surely a challenging task. In these simulations, we assumed an ideal case, where the background fields were uniform across the sensor array, i.e., no close noise sources, and that the common-mode rejection of the gradiometer is sufficiently high to cancel all ambient noise and the localization is limited purely by sensor noise. Future work involves improved gradiometry to account for spatial variations in the ambient field and the localization of multiple dipoles. While there are still many challenges to overcome, the prospect of unshielded brain imaging with scalar magnetometers is exciting and would open many applications.

Bibliography

[Abdelfattah et al., 2021] Abdelfattah, A., Anzt, H., Boman, E. G., Carson, E., Cojean, T., Dongarra, J., Fox, A., Gates, M., Higham, N. J., Li, X. S., et al. (2021). A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. The International Journal of High Performance Computing Applications, page 10943420211003313.

[Afach et al., 2015] Afach, S., Ban, G., Bison, G., Bodek, K., Chowdhuri, Z., Grujić, Z., Hayen, L., Hélaine, V., Kasprzak, M., Kirch, K., et al. (2015). Highly stable atomic vector magnetometer based on free spin precession. Optics express, 23(17):22108–22115.

[Ailon and Chazelle, 2009] Ailon, N. and Chazelle, B. (2009). The fast johnson–lindenstrauss transform and approximate nearest neighbors. SIAM Journal on computing, 39(1):302–322.

[Allred et al., 2002] Allred, J., Lyman, R., Kornack, T., and Romalis, M. V. (2002). High-sensitivity atomic magnetometer unaffected by spin-exchange relaxation. Physical review letters, 89(13):130801.

[Alvarez Loya and Appelö, 2022] Alvarez Loya, A. and Appelö, D. (2022). A hermite method with a discontinuity sensor for hamilton–jacobi equations. Journal of Scientific Computing, 90(3):1–31.

[Antun et al., 2020] Antun, V., Renna, F., Poon, C., Adcock, B., and Hansen, A. C. (2020). On instabilities of deep learning in image reconstruction and the potential costs of ai. Proceedings of the National Academy of Sciences, 117(48):30088–30095.

[Anzt et al., 2011] Anzt, H., Heuveline, V., Rocker, B., Castillo, M., Fern´ndez, J. C., Mayo, R., and Quintana-Orti, E. S. (2011). Power consumption of mixed precision in the iterative solution of sparse linear systems. In 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pages 829–836.

[Baboulin et al., 2009] Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., and Tomov, S. (2009). Accelerating scientific computations with mixed precision algorithms. Computer Physics Communications, 180(12):2526–2533.

[Baillet, 2017] Baillet, S. (2017). Magnetoencephalography for brain electrophysiology and imaging. Nature neuroscience, 20(3):327–339.

[Barndorff-Nielsen and Cox, 1979] Barndorff-Nielsen, O. and Cox, D. R. (1979). Edgeworth and saddle-point approximations with statistical applications. Journal of the Royal Statistical Society: Series B (Methodological), 41(3):279–299.

[Battauz, 2011] Battauz, M. (2011). Laplace approximation in measurement error models. Biometrical journal, 53(3):411–425.

[Beck, 2017] Beck, A. (2017). First-Order Methods in Optimization. MOS-SIAM Series on Optimization.

[Becker and Clancy, 2020] Becker, S. and Clancy, R. J. (2020). Robust least squares for quantized data matrices. Signal Processing, 176:107711.

[Bell and Bloom, 1957] Bell, W. E. and Bloom, A. L. (1957). Optical detection of magnetic resonance in alkali metal vapor. Physical Review, 107(6):1559.

[Berahas et al., 2019] Berahas, A. S., Jahani, M., and Takác, M. (2019). Quasi-Newton methods for deep learning: Forget the past, just sample. arXiv preprint arXiv:1901.09997, 16.

[Bertsekas, 1971] Bertsekas, D. P. (1971). Control of Uncertain Systems with a Set-Membership Description of Uncertainty. PhD thesis, MIT, Cambridge, MA.

[Bertsekas, 1999] Bertsekas, D. P. (1999). Nonlinear Programming. Athena Scientific.

[Bertsimas et al., 2011] Bertsimas, D., Brown, D. B., and Caramanis, C. (2011). Theory and applications of robust optimization. SIAM Review, 53(3):464–501.

[Bischof et al., 2002] Bischof, C. H., Bücker, H. M., Lang, B., Rasch, A., and Vehreschild, A. (2002). Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2002), pages 65–72, Los Alamitos, CA, USA. IEEE Computer Society.

[Björkman and Holmström, 2000] Björkman, M. and Holmström, K. (2000). Global optimization of costly nonconvex functions using radial basis functions. Optimization and Engineering, 1(4):373–397.

[Boto et al., 2018] Boto, E., Holmes, N., Leggett, J., Roberts, G., Shah, V., Meyer, S. S., Muñoz, L. D., Mullinger, K. J., Tierney, T. M., Bestmann, S., et al. (2018). Moving magnetoencephalography towards real-world applications with a wearable system. Nature, 555(7698):657–661.

[Boyd et al., 2004] Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). Convex optimization. Cambridge university press.

[Boyd et al., 2003] Boyd, S., Xiao, L., and Mutapcic, A. (2003). Subgradient methods. lecture notes of EE392o, Stanford University, Autumn Quarter, 2004:2004–2005.

[Bradbury et al., 2018] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs, http://github.com/google/jax.

[Budker and Romalis, 2007] Budker, D. and Romalis, M. (2007). Optical magnetometry. Nature physics, 3(4):227–234.

[Byrd et al., 1994] Byrd, R. H., Nocedal, J., and Schnabel, R. B. (1994). Representations of quasi-Newton matrices and their use in limited memory methods. Mathematical Programming, 63(1):129–156.

[Cardot and Degras, 2018] Cardot, H. and Degras, D. (2018). Online principal component analysis in high dimension: Which algorithm to choose? International Statistical Review, 86(1):29–50.

[Carmichael et al., 2019] Carmichael, Z., Langroudi, H. F., Khazanov, C., Lillie, J., Gustafson, J. L., and Kudithipudi, D. (2019). Performance-efficiency trade-off of low-precision numerical formats in deep neural networks. In Proceedings of the Conference for Next Generation Arithmetic 2019, pages 1–9.

[Chen et al., 2018] Chen, R., Menickelly, M., and Scheinberg, K. (2018). Stochastic optimization using a trust-region method and random models. Mathematical Programming, 169(2):447–487.

[Chiang et al., 2017a] Chiang, W.-F., Baranowski, M., Briggs, I., Solovyev, A., Gopalakrishnan, G., and Rakamarić, Z. (2017a). Rigorous floating-point mixed-precision tuning. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, page 300–315, New York, NY, USA. Association for Computing Machinery.

[Chiang et al., 2017b] Chiang, W.-F., Baranowski, M., Briggs, I., Solovyev, A., Gopalakrishnan, G., and Rakamarić, Z. (2017b). Rigorous floating-point mixed-precision tuning. SIGPLAN Not., 52(1):300–315.

[Cohen, 1972] Cohen, D. (1972). Magnetoencephalography: detection of the brain's electrical activity with a superconducting magnetometer. Science, 175(4022):664–666.

[Cohen and Hosaka, 1976] Cohen, D. and Hosaka, H. (1976). Part ii magnetic field produced by a current dipole. Journal of electrocardiology, 9(4):409–417.

[Conn et al., 2013] Conn, A. R., Gould, G., and Toint, P. L. (2013). LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A), volume 17. Springer Science & Business Media.

[Conn et al., 2000] Conn, A. R., Gould, N. I., and Toint, P. L. (2000). Trust region methods. SIAM.

[Conn et al., 2008a] Conn, A. R., Scheinberg, K., and Vicente, L. N. (2008a). Geometry of interpolation sets in derivative free optimization. Mathematical programming, 111(1):141–172.

[Conn et al., 2008b] Conn, A. R., Scheinberg, K., and Vicente, L. N. (2008b). Geometry of sample sets in derivative-free optimization: polynomial regression and underdetermined interpolation. IMA journal of numerical analysis, 28(4):721–748.

[Conn et al., 2009] Conn, A. R., Scheinberg, K., and Vicente, L. N. (2009). Introduction to derivative-free optimization. SIAM.

[Cruz, 2017] Cruz, J. Y. B. (2017). On proximal subgradient splitting method for minimizing the sum of two nonsmooth convex functions. Set-Valued and Variational Analysis, 25(2):245–263.

[Dahl and Vandenberghe, 2010] Dahl, J. and Vandenberghe, L. (2010). CVXOPT, Python Software for Convex Optimization, version 1.13. UCLA. Available at http://abel.ee.ucla.edu/cvxopt.

[Dang et al., 2010] Dang, H., Maloof, A. C., and Romalis, M. V. (2010). Ultrahigh sensitivity magnetic field and magnetization measurements with an atomic magnetometer. Applied Physics Letters, 97(15):151110.

[Daniels, 1954] Daniels, H. E. (1954). Saddlepoint approximations in statistics. The annals of mathematical statistics, pages 631–650.

[David et al., 2004] David, A., Cole, M., Horsley, T., Linford, N., Linford, P., and Martin, L. (2004). A rival to stonehenge? geophysical survey at stanton drew, england. Antiquity, 78(300):341–358.

[Debye, 1909] Debye, P. (1909). Näherungsformeln für die zylinderfunktionen für große werte des arguments und unbeschränkt veränderliche werte des index. Mathematische Annalen, 67(4):535–558.

[Dolan and Moré, 2002] Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. Mathematical Programming, 91:201–213.

[Domahidi et al., 2013] Domahidi, A., Chu, E., and Boyd, S. (2013). ECOS: An SOCP solver for embedded systems. In 2013 European Control Conference (ECC), pages 3071–3076. IEEE.

[Doucet et al., 2019] Doucet, N., Ltaief, H., Gratadour, D., and Keyes, D. (2019). Mixed-precision tomographic reconstructor computations on hardware accelerators. In 2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3), pages 31–38. IEEE.

[Du et al., 2017] Du, C., Xia, M., Huang, S., Xu, Z., Peng, X., and Guo, H. (2017). Detection of a moving magnetic dipole target using multiple scalar magnetometers. IEEE Geoscience and Remote Sensing Letters, 14(7):1166–1170.

[El Ghaoui and Lebret, 1997] El Ghaoui, L. and Lebret, H. (1997). Robust solutions to least-squares problems with uncertain data. SIAM J. Matrix Anal. Appl., 18(4):1035 – 1064.

[Eykholt et al., 2018] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1625–1634.

[Fagan et al., 2016] Fagan, M., Schlachter, J., Yoshii, K., Leyffer, S., Palem, K., Snir, M., Wild, S. M., and Enz, C. (2016). Overcoming the power wall by exploiting inexactness and emerging COTS architectural features: Trading precision for improving application quality. In 2016 29th IEEE International System-on-Chip Conference (SOCC), pages 241–246.

[Fasano et al., 2009] Fasano, G., Morales, J. L., and Nocedal, J. (2009). On the geometry phase in model-based algorithms for derivative-free optimization. Optimization Methods & Software, 24(1):145–154.

[Fowkes and Roberts, 2018] Fowkes, J. and Roberts, L. (2018). PyCUTEst, https://jfowkes.github.io/pycutest.

[Fuller, 1987] Fuller, W. A. (1987). Measurement error models.

[Galal and Horowitz, 2011] Galal, S. and Horowitz, M. (2011). Energy-efficient floating-point unit design. IEEE Transactions on Computers, 60(7):913–922.

[Gerginov et al., 2017] Gerginov, V., Krzyzewski, S., and Knappe, S. (2017). Pulsed operation of a miniature scalar optically pumped magnetometer. JOSA B, 34(7):1429–1434.

[Gerginov et al., 2020] Gerginov, V., Pomponio, M., and Knappe, S. (2020). Scalar magnetometry below 100 ft/hz 1/2 in a microfabricated cell. IEEE Sensors Journal, 20(21):12684–12690.

[Göddeke et al., 2007] Göddeke, D., Strzodka, R., and Turek, S. (2007). Performance and accuracy of hardware-oriented native-, emulated-and mixed-precision solvers in FEM simulations. International Journal of Parallel, Emergent and Distributed Systems, 22(4):221–256.

[Goldberg, 1991] Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. ACM computing surveys (CSUR), 23(1):5–48.

[Goldfarb and Iyengar, 2003] Goldfarb, D. and Iyengar, G. (2003). Robust portfolio selection problems. Mathematics of operations research, 28(1):1–38.

[Golub et al., 1999] Golub, G. H., Hansen, P. C., and O'Leary, D. P. (1999). Tikhonov regularization and total least squares. SIAM journal on matrix analysis and applications, 21(1):185–194.

[Golub and Van Loan, 1996] Golub, G. H. and Van Loan, C. F. (1996). Matrix computations. Johns Hopkins University Press, Baltimore, MD,.

[González, 2010] González, Á. (2010). Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. Mathematical Geosciences, 42(1):49.

[Gorissen et al., 2015] Gorissen, B. L., Yanıkoğlu, İ., and den Hertog, D. (2015). A practical guide to robust optimization. Omega, 53:124 – 137.

[Gould et al., 2015] Gould, N. I., Orban, D., and Toint, P. L. (2015). CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Computational Optimization and Applications, 60(3):545–557.

[Goutis and Casella, 1999] Goutis, C. and Casella, G. (1999). Explaining the saddlepoint approximation. The American Statistician, 53(3):216–224.

[Graillat et al., 2019] Graillat, S., Jézéquel, F., Picot, R., Févotte, F., and Lathuilière, B. (2019). Auto-tuning for floating-point precision with discrete stochastic arithmetic. Journal of computational science, 36:101017.

[Gratton and Toint, 2018] Gratton, S. and Toint, P. L. (2018). A note on solving nonlinear optimization problems in variable precision. arXiv preprint arXiv:1812.03467.

[Gratton and Toint, 2020] Gratton, S. and Toint, P. L. (2020). A note on solving nonlinear optimization problems in variable precision. Computational Optimization and Applications, 76(3):917–933.

[Guo and Rubio-González, 2018] Guo, H. and Rubio-González, C. (2018). Exploiting community structure for floating-point precision tuning. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 333–343.

[Guo and Lewis, 2018] Guo, J. and Lewis, A. (2018). Nonsmooth variants of powell's BFGS convergence theorem. SIAM J. Optimization, 28(2):1301–1311.

[H. Golub and F. van Loan, 1980] H. Golub, G. and F. van Loan, C. (1980). An analysis of the total least squares problem. SIAM J. Numer. Anal., 17(6):883 – 893.

[Hall, 2013] Hall, P. (2013). The bootstrap and Edgeworth expansion. Springer Science & Business Media.

[Hämäläinen et al., 1993] Hämäläinen, M., Hari, R., Ilmoniemi, R. J., Knuutila, J., and Lounasmaa, O. V. (1993). Magnetoencephalography—theory, instrumentation, and applications to noninvasive studies of the working human brain. Reviews of modern Physics, 65(2):413.

[Hao, 2019] Hao, K. (2019). Training a single AI model can emit as much carbon as five cars in their lifetimes. MIT Technology Review.

[Happer and Tang, 1977] Happer, W. and Tang, H. Y. (1977). Method and apparatus for stimulating narrow line resonance conditions. US Patent 4,005,355.

[Heinkenschloss and Vicente, 2002] Heinkenschloss, M. and Vicente, L. (2002). Analysis of inexact trust-region sqp algorithms. SIAM Journal on Optimization, 12:283–302.

[Hodge and Moore, 1972] Hodge, S. and Moore, P. (1972). Data uncertainties and least squares regression. Journal of the Royal Statistical Society. Series C (Applied Statistics),, 21:185 – 195.

[Hückelheim, 2019] Hückelheim, J. (2019). PREDUCER, https://github.com/jhueckelheim/preducer.

[Huzurbazar, 1999] Huzurbazar, S. (1999). Practical saddlepoint approximations. The American Statistician, 53(3):225–232.

[Hyvärinen and Oja, 2000] Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. Neural networks, 13(4-5):411–430.

[Ichimura et al., 2018] Ichimura, T., Fujita, K., Yamaguchi, T., Naruse, A., Wells, J. C., Schulthess, T. C., Straatsma, T. P., Zimmer, C. J., Martinasso, M., Nakajima, K., et al. (2018). A fast scalable implicit solver for nonlinear time-evolution earthquake city problem on low-ordered unstructured finite elements with artificial intelligence and transprecision computing. In SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 627–637. IEEE.

[Ilmoniemi et al., 1985] Ilmoniemi, R., Hämäläinen, M., and Knuutila, J. (1985). The forward and inverse problems in the spherical model biomagnetism: Applications and theory ed h weinberg, g stroink and t katila.

[Ilmoniemi and Sarvas, 2019] Ilmoniemi, R. J. and Sarvas, J. (2019). Brain signals: Physics and mathematics of MEG and EEG. Mit Press.

[Jackson et al., 2020] Jackson, R., Collis, S., Potvin, C., and Munson, T. (2020). PyDDA: A Pythonic direct data assimilation framework for wind retrievals. Journal of Open Research Software, 8(1).

[Jia et al., 2018] Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., et al. (2018). Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. arXiv preprint arXiv:1807.11205.

[Johnson and Lindenstrauss, 1984] Johnson, W. B. and Lindenstrauss, J. (1984). Extensions of lipschitz mappings into a hilbert space 26. Contemporary mathematics, 26:28.

[Kamgar-Parsi et al., 1995] Kamgar-Parsi, B., Kamgar-Parsi, B., and Brosh, M. (1995). Distribution and moments of the weighted sum of uniforms random variables, with applications in reducing monte carlo simulations. Journal of Statistical Computation and Simulation, 52(4):399–414.

[Kastler, 1950] Kastler, A. (1950). Quelques suggestions concernant la production optique et la détection optique d'une inégalité de population des niveaux de quantifigation spatiale des atomes. application à l'expérience de stern et gerlach et à la résonance magnétique. J. phys. radium, 11(6):255–265.

[Kestor et al., 2013] Kestor, G., Gioiosa, R., Kerbyson, D. J., and Hoisie, A. (2013). Quantifying the energy cost of data movement in scientific applications. In 2013 IEEE international symposium on workload characterization (IISWC), pages 56–65. IEEE.

[Ko and Davidian, 2000] Ko, H. and Davidian, M. (2000). Correcting for measurement error in individual-level covariates in nonlinear mixed effects models. Biometrics, 56(2):368–375.

[Kominis et al., 2003] Kominis, I., Kornack, T., Allred, J., and Romalis, M. V. (2003). A subfemtotesla multichannel atomic magnetometer. Nature, 422(6932):596–599.

[Kouri et al., 2014] Kouri, D. P., Heinkenschloss, M., Ridzal, D., and van Bloemen Waanders, B. G. (2014). Inexact objective function evaluations in a trust-region algorithm for PDE-constrained optimization under uncertainty. SIAM J. Scientific Computing, 36.

[Larson et al., 2019] Larson, J., Menickelly, M., and Wild, S. M. (2019). Derivative-free optimization methods. Acta Numerica, 28:287–404.

[Lemarechal, 1982] Lemarechal, C. (1982). Numerical experiments in nonsmooth optimization. In Progress in Nondifferentiable Optimization, pages 61–84.

[Lewis and Overton, 2013] Lewis, A. and Overton, M. (2013). Nonsmooth optimization via quasi-newton methods. Mathematical Programming, 141(2):135–163.

[Limes et al., 2020a] Limes, M., Foley, E., Kornack, T., Caliga, S., McBride, S., Braun, A., Lee, W., Lucivero, V., and Romalis, M. (2020a). Portable magnetometry for detection of biomagnetism in ambient environments. Physical Review Applied, 14(1):011002.

[Limes et al., 2020b] Limes, M., Foley, E., Kornack, T., Caliga, S., McBride, S., Braun, A., Lee, W., Lucivero, V., and Romalis, M. (2020b). Total-field atomic gradiometer for unshielded portable magnetoencephalography. arXiv preprint arXiv:2001.03534.

[Limes et al., 2020c] Limes, M., Foley, J., Kornack, T., Caliga, S., McBride, S., Braun, A., Lee, W., Lucivero, V.-G., and Romalis, M. (2020c). A sensitive all-optical portable scalar ^{87} rb atomic gradiometer operating in earth's field. Bulletin of the American Physical Society.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. Mathematical programming, 45(1-3):503–528.

[Livanov et al., 1978] Livanov, M., Kozlov, A., Korinevskiĭ, A., Markin, V., and Sinel'nikova, S. (1978). Recording of human magnetic fields. Doklady Akademii Nauk SSSR, 238(1):253.

[Lugannani and Rice, 1980] Lugannani, R. and Rice, S. (1980). Saddle point approximation for the distribution of the sum of independent random variables. Advances in applied probability, 12(2):475–490.

[Mäkelä, 2002] Mäkelä, M. (2002). Survey of bundle methods for nonsmooth optimization. Optimization methods and software, 17(1):1 – 29.

[Markovsky and Van Huffel, 2007] Markovsky, I. and Van Huffel, S. (2007). Overview of total least-squares methods. Signal Processing, 87:2283 – 2302.

[Martinsson and Tropp, 2020] Martinsson, P.-G. and Tropp, J. A. (2020). Randomized numerical linear algebra: Foundations and algorithms. Acta Numerica, 29:403–572.

[Mattingley and Boyd, 2012] Mattingley, J. and Boyd, S. (2012). CVXGEN: A code generator for embedded convex optimization. Optimization and Engineering, 13(1):1–27.

[Menon et al., 2018] Menon, H., Lam, M. O., Osei-Kuffuor, D., Schordan, M., Lloyd, S., Mohror, K., and Hittinger, J. (2018). Adapt: Algorithmic differentiation applied to floating-point precision tuning. In SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 614–626. IEEE.

[Micikevicius et al., 2018] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. (2018). Mixed precision training. In International Conference on Learning Representations.

[Millán and Machado, 2019] Millán, R. D. and Machado, M. P. (2019). Inexact proximal ϵ-subgradient methods for composite convex optimization problems. Journal of Global Optimization, 75(4):1029–1060.

[Molka et al., 2010] Molka, D., Hackenberg, D., Schöne, R., and Müller, M. S. (2010). Characterizing the energy consumption of data transfers and arithmetic operations on x86- 64 processors. In International conference on green computing, pages 123–133. IEEE.

[Moré and Wild, 2009] Moré, J. J. and Wild, S. M. (2009). Benchmarking derivative-free optimization algorithms. SIAM Journal on Optimization, 20(1):172–191.

[Mosher et al., 1999] Mosher, J. C., Leahy, R. M., and Lewis, P. S. (1999). Eeg and meg: forward solutions for inverse methods. IEEE Transactions on Biomedical Engineering, 46(3):245–259.

[Nabighian et al., 2005] Nabighian, M. N., Grauch, V., Hansen, R., LaFehr, T., Li, Y., Peirce, J., Phillips, J., and Ruder, M. (2005). The historical development of the magnetic method in exploration. Geophysics, 70(6):33ND–61ND.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. The computer journal, 7(4):308–313.

[Nocedal, 1980] Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage. Math. Comp., 25(151):773–782.

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. (2006). Numerical optimization. Springer Science & Business Media.

[Ong et al., 2008] Ong, Y.-S., Lum, K. Y., and Nair, P. B. (2008). Hybrid evolutionary algorithm with hermite radial basis function interpolants for computationally expensive adjoint solvers. Computational Optimization and Applications, 39(1):97–119.

[Oyama et al., 2015] Oyama, D., Adachi, Y., Yumoto, M., Hashimoto, I., and Uehara, G. (2015). Dry phantom for magnetoencephalography—configuration, calibration, and contribution. Journal of Neuroscience Methods, 251:24–36.

[O'brien et al., 2017] O'brien, K., Pietri, I., Reddy, R., Lastovetsky, A., and Sakellariou, R. (2017). A survey of power and energy predictive models in HPC systems and applications. ACM Computing Surveys (CSUR), 50(3):1–38.

[Parzen, 1962] Parzen, E. (1962). On estimation of a probability density function and mode. The annals of mathematical statistics, 33(3):1065–1076.

[Perry et al., 2020] Perry, A., Bulatowicz, M., Larsen, M., Walker, T., and Wyllie, R. (2020). All-optical intrinsic atomic gradiometer with sub-20 ft/cm/$\sqrt{\text{hz}}$ sensitivity in a 22 μt earth-scale magnetic field. Optics Express, 28(24):36696–36705.

[Pintelon and Schoukens, 2012] Pintelon, R. and Schoukens, J. (2012). System identification: a frequency domain approach. John Wiley & Sons.

[Potvin et al., 2012] Potvin, C. K., Shapiro, A., and Xue, M. (2012). Impact of a vertical vorticity constraint in variational dual-doppler wind analysis: Tests with real and simulated supercell data. Journal of Atmospheric and Oceanic Technology, 29(1):32–49.

[Powell, 2003] Powell, M. J. (2003). On trust region methods for unconstrained minimization without derivatives. Mathematical programming, 97(3):605–623.

[Reid, 1988] Reid, N. (1988). Saddlepoint methods and statistical inference. Statistical Science, pages 213–227.

[Revels et al., 2016] Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-mode automatic differentiation in Julia. arXiv:1607.07892 [cs.MS].

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. The Annals of Mathematical Statistics, 22(3):400–407.

[Rosenblatt, 1956] Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. The annals of mathematical statistics, 27(3):832–837.

[Rubio-González et al., 2013] Rubio-González, C., Nguyen, C., Nguyen, H. D., Demmel, J., Kahan, W., Sen, K., Bailey, D. H., Iancu, C., and Hough, D. (2013). Precimonious: Tuning assistant for floating-point precision. In SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pages 1–12.

[Sarvas, 1987] Sarvas, J. (1987). Basic mathematical and electromagnetic concepts of the biomagnetic inverse problem. Physics in Medicine & Biology, 32(1):11.

[Scheinberg and Toint, 2010] Scheinberg, K. and Toint, P. L. (2010). Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. SIAM Journal on Optimization, 20(6):3512–3532.

[Schmidt, 2005] Schmidt, M. (2005). minFunc: unconstrained differentiable multivariate optimization in Matlab. https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html.

[Schmidt, 2012] Schmidt, M. (2012). minFunc software package. http://www.di.ens.fr/~mschmidt/Software/minFunc.html.

[Schwindt et al., 2004] Schwindt, P. D., Knappe, S., Shah, V., Hollberg, L., Kitching, J., Liew, L.-A., and Moreland, J. (2004). Chip-scale atomic magnetometer. Applied Physics Letters, 85(26):6409–6411.

[Shapiro et al., 2009] Shapiro, A., Potvin, C. K., and Gao, J. (2009). Use of a vertical vorticity equation in variational dual-Doppler wind analysis. Journal of Atmospheric and Oceanic Technology, 26(10):2089–2106.

[Sheng et al., 2013] Sheng, D., Li, S., Dural, N., and Romalis, M. V. (2013). Subfemtotesla scalar atomic magnetometry using multipass cells. Physical review letters, 110(16):160802.

[Shivaswamy et al., 2006] Shivaswamy, P. K., Bhattacharyya, C., and Smola, A. J. (2006). Second order cone programming approaches for handling missing and uncertain data. J. Mach. Learn. Res., 7:1283–1314.

[Spady, 1991] Spady, R. H. (1991). Saddlepoint approximations for regression models. Biometrika, 78(4):879–889.

[Spall, 2005] Spall, J. C. (2005). Introduction to stochastic search and optimization: estimation, simulation, and control, volume 65. John Wiley & Sons.

[Strang, 2007] Strang, G. (2007). Computational methods for inverse problems, volume 23. Wellesley-Cambridge Press.

[Strawderman et al., 1996] Strawderman, R. L., Casella, G., and Wells, M. T. (1996). Practical small-sample asymptotics for regression problems. Journal of the American Statistical Association, 91(434):643–654.

[Strzodka and Göddeke, 2006] Strzodka, R. and Göddeke, D. (2006). Mixed precision methods for convergent iterative schemes. EDGE, 6:23–24.

[Supek and Aine, 2016] Supek, S. and Aine, C. J. (2016). Magnetoencephalography. Springer.

[Uehara et al., 2008] Uehara, G., Adachi, Y., Hashimoto, I., and Yumoto, M. (2008). A triangle-coil phantom as an evaluation standard for meg system. The Journal of Japan Biomagnetism and Bioelectromagnetics Society, 21:11–17.

[Van Huffel and Vandewalle, 1991] Van Huffel, S. and Vandewalle, J. (1991). The total least squares problem: computational aspects and analysis. SIAM.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods, 17(3):261–272.

[Vogel, 2002] Vogel, C. R. (2002). Computational methods for inverse problems, volume 23. Siam.

[Vogel, 1979] Vogel, H. (1979). A better way to construct the sunflower head. Mathematical biosciences, 44(3-4):179–189.

[Vonesh, 1996] Vonesh, E. F. (1996). A note on the use of laplace's approximation for nonlinear mixed-effects models. Biometrika, 83(2):447–452.

[Wang et al., 2018] Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. (2018). Training deep neural networks with 8-bit floating point numbers. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, pages 7686–7695.

[Watson, 1995] Watson, G. N. (1995). A treatise on the theory of Bessel functions. Cambridge university press.

[Weinstein and Rao, 2017] Weinstein, M. J. and Rao, A. V. (2017). Algorithm 984: Adigator, a toolbox for the algorithmic differentiation of mathematical functions in matlab using source transformation via operator overloading. ACM Transactions on Mathematical Software (TOMS), 44(2):21.

[Wiesel et al., 2008] Wiesel, A., Eldar, Y. C., and Yeredor, A. (2008). Linear regression with gaussian model uncertainty: Algorithms and bounds. IEEE Transactions on Signal Processing, 56(6):2194–2205.

[Wild et al., 2008] Wild, S. M., Regis, R. G., and Shoemaker, C. A. (2008). Orbit: Optimization by radial basis function interpolation in trust-regions. SIAM Journal on Scientific Computing, 30(6):3197–3219.

[Wilkinson, 1971] Wilkinson, J. H. (1971). Modern error analysis. SIAM review, 13(4):548–568.

[Xu et al., 2010] Xu, H., Caramanis, C., and Mannor, S. (2010). Robust regression and lasso. IEEE Tran. Info. Theory, 56(7):3561–3574.

[Zhang et al., 2020a] Zhang, R., Mhaskar, R., Smith, K., and Prouty, M. (2020a). Portable intrinsic gradiometer for ultra-sensitive detection of magnetic gradient in unshielded environment. Applied Physics Letters, 116(14):143501.

[Zhang et al., 2020b] Zhang, R., Xiao, W., Ding, Y., Feng, Y., Peng, X., Shen, L., Sun, C., Wu, T., Wu, Y., Yang, Y., et al. (2020b). Recording brain activities in unshielded earth's field with optically pumped atomic magnetometers. Science Advances, 6(24):eaba8792.

[Zhu et al., 2014] Zhu, J., Wang, X., Lin, X., and Gu, Y. (2014). Maximum likelihood estimation from sign measurements with sensing matrix perturbation. IEEE transactions on signal processing, 62(15):3741–3753.

[Zhu et al., 2020] Zhu, J., Zhang, Q., Meng, X., and Xu, Z. (2020). Vector approximate message passing algorithm for compressed sensing with structured matrix perturbation. Signal Processing, 166:107248.

Appendix A

Gradient Factors

The gradient for our approximate likelihood function is outlined in Equations 3.16-3.19 when $\mathbf{t}$ cannot be solved for explicitly. To calculate $\nabla_{\mathbf{x}}\ell(\mathbf{x})$, we need expressions for the joint CGF and a number of its derivatives. In particular, we require $K^{(i)}_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t})$ for $i = 0, 1, 2, 3$ and $\frac{\partial}{\partial \mathbf{x}} K^{(j)}_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t})$ for $j = 0, 1, 2$. We present the necessary derivatives for gradient determination for the examples listed in Section 3.5. Recall that $\mathbf{t}$ is the solution to the equation $K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \mathbf{y}$ for a given $\mathbf{x}$.

A.1 Floating point/rounding error

In what follows, let $\mathbf{D}$ be the floating point error matrix and $\mathbf{M} = \mathbf{D} \odot (\mathbf{t}\mathbf{x}^T)$. For fixed point or rounding error with uncertainty parameter δ, the matrix reduces to $\mathbf{D} = \delta \mathbf{1}_m \mathbf{1}_n^T$ such that $\mathbf{M} = \delta \mathbf{t}\mathbf{x}^T$

$$K_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \frac{\sigma^2 \mathbf{t}^2}{2} + \mathbf{t} \odot \mathbf{H}\mathbf{x} + \ln\left(\sinh\left(\mathbf{M}\right) \oslash \mathbf{M}\right)\mathbf{1} \tag{A.1}$$

$$K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \sigma^2 \mathbf{t} + \mathbf{H}\mathbf{x} + \left[\mathbf{D} \odot \coth\left(\mathbf{M}\right)\right]\mathbf{x} - n(\mathbf{1} \oslash \mathbf{t}) \tag{A.2}$$

$$K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \sigma^2 \mathbf{1} - \left[\mathbf{D}^2 \odot \operatorname{csch}^2\left(\mathbf{M}\right)\right]\mathbf{x}^2 + n(\mathbf{1} \oslash \mathbf{t}^2) \tag{A.3}$$

$$K'''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = 2\left[\mathbf{D}^3 \odot \coth\left(\mathbf{M}\right) \odot \operatorname{csch}^2\left(\mathbf{M}\right)\right]\mathbf{x}^3 - 2n(\mathbf{1} \oslash \mathbf{t}^3). \tag{A.4}$$

Now differentiating the CGF and derivatives with respect $\mathbf{x}$ we have

$$\frac{\partial}{\partial \mathbf{x}} K_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \mathbf{t}\mathbf{1}^T \odot \left[\mathbf{H} + \mathbf{D} \odot \coth\left(\mathbf{M}\right)\right] - \mathbf{1}\left(\mathbf{1} \oslash \mathbf{x}\right)^T \tag{A.5}$$

$$\frac{\partial}{\partial \mathbf{x}} K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \mathbf{H} + \mathbf{D} \odot \coth\left(\mathbf{M}\right) - \mathbf{D} \odot \mathbf{M} \odot \operatorname{csch}^2\left(\mathbf{M}\right) \tag{A.6}$$

$$\begin{aligned}\frac{\partial}{\partial \mathbf{x}} K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) =& 2\mathbf{D}^2 \odot \operatorname{csch}^2\left(\mathbf{M}\right) \\ & \odot \left[\left(\mathbf{D} \odot \mathbf{t}\left(\mathbf{x}^2\right)^T\right) \odot \coth\left(\mathbf{M}\right) - \mathbf{1}\mathbf{x}^T\right].\end{aligned} \tag{A.7}$$

A.2 Exponential clipping

In the case of exponential clipping, λ is the rate of the exponential for which elements of $\mathbf{G}$ are drawn. Letting $\boldsymbol{\Lambda} = \lambda \mathbf{1}_m \mathbf{1}_n^T$ and $\mathbf{C} = \mathbf{S} \odot \left(\mathbf{t}\mathbf{x}^T\right)$ derivative are given as

$$K_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \frac{\sigma^2 \mathbf{t}^2}{2} + \mathbf{t} \odot \mathbf{H}\mathbf{x} + \left(\mathbf{A} \odot \ln\left[\boldsymbol{\Lambda} \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)\right]\right)\mathbf{1}, \tag{A.8}$$

$$K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \sigma^2 \mathbf{t} + \mathbf{H}\mathbf{x} + \left[\mathbf{A} \odot \mathbf{S} \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)\right]\mathbf{x}, \tag{A.9}$$

$$K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \sigma^2 \mathbf{1} + \left[\mathbf{A} \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)^2\right]\mathbf{x}^2, \tag{A.10}$$

$$K'''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = 2\left[\mathbf{A} \odot \mathbf{S} \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)^3\right]\mathbf{x}^3. \tag{A.11}$$

Now differentiating the CGF and derivatives with respect $\mathbf{x}$ we have

$$\frac{\partial}{\partial \mathbf{x}} K_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) = \left(\mathbf{t}\mathbf{1}^T\right) \odot \left[\mathbf{H} + \mathbf{A} \odot \mathbf{S} \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)\right] \tag{A.12}$$

$$\begin{aligned}\frac{\partial}{\partial \mathbf{x}} K'_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) =& \mathbf{H} + \mathbf{A} \odot \left[\mathbf{S} \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)\right. \\ & \left. + \left(\mathbf{t}\mathbf{x}^T\right) \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)^2\right]\end{aligned} \tag{A.13}$$

$$\begin{aligned}\frac{\partial}{\partial \mathbf{x}} K''_{\mathbf{Gx}+\boldsymbol{\eta}}(\mathbf{t}) =& 2\mathbf{A} \odot \left[\left(\mathbf{1}\mathbf{x}^T\right) \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)^2\right. \\ & \left. + \mathbf{S} \odot \left(\mathbf{t}\left(\mathbf{x}^2\right)^T\right) \oslash \left(\boldsymbol{\Lambda} - \mathbf{C}\right)^3\right].\end{aligned} \tag{A.14}$$

Appendix B

Performance Plots for Hermite Interpolation Models

We have included several plots on the following pages showing the performance of trust region solvers using standard interpolation models versus the Hermite interpolation models discussed in Chapter 4. In particular, we show scaled function values in Figures B.1 and B.2 (see Eq. 4.48) and gradient norms, $\|\mathbf{g}\|$, for the *polynomial model* by iteration count in Figures B.3 and B.4. We track the model gradient since, in practice, we do not have access to the true gradient at each iteration. When the gradient at the current iterate is interpolated, the true and model gradients match.

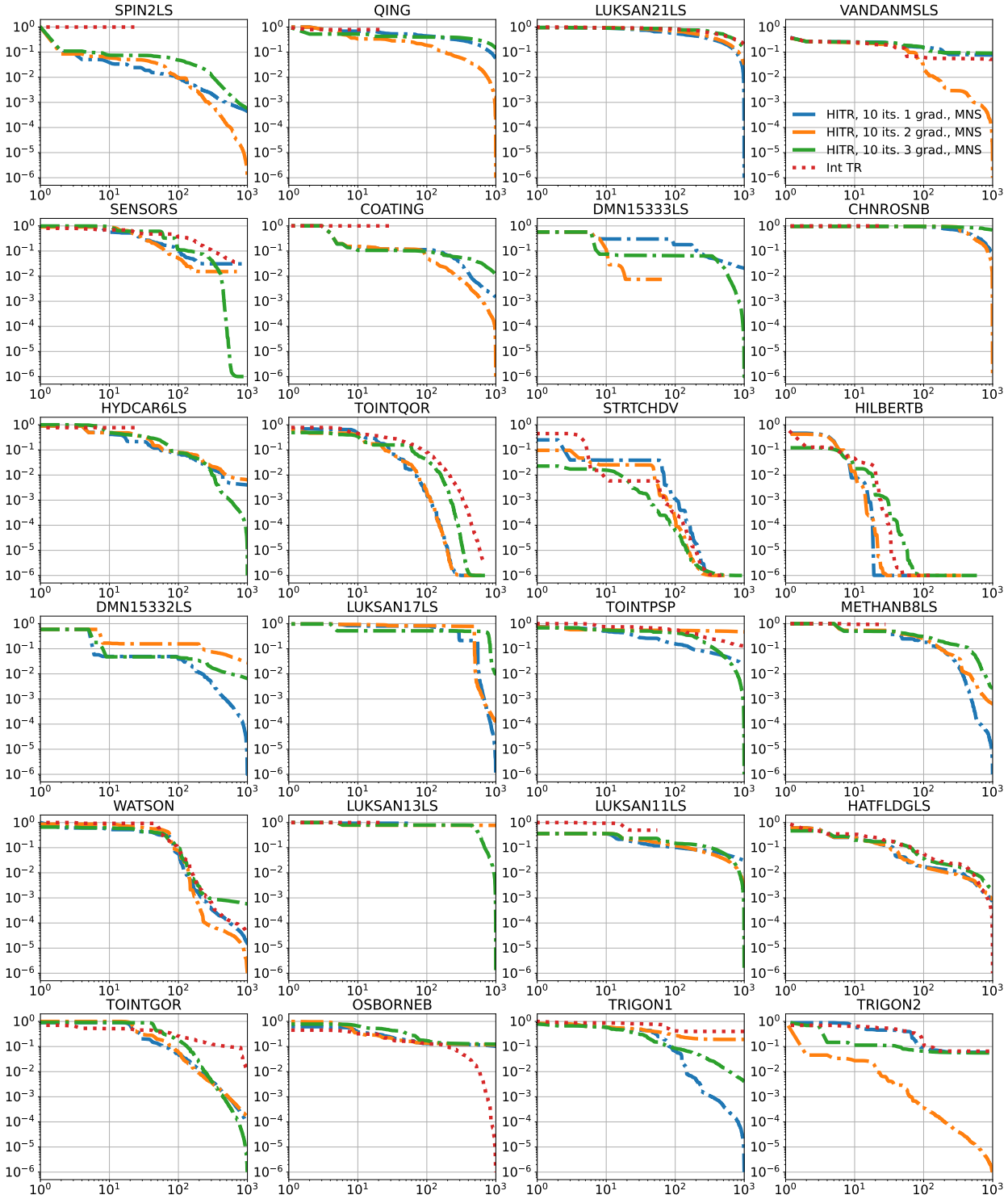

Figure B.1: Scaled function values (y-axis) at each iteration (x-axis) using minimum norm solution (MNS) for Hermite TR model with 1, 2, or 3 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.

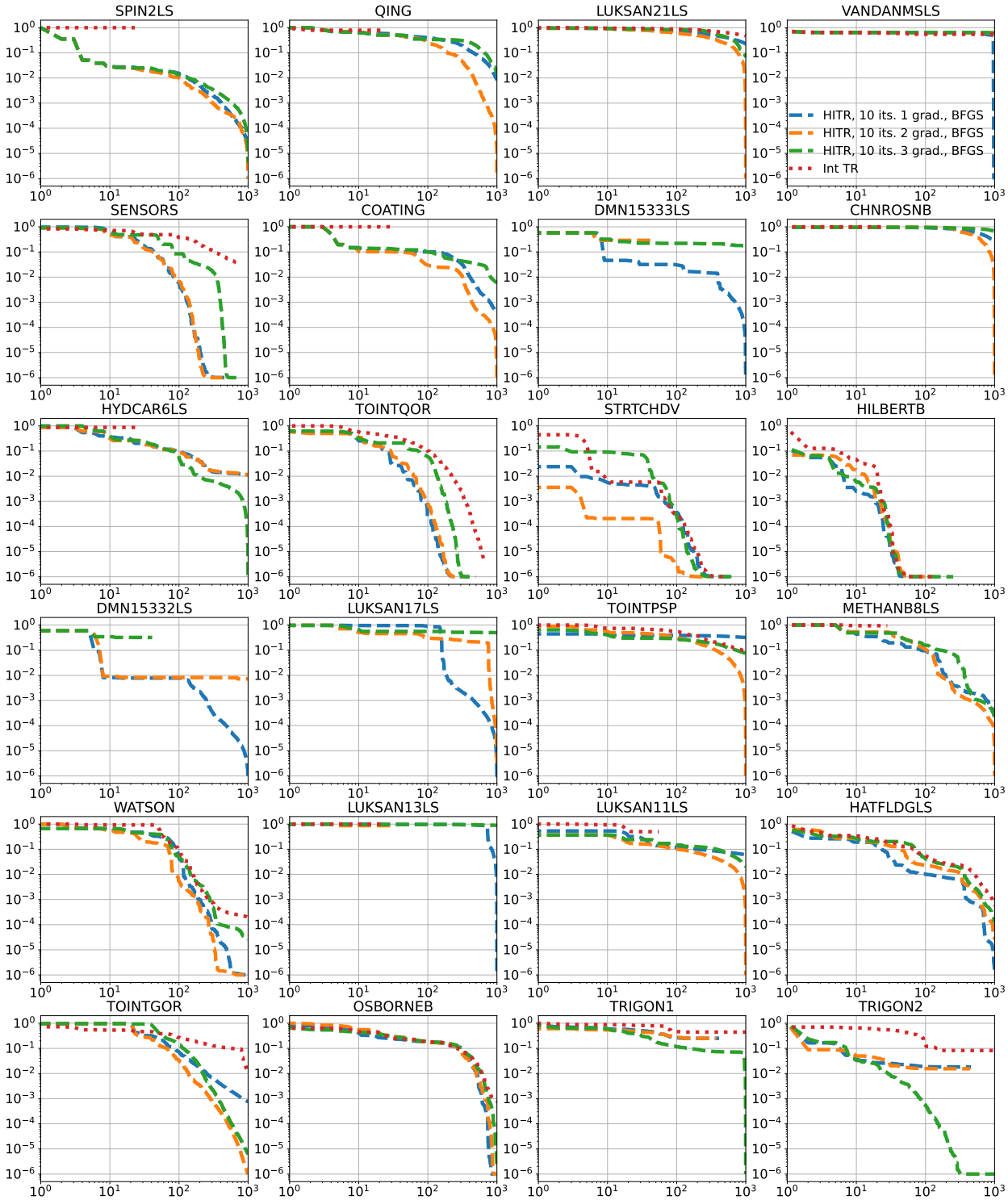

Figure B.2: Scaled function values (y-axis) at each iteration (x-axis) regularized to BFGS approximate Hessian for Hermite TR model with 1, 2, or 3 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.

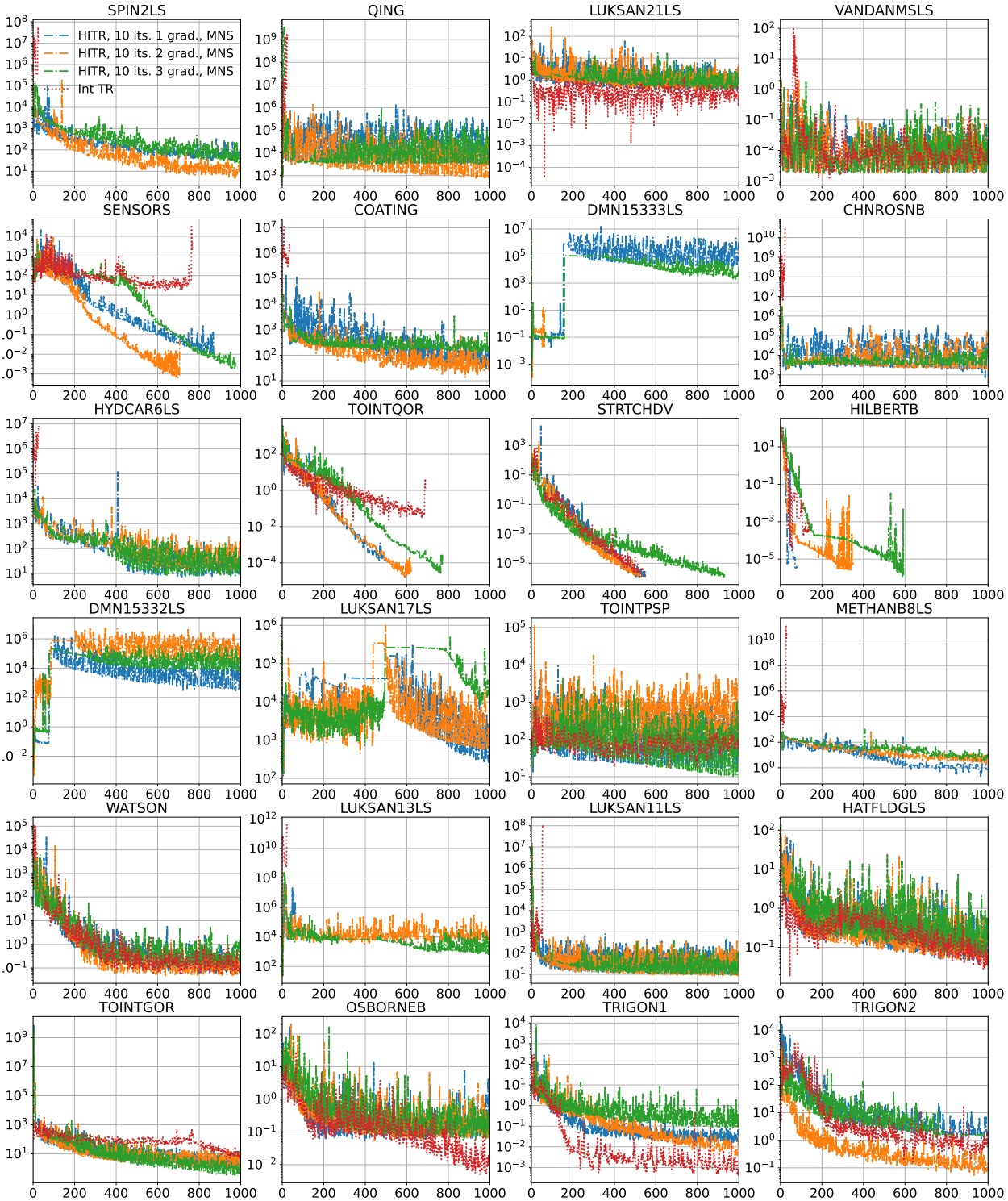

Figure B.3: Model gradient norms $\|\mathbf{g}\|$ (y-axis) at each iteration (x-axis) using minimum norm solution (MNS) for Hermite TR model with 1, 2, or 3 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.

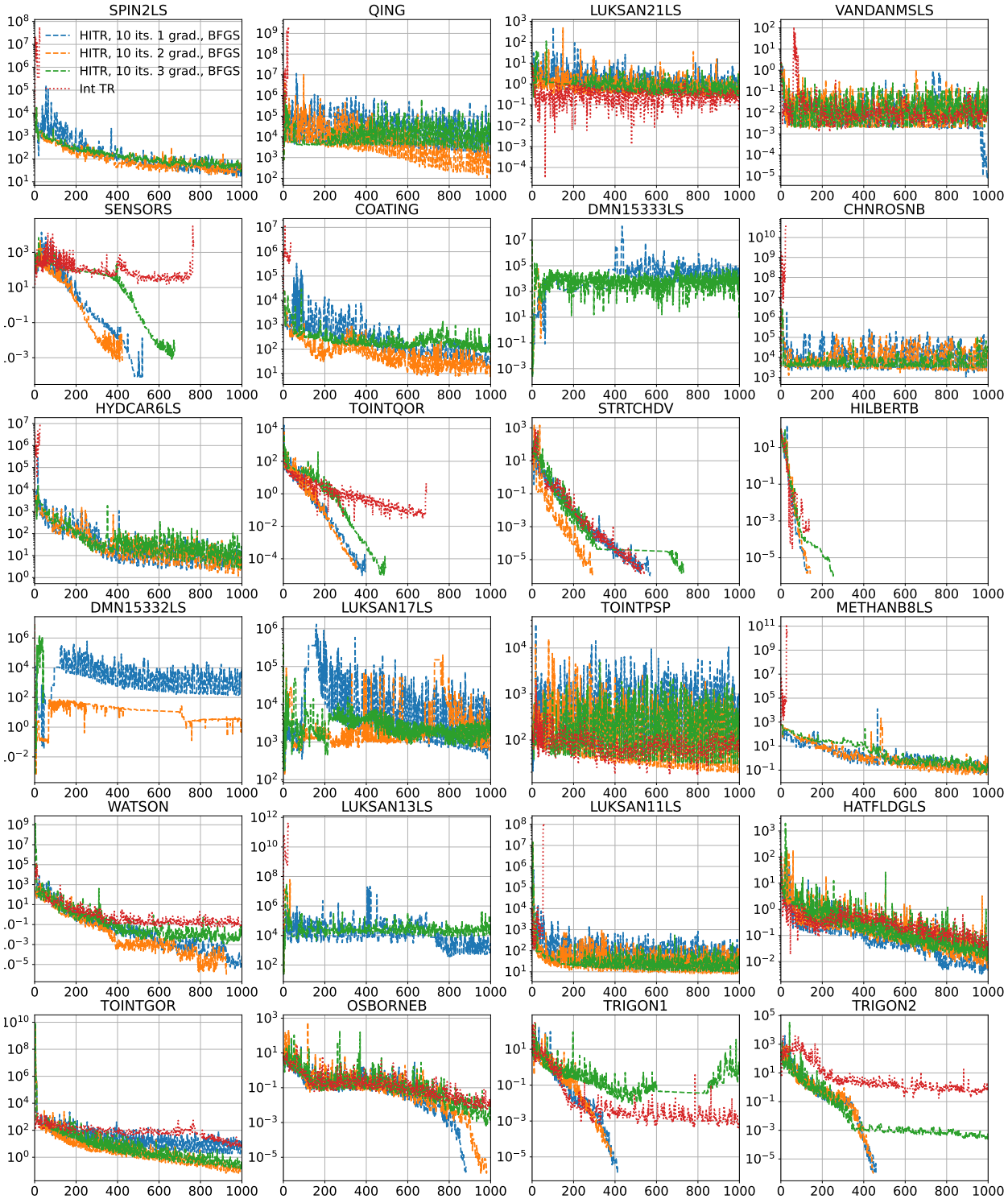

Figure B.4: Model gradient norms $\|\mathbf{g}\|$ (y-axis) at each iteration (x-axis) regularized to BFGS approximate Hessian for Hermite TR model with 1, 2, or 3 gradients interpolated. New gradient computed every 10 iterations. Standard interpolation based TR model included for comparison.