Optimizing Optimization: Scalable Convex Programming with Proximal Operators

Matt Wytock

March 2016
CMU-ML-16-100

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:
J. Zico Kolter, Chair
Ryan Tibshirani
Geoffrey Gordon
Stephen Boyd, Stanford University
Arunava Majumdar, Stanford University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant number IIS1320402, the Air Force Office of Scientific Research under grant number FA95501010247, the Office of Naval Research under grant number N000141410018, the Duquesne Light Company, the Thomas and Stacey Siebel Foundation, and a gift from Google, Inc.

Keywords: convex optimization, proximal operator, operator splitting, Newton method, sparsity, graphical model, wind power, forecasting, energy disaggregation, microgrids

For Audra

Abstract

Convex optimization has developed a wide variety of useful tools critical to many applications in machine learning. However, unlike linear and quadratic programming, general convex solvers have not yet reached sufficient maturity to fully decouple the convex programming model from the numerical algorithms required for implementation. Especially as datasets grow in size, there is a significant gap in speed and scalability between general solvers and specialized algorithms.

This thesis addresses this gap with a new model for convex programming based on an intermediate representation of convex problems as a sum of functions with efficient proximal operators. This representation serves two purposes: 1) many problems can be expressed in terms of functions with simple proximal operators, and 2) the proximal operator form serves as a general interface to any specialized algorithm that can incorporate additional ℓ_2-regularization. On a single CPU core, numerical results demonstrate that the prox-affine form results in significantly faster algorithms than existing general solvers based on conic forms. In addition, splitting problems into separable sums is attractive from the perspective of distributing solver work amongst multiple cores and machines.

We apply large-scale convex programming to several problems arising from building the next-generation, information-enabled electrical grid. In these problems (as is common in many domains) large, high-dimensional datasets present opportunities for novel data-driven solutions. We present approaches based on convex models for several problems: probabilistic forecasting of electricity generation and demand, preventing failures in microgrids and source separation for whole-home energy disaggregation.

Acknowledgments

Zico Kolter, my advisor, has been an incredible resource and a key influence in shaping my research over the course of my graduate career. It was Zico who first exposed me to the wide range of exciting problems in energy and I still recall vividly our first conversations about how machine learning and data will transform this domain. To this day, I am incredibly impressed with his capacity to rapidly assimilate a vast range of topics spanning linear algebra, optimization and machine learning as well as his ability to distill research in these areas into its most basic atoms and communicate those ideas concisely. I have been incredibly fortunate to be advised by him as well as an entire thesis committee that embodies the broad-based approach to research that I strive to emulate in my own work. Furthermore, I am grateful to Stephen Boyd for hosting me at Stanford and for fruitful discussions around the past, present and future of optimization. I also thank Arun Majumdar for providing guidance and opportunities in working on new high-impact projects beyond the boundaries of traditional computer science.

My path to pursuing the Ph.D. began not in academia, but at Google, where I witnessed the application of state-of-the-art research to real problems. During this period, I learned a great deal from an incredible cast of colleagues, but one in particular has been an important mentor over the past ten years. Ramanthan V. Guha, whose project I was assigned to on my first day, has taught me a great deal and in particular forcefully encouraged me to pursue graduate study. Guha once said that his own graduate advisor taught him "how to think" and it is clear to me that through our many interactions over the years, he has profoundly shaped my own thought processes on research and technology.

Finally, this dissertation would not have been possible without the support of my family. Foremost, my wife Audra, who selflessly moved across the country from California to Pittsburgh to support my studies. In many ways, the growth of our family has paralleled the Ph.D. journey—our wedding taking place in my first year, followed by the the birth of our son in my final year. Audra has been an amazing partner in both undertakings and in particular played a pivotal role in realizing this thesis. In addition, the first year with our son, Preston, reminds me of my deep appreciation for my own parents who have been inspirational role models. I thank them for teaching me the importance of curiosity, education, and effecting change on the world; I hope that my own work and life will one day provide the same example for my son.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convex optimization, with its robust theory and rich toolbox, has found many applications, especially in machine learning and control. One especially useful set of tools are convex programming frameworks based on *disciplined convex programming* (DCP) (e.g. CVX [53]), which provide high-level specification languages for convex optimization problems, allowing a wide variety of problems to be solved with general algorithms. The appeal of these tools is that they decouple the task of formulating the mathematical optimization model from the implementation of numerical algorithms—two tasks that involve different concerns: mathematical modeling of the problem domain vs. programming efficient numerical routines. The decoupling of these concerns allows domain practitioners to focus on developing convex models for particular problems, while programming and optimization experts develop general methods, algorithms and code that can be applied to solve these problems.

The existing approach to general convex programming has relied heavily on cone solvers and in particular primal-dual interior point methods [87]. These methods serve this purpose well as they are robust to problem inputs and able to find highly accurate solutions in polynomial time. For a small problem (i.e., megabyte-scale data), the existing approach of transforming to cone form and solving with a general cone solver is often sufficient, especially in the prototyping phase when the model is being developed. However, on larger datasets and problem sizes, this approach encounters major scalability issues rendering general convex programming frameworks far less applicable. In practice, convex methods can only be applied to "big data" with specialized algorithms developed for a single problem or class of problems, requiring custom implementations to apply even the most basic optimization techniques (e.g. gradient descent). The development of these special case numerical methods is time-consuming, error prone and suffers from a lack of the extensibility that is so easily provided by the high-level convex programming frameworks.

The aim of this thesis is to address the gap between general convex programming and specialized algorithms, bringing the scalability achievable with specialized methods to the declarative style of convex programming frameworks. Our approach adopts the disciplined convex programming syntax and interface but develops new methods for transforming and solving problems specified in this form, targeting new classes of algorithms beyond cone solvers. In essence, this requires increasing the functionality of the *compiler*, the system responsible for transforming input problems into a solvable form. In particular, we introduce a new representation for general convex programming: a sum of functions with efficient *proximal operators*. This representation

allows us to apply operator splitting algorithms (e.g. ADMM [19]), a class of methods which has received considerable attention in recent years and are in fact often the method of choice for specialized algorithms. From the perspective of general convex programming, operator splitting is appealing as it can support a wide variety of convex problems (it is in fact a generalization of cone form) while still maintaining significant problem structure that can be exploited at the algorithmic level through efficient proximal operator implementations. Clearly, due to the enormous popularity of convex optimization, there are a large number of problems and specialized algorithms and thus completely closing the gap with general convex programming is outside the scope of this work; however, on many common problems, our approach improves on existing general methods by multiple orders of magnitude, approaching the speed of specialized solvers.

In terms of related work, the general challenge of coping with large datasets has received significant attention from both research and industry. From a software perspective, frameworks such as MapReduce [32], Spark [147], Pregel [80], GraphLab [78], Dataflow [4] and others each provide abstractions for implementing parallel algorithms as well as implementations of the systems required for executing these algorithms in various large-scale computing environments. Although these frameworks substantially ease the burden of developing and deploying distributed systems for parallel computing, they operate at a significantly lower level than convex programming frameworks. In addition, they are primarily focused on distributing computation across network nodes whereas in scaling general convex programming, there is already a significant gap between general methods and specialized algorithms even on problems that comfortably fit on a single machine. Ultimately, these frameworks are complementary to our goals and our methods are developed so that they can be implemented and deployed on these existing systems.

In the area of large-scale machine learning, a number of frameworks providing efficient implementations of neural networks have recently become popular, coinciding with the surge in popularity of deep learning. Frameworks such as Torch [1], Theano [13], Caffe [64] and TensorFlow [2] allow deep learning practitioners to rapidly develop models in a high-level language appropriate to neural network modeling. The design and implementation of these systems is highly related to the challenge of scaling general convex programming in that both rely certain on certain basic primitives, e.g. fast CPU- or GPU-based linear algebra and communication between processing units. However, deep learning is rather different mathematically from convex optimization, leading to a different modeling language and different algorithmic approaches for optimization. The core function of a general convex programming framework, to translate a general convex problem to a solvable form, is basically nonexistent in neural networks; instead, neural nets involve basic computational units that are optimized directly via gradients and backpropagation. Nonetheless, the popularity of these frameworks motivates our desire to provide a high-level declarative model with similar scalability for general convex programming.

In addition, our work on general convex programming is inspired by our own experience in developing specialized algorithms for convex models. In Part II, we present specialized algorithms for several convex optimization problems: the sparse Gaussian conditional random field [138], the sparse linear-quadratic regulator [136], and the group fused lasso [141]. Each of these models is of independent interest, having been proposed by researchers in several different fields in recent years with many different applications. However, their adoption has been somewhat limited due to the computational difficultly of the optimization problems involved, motivating our development of these specialized methods. At a high level, our approach to each of these

problems is based on specialized Newton methods which exploit sparsity at an algorithmic level through the use of coordinate descent algorithms. Toward the broader goal of this thesis—the development of general convex programming methods—these problems and algorithms serve two purposes: 1) as advanced examples of convex optimization problems that should (eventually) be supported by general convex programming frameworks achieving similar performance as our specialized algorithms; and 2) as examples of proximal operators that could be incorporated in our existing framework for general convex programming as described in Part I.

In Part III we present applications of large-scale convex programming to problems arising from the development of the next-generation electrical grid. In our first application, we develop probabilistic forecasting models for balancing supply and demand in the electricity grid—it is our contention that the integration of fundamentally uncertain renewable energy sources (e.g. wind, solar) requires models capable of accurate probabilistic predictions incorporating spatiotemporal correlations as opposed to classical point forecasts. In our second application, we develop disaggregation models for understanding energy end-use from millions of residential smart meters, an important task for improving the efficiency of residential consumption. Finally, our third application considers a machine learning approach to prevent failures in microgrids by learning safety parameters from data.

To summarize, the main contributions of this thesis are:

1. A new approach to general convex programming based on transforming problems to sums of functions with computationally efficient proximal operators [133, 142].
2. Specialized algorithms for several problems from machine learning and control: the sparse Gaussian conditional random field [138], the sparse linear-quadratic regulator [136] and the group fused lasso [141].
3. Convex models for large-scale energy applications: probabilistic forecasting [137], energy disaggregation [139] and predicting failures in microgrids [140].

Part I

Scalable Convex Programming

Chapter 2

Background

Our approach to general convex programming combines the interface and syntax of disciplined convex programming (DCP) with algorithms based on proximal operators and operator splitting. Disciplined convex programming provides a declarative model for programming convex optimization problems in a natural mathematical syntax as well as methods for transforming problems to a general form, historically a cone problem. In our work we extend the DCP approach to target proximal algorithms, an appealing approach due to the flexibility provided by proximal operators—conceptually, a proximal operator is defined for any function although certain functions give rise to especially efficient implementations. In this Chapter, we review disciplined convex programming as well as the proximal operators and operator splitting techniques that will later be employed to solve general convex problems.

2.1 Disciplined convex programming

Consider a convex optimization problem

$$\text{minimize} \quad \|Ax - b\|_2^2 + \lambda\|x\|_1 \tag{2.1}$$

with optimization variables $x \in \mathbb{R}^n$, problem data $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and regularization parameter $\lambda \geq 0$. Disciplined convex programming provides a syntax for writing this problem in a high-level programming language, rules for verifying its properties (in particular, convexity) and methods for transforming the problem to a form that can be solved with a general cone solver. Frameworks based on disciplined convex programming (e.g. CVX [53], CVXPY [34] and Convex.jl [127]) provide this functionality in several popular environments for numerical computing. For the purposes of this thesis, we adopt the CVXPY syntax which is implemented as standard classes and functions in Python. In this syntax, the optimization problem above can be written in a few lines of Python.

```
x = Variable(n)
f = sum_squares(A*x - b) + lam*norm1(x)
prob = Problem(Minimize(f))
```

As such, these libraries and other similar modeling frameworks (e.g. YALMIP [77]) have substantially lowered the barrier to quickly prototyping convex programs without the need to manually convert them to a form that can be fed directly into a numerical solver.

The innovation of these frameworks is in providing a separation between the specification of the convex programming problem and the numerical routines required to find a solution. In doing so, they enable domain practitioners whose primary focus is the application to rapidly experiment with convex models using numerical routines developed by optimization experts. Two key elements enable this separation: 1) a library of *atoms*—functions with known properties and numerical implementations; and 2) a set of rules for composing these functions in a way that can easily be verified to be convex. Importantly, the DCP ruleset is sufficient but *not necessary* for a problem to be convex: for instance, the log-sum-exp function

$$f(x) = \log(e^{x_1} + \cdots + e^{x_n}) \tag{2.2}$$

is convex, but is a composition of a concave monotonic and convex function, which does not imply convexity using standard rules. However, log-sum-exp *can* be incorporated in a DCP framework through direct implementation as an atom that is convex (and monotonic in its arguments). Adding new atoms to the DCP library is straightforward and in practice, most convex problems can be written using the DCP ruleset with a relatively small set of atoms.

Conceptually, the specification and verification steps in disciplined convex programming are agnostic to the numerical routines used to produce a solution. However, to the best of our knowledge, all DCP-based frameworks target the same canonical cone representation (originally proposed in [52]) and nearly all general solvers available to these frameworks are based on interior point methods.[1] Although interior point methods are robust, providing highly accurate solutions with provably polynomial time algorithms, they are often intractable for even moderately-sized problems. Fundamentally, our approach differs from the conventional approach by employing a different set of transformations to target a new class of algorithms based on proximal operators and operator splitting techniques.

2.1.1 Expressions and atoms

The basic building block of disciplined convex programming are expressions representing vector-valued functions, $f : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k} \to \mathbb{R}^m$. We can describe these expressions programmatically using a language with only three types of elements.

- *Constant.* A literal value (`10`) or a symbol representing a data constant (`A`)
- *Variable*. A symbol representing an optimization variable (`x`)
- *Atom*. A vector-valued function operating on one or more expressions, e.g. `norm1(x)`, `Ax`, or `x + y`

In essence, this simply formalizes the notion of variables and constants transformed by a set of functions (called *atoms* in DCP) with known properties. For example, the function $f : \mathbb{R}^n \to \mathbb{R}$

[1]The exception is recent work on the splitting conic solver [91] which employs a first-order method on the canonical cone representation. We compare to this solver extensively in Section 3.3 and see that on many problems Epsilon can be much faster.

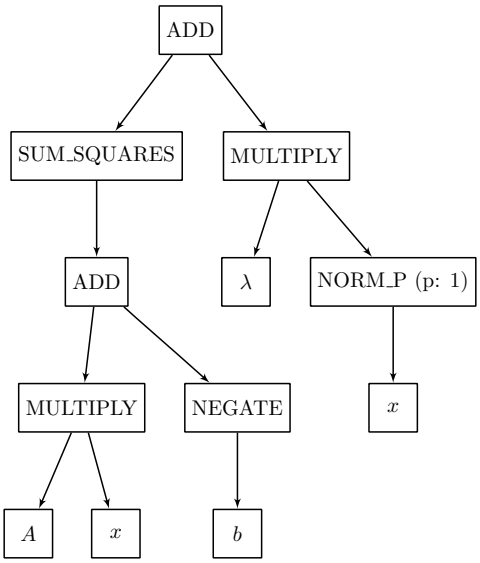

Figure 2.1: Example abstract syntax tree for a DCP expression.

defined by

$$f(x) = \|Ax - b\|_2^2 + \lambda\|x\|_1 \tag{2.3}$$

can be written as the DCP expression (in Python syntax provided by CVXPY):

```
f = sum_squares(A*x - b) + lam*norm1(x)
```

where `A`, `b`, `lam` are constants, `x` is a variable and `sum_squares`, `norm1`, `+`, `-` and `*` are all atoms provided by the DCP framework (in this case the binary operators in Python have been overloaded to produce the appropriate atom). When manipulating expressions programmatically, it is convenient to conceive of them as abstract syntax trees (ASTs) with constants and variables at the leaves and atoms at the internal nodes. The AST for this particular example is shown in Figure 2.1.

The fundamental characteristic of disciplined convex programming frameworks is a set of rules that can be used to verify the properties of expressions programmatically. These rules associate with each node of the AST a set of attributes that characterize the properties of the mathematical expression represented by that node.

- *Size*. The output dimension of the function represented by this expression.
- *Curvature*. The curvature of the expression in all of its inputs: `constant`, `affine`, `convex`, `concave`, or `unknown`.
- *Sign*. The sign of the function: `positive`, `negative`, `zero` or `unknown`.

Then, the DCP verification process amounts to computing these attributes for each node in the AST—once these attributes have been computed for each node, the mathematical properties of the entire expression and each sub-expression can be determined directly from attributes associated with their respective nodes.

2.1.2 Disciplined convex programming rules

We next specify the rules for formulating a convex optimization problem in the DCP framework. The first set of rules specify how DCP expressions are arranged to form a valid convex problem; first, the objective must be scalar-valued and be a valid minimization or maximization problem.

$$\texttt{minimize(convex)} \ \textit{or} \ \texttt{maximize(concave)}$$

In addition, it can include zero or more convex constraints.

$$\begin{aligned}
&\texttt{affine} = \texttt{affine}\\
&\texttt{convex} \leq \texttt{concave}\\
&\texttt{concave} \geq \texttt{convex}\\
&\texttt{(affine, ..., affine)} \in \textit{convex set}
\end{aligned}$$

Note that in describing the curvature of an expression, stronger notions of curvature imply the more general ones.

$$\texttt{constant} \implies \texttt{affine} \implies \texttt{convex} \ \textit{and} \ \texttt{concave}$$

Thus, in the rules described below an expression with constant curvature is also affine, convex and concave.

The curvature of the expressions in the objective and the constraints is verified according to the DCP attributes which are computed for each node in the AST in a bottom-up fashion. In order to define these rules, we first need to define some additional properties for each atom.

- *Monotonicity*. The monotonicity of the function in each of its arguments: `increasing`, `decreasing`, `signed`, `nonmonotonic` or `unknown`.
- *Function curvature*. The curvature of the function independent of its inputs: `constant`, `affine`, `convex`, `concave`, or `unknown`.

Signed monotonicity handles "absolute value"-style functions which are nondecreasing for positive inputs and nonincreasing for negative ones. Importantly, these attributes depend only on the atom itself and not the entire expression. In particular the *function curvature* attribute does not depend on a function's arguments (which may themselves be other atoms) unlike the *curvature* attribute defined in Section 2.1.1.

With these additional properties, the curvature of an expression is computed in each of its arguments according to the following rules which depend on the function in question as well as the DCP attributes of its arguments.

$$\begin{aligned}
&\textit{argument curvature} = \texttt{constant} \implies \texttt{constant}\\
&\textit{argument curvature} = \texttt{affine} \implies \textit{function curvature}\\
&\textit{monotonicity} = \texttt{increasing} \implies \textit{argument curvature} + \textit{function curvature}\\
&\textit{monotonicity} = \texttt{decreasing} \implies \textit{argument curvature} - \textit{function curvature}
\end{aligned}$$

In the above rules, the binary operator "$+$" applied to two curvature types returns the least general

curvature type encompassing both, e.g.

$$\texttt{convex} + \texttt{convex} = \texttt{convex}$$
$$\texttt{convex} + \texttt{affine} = \texttt{convex}$$
$$\texttt{affine} + \texttt{constant} = \texttt{affine}$$
$$\texttt{convex} + \texttt{concave} = \texttt{unknown}$$

Similarly, the binary operator "$-$" performs the same operation after first modifying the second argument from convex to concave and vice versa.

There is one final set of rules applying only to the special case of a convex function with signed monotonicity (e.g. absolute value).

$$\textit{argument curvature} = \texttt{convex},\ \textit{argument sign} = \texttt{positive} \implies \textit{function curvature}$$
$$\textit{argument curvature} = \texttt{concave},\ \textit{argument sign} = \texttt{negative} \implies \textit{function curvature}$$

As a simple example of the above rules consider the atom for binary addition. This atom has affine curvature and is monotonically increasing in both of its two arguments. Thus the curvature of an addition expression largely mirrors the curvature of its arguments: in the case where both arguments are convex then the overall expression will also be convex—similarly for concave, affine or constant expressions. However, if one argument is convex and the other concave this will result in an expression with unknown curvature which (in general) represents a nonconvex function.

Unlike addition, multiplication tends to be significantly restricted in disciplined convex programming. In general, inferring the curvature of multiplication expressions is a difficult problem and thus all existing DCP frameworks require at least one constant argument to the binary multiplication atom. This restriction allows for a straightforward application of the curvature rules: the curvature for multiplicaton atom is computed as a single-argument atom with affine curvature and monotonicity determined by the sign of the constant argument.

In addition to computing the DCP attribute for curvature, we must also compute signs of various expressions based on the signs of their sub-expressions. This operation is straightforward and here we give the standard rules for the binary operators.

$$\texttt{positive} \times \texttt{positive} = \texttt{positive}$$
$$\texttt{positive} \times \texttt{negative} = \texttt{negative}$$
$$\texttt{negative} \times \texttt{negative} = \texttt{positive}$$
$$\texttt{positive} + \texttt{positive} = \texttt{positive}$$
$$\texttt{negative} + \texttt{negative} = \texttt{negative}$$
$$\texttt{zero} \times \textit{any} = \texttt{zero}$$
$$\texttt{zero} + \textit{any} = \textit{any}$$

In the above rules, *any* refers to an expression with arbitrary sign, e.g. adding zero to an expression with any sign retains the same sign.

Taken together, these rules describe the basic elements used by the DCP framework to compute DCP attributes for curvature and sign of an expression based on the properties of subexpressions for most of the common cases. In addition, each atom is also responsible for specifying its dimension as a function of its arguments. Finally, we note that although the rules above

cover the most common cases, each individual atom can override these definitions to specify custom behavior in order to handle important special cases capturing the properties of particular functions.

2.1.3 Conic graph implementations

After convexity of the problem has been verified by the DCP rules, traditionally the next step in the DCP framework is to convert problems into a standard conic form. This is accomplished using the *graph implementation*: a representation of each atom as the solution to a linear cone program. For example, the ℓ_1-norm can be expressed as

$$\|x\|_1 \equiv \underset{t}{\text{minimize}}\ 1^T t, \ \text{ subject to } -t \leq x \leq t. \tag{2.4}$$

Thus, when the transformation step encounters an ℓ_1-norm, it can be replaced with the linear cone problem above by simply introducing the t variable, modifying the expression to be that of the objective and adding the constraints above.

By applying these transformation repeatedly, the entire problem is reduced to a single linear cone problem

$$\begin{aligned} \text{minimize } & c^T x \\ \text{subject to } & Ax = b \\ & x \in \mathcal{K}, \end{aligned} \tag{2.5}$$

which can then be solved by standard cone solvers. An important detail of this process is that the graph implementation for each atom can depend only on cones supported by the general solvers available to the DCP framework, which in practice is a small set of cones:

- *Nonnegative orthant*. $\{x \mid x \geq 0\}$
- *Second-order cone*. $\{(x,t) \mid \|x\|_2 \leq t\}$
- *Positive semidefinite cone*. $\{X \in \mathbb{S}^n \mid X \succeq 0\}$
- *Exponential cone*. $\{(x,y,z) \mid y > 0, ye^{x/y} \leq z\} \cup \{(x,y,z) \mid x \leq 0, y = 0, z \geq 0\}$

These cones are supported by a number of solvers (e.g. SeDuMi [120], SDPT3 [125], Gurobi [95], MOSEK [85], GLPK [79], CVXOPT [28], ECOS [35] and SCS [91]), although it is the case that not all solvers support all cones, requiring the DCP framework to choose the appropriate solver based on the problem.

The most common approach to solving conic problems in standard form are primal-dual interior point methods. These methods are robust to problem inputs and find highly accurate solutions in a moderate amount of time on small problems—a statement that is made rigorous by analysis (via self-concordance [88]) which bounds the number of iterations required for convergence independent of problem scaling factors. However, each Newton iteration of the primal-dual method requires solving a large linear system, which quickly becomes intractable as the problem grows. This problem can be exacerbated by the cone representation, which often requires many auxiliary variables to represent a problem in standard form. For example, representing $\|x\|_1$ in cone form with $x \in \mathbb{R}^n$ requires the introduction of an additional variable $t \in \mathbb{R}^n$, doubling the number of variables. These difficulties motivate our approach in adapting the DCP framework to target a new class of algorithms based on operator splitting.

2.2 Proximal operators and algorithms

The *proximal operator* of a function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is

$$\text{prox}_f(v) = \underset{x}{\text{argmin}} \left(f(x) + (1/2)\|x - v\|_2^2 \right) \tag{2.6}$$

where $\|\cdot\|_2$ denotes the usual Euclidean norm. Conceptually, the proximal operator generalizes set projections to functions—given an input $v \in \mathbb{R}^n$ we find a point x that is close to v (in the ℓ_2-norm) but also makes f small. Naturally, there is a tradeoff between these two objectives which we control explicitly with parameter $\lambda > 0$

$$\text{prox}_{\lambda f}(v) = \underset{x}{\text{argmin}} \left(f(x) + (1/2\lambda)\|x - v\|_2^2 \right). \tag{2.7}$$

In this form, larger values of λ increase the weight given to f while smaller values prefer points close to v. We recover the operator for projection onto a set $\mathcal{C}$ with the indicator function

$$I_{\mathcal{C}}(x) = \begin{cases} 0 & x \in \mathcal{C} \\ \infty & x \notin \mathcal{C}, \end{cases} \tag{2.8}$$

since

$$\text{prox}_{I_{\mathcal{C}}}(v) = \underset{x \in C}{\text{argmin}} \|x - v\|_2 = \Pi_{\mathcal{C}}(v). \tag{2.9}$$

2.2.1 Properties of proximal operators

All proximal operators satisfy the following basic properties.

- *Scalar multiplication.* If $f(x) = \alpha g(x)$ with $\alpha > 0$, then

$$\text{prox}_{\lambda f}(v) = \text{prox}_{\alpha\lambda g}(v).$$

- *Linear addition.* If $f(x) = g(x) + c^T x$, then

$$\text{prox}_{\lambda f}(v) = \text{prox}_{\lambda g}(v - \lambda c)$$

- *Post composition.* If $f(x) = g(\alpha x + b)$ with $\alpha \neq 0$, then

$$\text{prox}_{\lambda f}(v) = \frac{1}{\alpha}\left(\text{prox}_{\alpha^2\lambda g}(\alpha v + b) - b\right).$$

We refer to the ℓ_2-regularization $(1/2)\|x - v\|_2^2$ as the *proximal term*. Multiple proximal terms can be combined into a single proximal operator

$$\underset{x}{\text{argmin}}\, f(x) + (1/2\lambda)\|x - v\|_2^2 + (1/2\rho)\|x - u\|_2^2 = \text{prox}_{\gamma f}((\gamma/\lambda)v + (\gamma/\rho)u) \tag{2.10}$$

where $\gamma = (\lambda\rho)/(\lambda + \rho)$. On the other hand, for a separable function $h(x, y) = f(x) + g(y)$, the proximal operator reduces to evaluation in parts

$$\text{prox}_h(v, u) = (\text{prox}_f(v), \text{prox}_g(u)). \tag{2.11}$$

2.2.2 Functions with simple proximal operators

In this section, we give examples of functions with simple proximal operators with closed form evaluations.

Functions on scalars. We start with several functions on scalar inputs, $x \in \mathbb{R}$.

- *Identity*. Let $f(x) = x$, $\text{prox}_{\lambda f}(v) = v - \lambda$.
- *Square*. Let $f(x) = (1/2)x^2$, $\text{prox}_{\lambda f}(v) = v/(\lambda + 1)$.
- *Absolute value*. Let $f(x) = |x|$,

$$\text{prox}_{\lambda f}(v) = \begin{cases} v + \lambda & v < -\lambda \\ 0 & |v| \leq \lambda \\ v - \lambda & v > \lambda. \end{cases}$$

- *Hinge*. Let $f(x) = \max\{x, 0\}$,

$$\text{prox}_{\lambda f}(v) = \begin{cases} v & v < 0 \\ 0 & 0 \leq v \leq \lambda \\ v - \lambda & v > \lambda. \end{cases}$$

- *Negative log*. Let $f(x) = -\log(x)$, $\text{prox}_{\lambda f}(v) = v + (1/2)\sqrt{v^2 + 4\lambda}$.
- *Inverse*. Let $f(x) = 1/x$, $\text{prox}_{\lambda f}(v)$ is the solution to the polynomial

$$x^3 - vx^2 - \lambda = 0$$

  which can be found with the cubic formula.
- *Nonnegative*. Let $f(x) = I(x \geq 0)$, $\text{prox}_{\lambda f}(v) = (x)_+$.
- *Box*. Let $f(x) = I(a \leq x \leq b)$

$$\text{prox}_{\lambda f}(v) = \begin{cases} a & v < a \\ v & a \leq v \leq b \\ b & v > b. \end{cases}$$

These functions functions can be applied elementwise to form functions on vectors. For example, the ℓ_1-norm

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \tag{2.12}$$

and the sum-of-squares function

$$(1/2)\|x\|_2^2 = (1/2)\sum_{i=1}^{n} x_i^2 \tag{2.13}$$

are both separable functions, applying elementwise to a vector $x \in \mathbb{R}^n$.

Functions on vectors. Next, we turn to functions on vectors that cannot be expressed elementwise.

- *ℓ_2-norm*. For $x \in \mathbb{R}^n$, let $f(x) = \|x\|_2$,

$$\text{prox}_{\lambda f}(v) = \begin{cases} (1 - \lambda\|v\|_2)v & \|v\|_2 \geq \lambda \\ 0 & \|v\|_2 \leq \lambda. \end{cases}$$

- *Quadratic*. For $x \in \mathbb{R}^n$, let $f(x) = (1/2)x^TQx + c^Tx$ with $Q \succeq 0$,

$$\text{prox}_{\lambda f}(v) = (I + \lambda Q)^{-1}(v - \lambda c).$$

- *Second-order cone*. For $x \in \mathbb{R}^n$ and $t \in \mathbb{R}$, let $f(x,t) = I(\|x\|_2 \leq t)$

$$\text{prox}_{\lambda f}(v,s) = \begin{cases} 0 & \|v\|_2 \leq -s \\ (v,s) & \|v\|_2 \leq s \\ (1/2)(1 + s/\|v\|_2)(v, \|v\|_2) & \|v\|_2 > |s|. \end{cases}$$

Functions on matrices. Finally, we give a few examples of functions on matrices. In this case, the proximal operator is

$$\text{prox}_{\lambda f}(V) = \underset{X}{\text{argmin}}\, f(X) + (1/2\lambda)\|X - V\|_F^2 \tag{2.14}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, the ℓ_2-norm applied elementwise to the entries of a matrix.

- *Negative log-det*. For $X \in \mathbb{S}^n$, let $f(X) = -\log|X|$,

$$\text{prox}_{\lambda f}(V) = U\tilde{X}U^T$$

  where $V = UDU^T$ is the eigenvalue decomposition and $\tilde{X}$ is the diagonal matrix with

$$\tilde{X}_{ii} = \frac{D_{ii} + \sqrt{D_{ii}^2 + 4\lambda}}{2}.$$

- *Semidefinite cone*. For $X \in \mathbb{S}^n$, let $f(X) = I(X \succeq 0)$,

$$\text{prox}_{\lambda f}(V) = \sum_{i=1}^n (d_i)_+ u_i u_i^T$$

  where $V = \sum_{i=1}^n d_i u_i u_i^T$ is the eigenvalue decomposition.

Linear composition. In general, the existence of a simple proximal operator for $f(x)$ does not imply the existence of one for $f(Ax)$. However, in a few special cases a function composed with a linear operator can be solved in closed form.

- *Linear equality*. Let $f(x) = I(Ax = b)$,

$$\text{prox}_{\lambda f}(v) = v - A^T(AA^T)^{-1}(Av - b).$$

- *Least squares*. Let $f(x) = (1/2)\|Ax - b\|_2^2$,

$$\text{prox}_{\lambda f}(v) = (I + \lambda A^TA)^{-1}(A^Tb + v).$$

In addition, functions involving linear and quadratic terms can often be combined into a single proximal operator. For example, let $f(x) = (1/2)x^T Qx + c^T x + I(Ax = b)$ with $Q \succeq 0$. The optimality conditions for the proximal operator $\mathrm{prox}_{\lambda f}(v)$ are

$$\begin{aligned} (\lambda Q + I)x + A^T y &= v - \lambda c \\ Ax &= b \end{aligned} \tag{2.15}$$

where y is the dual variable for the equality constraint. We can solve this with Gaussian elimination and back substitution

$$\begin{aligned} y &= (AA^T)^{-1}A(\lambda Q + I)^{-1}(v - \lambda c) \\ x &= (\lambda Q + I)^{-1}(v - \lambda c) - A^T y. \end{aligned} \tag{2.16}$$

In many of the examples above, the main computational cost arises from solving linear systems. Typically, proximal algorithms require applying the same proximal operator many times which allows us to amortize this cost by computing and caching the factorization required by the linear solver. Furthermore, the matrix inversion lemma can often be used to reduce the operations required for factorization. For example, least squares $(1/2)\|Ax - b\|_2^2$ may have A with $m \leq n$ in which case

$$(I + \lambda A^T A)^{-1} = I - A^T((1/\lambda)I + AA^T)^{-1}A \tag{2.17}$$

and we reduce computation by factoring $(1/\lambda)I + AA^T$ instead of $I + \lambda A^T A$.

As an example of applying the atomic proximal operators discussed in this section with the basic properties from Section 2.2.1, consider the weighted ℓ_1-norm applied to a vector $x \in \mathbb{R}^n$

$$\lambda f(x) = \lambda \|w \circ x\|_1 = \lambda \sum_{i=1}^{n} w_i |x_i|. \tag{2.18}$$

where $w \in \mathbb{R}^n$ is a vector of weights. Since this is an elementwise function, by the separable sum property and the precomposition rule,

$$(\mathrm{prox}_{\lambda f}(v))_i = \mathrm{prox}_{\lambda w_i |\cdot|}(v_i) = \begin{cases} v_i + \lambda w_i & v_i < -\lambda w_i \\ 0 & |v_i| \leq \lambda w_i \\ v_i - \lambda w_i & v_i > \lambda w_i \end{cases} \tag{2.19}$$

where in the last step we simply apply the proximal operator for the scalar absolute value to each element of v.

2.2.3 Alternating direction method of multipliers

For the purposes of this work, we are interested in algorithms which interface with optimization problems primarily through the proximal operator. In this section, we present one such algorithm based on the alternating direction method of multipliers (ADMM) [19]. For a detailed discussion of other proximal algorithms, we refer the reader to the recent monograph [97].

Given a problem of the form

$$\text{minimize} \quad f(x) + g(x) \tag{2.20}$$

we introduce a new variable z and a *consensus* equality constraint

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x = z \end{array} \tag{2.21}$$

which makes the objective separable. The alternating updates

$$\begin{aligned} x^{k+1} &:= \text{prox}_{\lambda f}(z^k - u^k) \\ z^{k+1} &:= \text{prox}_{\lambda g}(x^{k+1} + u^k) \\ u^{k+1} &:= u^k + x^{k+1} - z^{k+1} \end{aligned} \tag{2.22}$$

are applied in a round robin Gauss-Seidel fashion; k is the iteration counter and $\lambda > 0$ is an algorithm parameter. The standard motivation for ADMM comes from the augmented Lagrangian

$$L_\lambda(x, z, y) = f(x) + g(z) + y^T(x - z) + (1/\lambda)\|x - z\|_2^2 \tag{2.23}$$

where y is the dual variable corresponding to the equality constraint. The updates are derived by minimizing the augmented Lagrangian over x and z and updating a scaled version of the dual variable $u = \lambda y$ with a gradient step. We can also view ADMM from the perspective of integral control [46], in which case u is the sum of errors fed back to each proximal operator.

Chapter 3

Convex Programming with Fast Proximal and Linear Operators

In this chapter we present Epsilon, a new system for general convex programming that combines the interface of disciplined convex programming with efficient operator splitting algorithms, automating the decomposition of general convex problems into subproblems with efficient proximal operators. In the sequel, we describe the complete Epsilon system, starting with the details of the Epsilon compiler and solver in Section 3.1—like existing convex programming frameworks, Epsilon takes as input convex problems formulated with the DCP ruleset (see Section 2.1) and thus provides a natural mathematical syntax that can easily and intuitively express complex objectives and constraints. However, unlike existing frameworks which transform the problem into a standard conic form and then pass to a cone solver, Epsilon transforms the problem into a form we call *prox-affine*: a sum of "prox-friendly" functions (i.e., functions that have efficient proximal operators) composed with affine transformations. These functions include the cone projections sufficient to solve existing cone problems, but prox-affine form is a much richer representation also including a wide range of other convex functions with efficient proximal operators. Section 3.2 presents a library of efficient proximal operator implementations for many common functions along with a high-level discussion of their implementation details. As is often the case in convex optimization, for many problems the evaluation of *linear* operators accounts for the majority of time and thus we also present a library of efficient linear operators extending beyond the traditional sparse and dense matrices. In total, the resulting Epsilon system can solve a wide range of optimization problems an order of magnitude faster (or more) than existing approaches to general convex programming—we provide several examples of popular problems from statistics and machine learning in Section 3.3.

3.1 The Epsilon compiler and solver

The Epsilon system has two components: 1) the *compiler*, which transforms a DCP representation into a prox-affine form (and eventually a separable prox-affine form, to be discussed shortly), by a series of passes over the AST corresponding to the original problem; and 2) the *solver*, which solves the resulting problem using the fast implementation of these proximal and linear opera-

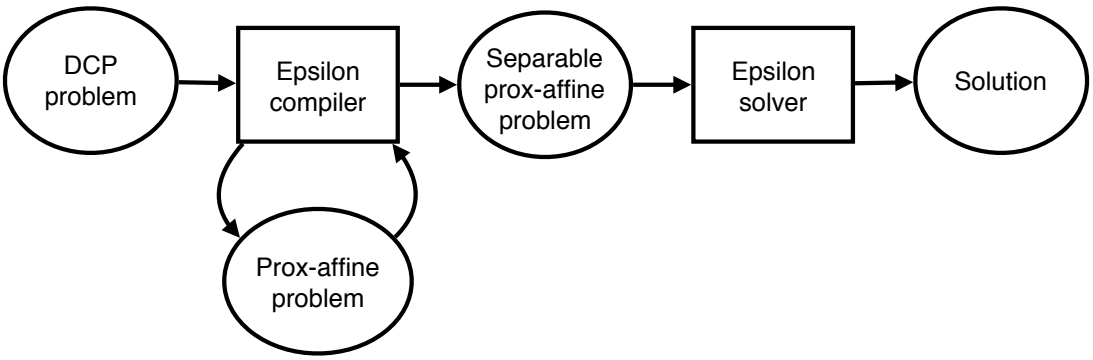

Figure 3.1: The Epsilon system: the compiler transforms a DCP-valid input problem into separable prox-affine problem to be solved by the solver. The prox-affine problem is also used as an intermediate representation internal to the compiler.

tors. An overview of the system is shown in Figure 3.1. This section describes each of these two components in detail.

Before describing the details of the prox-affine form and the compiler transformations, we emphasize that for a given DCP problem, multiple translations to prox-affine form are possible and we do not in general find the "best" representation. Our approach is rule-based, adopting various heuristics that attempt to produce a reasonably efficient prox-affine form for any problem, ensuring a valid transformation while attempting to minimize the number of auxiliary variables introduced. At a high level, the prox-affine representation balances per-iteration computational complexity and the overall number of iterations that will be required to solve a given problem. As an extreme example, we can often split problems into a large number of proximal operators which are very cheap to evaluate but will require a large number of iterations. The theoretical analysis of convergence rates for operator splitting algorithms is an active area of research (see e.g. [31, 51, 90]) but Epsilon follows the simple heuristic of minimizing the total number of proximal operators in the separable form provided that each operator can be evaluated efficiently. This is implemented with multiple passes on the prox-affine form which split the problem as needed until it it satisfies the constraints required for applying the operator splitting algorithm described in Section 3.1.4.

3.1.1 The prox-affine form

The internal representation used by the compiler, as well as the input to the solver, is a convex optimization problem in prox-affine form

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(H_i(x)), \tag{3.1}$$

where x is the optimization variable, $f_1, \ldots, f_N$ are functions with efficient proximal operators and $H_1, \ldots, H_n$ are affine transformations. The affine transformations are implemented with a library of linear operators which includes matrices (either sparse or dense), as well as special

cases like diagonal or scalar matrices and more complex linear transformations not easily represented as a single matrix, like Kronecker products. Crucially, not every proximal operator can be combined with every linear operator, but it is the job of the Epsilon compiler, described shortly, to ensure that the problem is transformed to one where the compositions of proximal and linear operators have an available implementation. For example, very few proximal operators support composition with a general dense matrix (the sum-of-squares and subspace equality constraint being some of the only instances), but several can support composition with a diagonal matrix. Representing these distinctions in Epsilon compiler is critical to deriving efficient proximal updates for the final optimization problems.

As an example of prox-affine form, the linear cone problem which forms the basis for existing disciplined convex programming systems (see Section 2.1),

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \in \mathcal{K}, \end{aligned} \tag{3.2}$$

can be represented in prox-affine form as

$$\text{minimize} \quad c^T x + I_0(Ax - b) + I_{\mathcal{K}}(x), \tag{3.3}$$

with f_1 being the identity function, H_1 being inner product with c, f_2 the indicator of the zero set, H_2 being the affine transformation $Ax - b$, f_3 the indicator of the cone $\mathcal{K}$ and H_2 the identity. Each of these functions has an efficient proximal operator; for example the proximal operator of $I_0(Ax - b)$ is simply the projection onto this subspace, the proximal operator for $I_{\mathcal{K}}$ is the cone projection, and the proximal operator for $c^T x$ is simply $v - c$ (in fact, this term can be merged with one or both of the other terms and thus only two proximal operators are necessary). As the linear cone problem is thus a special case of prox-affine form, Epsilon enjoys the same generality as existing DCP systems.

In order to apply the operator splitting algorithm, we also define the *separable* prox-affine form

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} f_i(H_i(x_i)) \\ \text{subject to} \quad & \sum_{i=1}^{N} A_i(x_i) = 0 \end{aligned} \tag{3.4}$$

which has a separable objective and explicit linear equality constraints (these are required in order to guarantee that the objective can be made separable while remaining equivalent to the original problem). As above, the affine transformations A_i are implemented by the linear operator library and can thus be represented with matrices or Kronecker products, as well as simpler forms such as diagonal or scalar matrices. The latter are especially common in the separable form because they are often introduced for representing the consensus constraint that two variables be equal (e.g. $A_i = I$ or $A_i = -I$). Mathematically, there is little difference between the separable and non-separable forms; but computationally the separable form allows for direct application

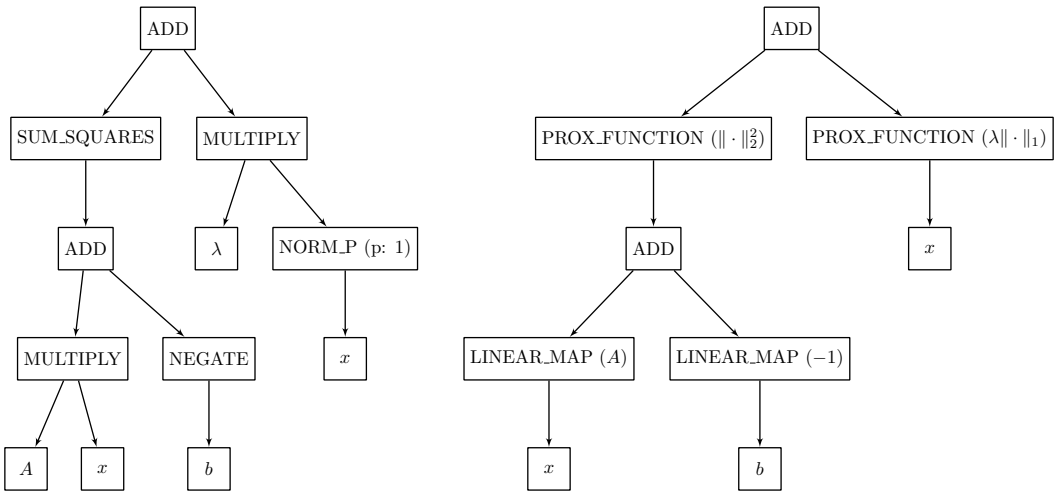

Figure 3.2: Original abstract syntax tree for lasso $\|Ax - b\|_2^2 + \lambda\|x\|_1$ (left) and the same expression converted to prox-affine form (right).

of the ADMM-based operator splitting algorithm. Specifically, the separable prox-affine form maps directly to a sequence of proximal and linear operator evaluations employed by the Epsilon variant of ADMM.

The algorithm we present shortly for problems in separable prox-affine form will ultimately reduce to solving generalized proximal operators of the form

$$\text{prox}_{f\circ H,A}(v) = \underset{x}{\text{argmin}}\, \lambda f(H(x)) + (1/2)\|A(x) - v\|_2^2. \tag{3.5}$$

For most functions f and affine transformations H and A, this function will not have a simple closed-form solution, even if a simple proximal operator exists for f. The three important exceptions to this are when: 1) f is the null function, 2) f is the indicator of the zero cone, or 3) f is a sum-of-squares function; in all these cases, the solution boils down to a linear least squares problem. However, for certain linear operators (namely, scalar or diagonal transformations), there *are* many cases where the generalized proximal operator has a straightforward solution. The Epsilon compiler produces a prox-affine form where the combination of f, H, and A results in an efficient proximal operator.

3.1.2 Conversion to prox-affine form

The first stage of the Epsilon compiler transforms an arbitrary disciplined convex problem into a prox-affine problem. Concretely, given an AST representing the optimization problem in its original form, which may consist of any valid composition of functions from the DCP library, this stage produces an AST with a reduced set of nodes:

- `ADD`. The sum of its children $x_1 + \cdots + x_n$.
- `PROX_FUNCTION`. A prox-friendly function with proximal operator implementation in the Epsilon solver.

- `LINEAR_MAP`. A linear function with linear operator implementation in the Epsilon solver.
- `VARIABLE`, `CONSTANT`. Each leaf of the AST is either a variable or constant.

In addition, once the AST is transformed into prox-affine form, each `PROX_FUNCTION` corresponds to a function with an efficient proximal operator implementation from the library described in Section 3.2. An example of this transformation is shown in Figure 3.2.

The transformation to prox-affine form is done in two passes over the AST representing the optimization problem. In the first pass, we convert all nodes representing linear functions to a nodes of type `LINEAR_MAP` which map directly to the linear operators available in the Epsilon solver. In terms of ASTs representing linear functions, there are two possibilities in any DCP-valid input problem: 1) direct linear transformations of inputs, such as the unary `SUM` or variable argument `HSTACK` and 2) binary functions such as `MULTIPLY` which apply a linear operator defined by a constant expression. The former have straightforward transformations to `LINEAR_MAP` representations; for the latter, DCP rules require that one of the arguments be constant and thus this argument is evaluated and converted to a linear operator (typically, a sparse, dense or diagonal matrix) and a `LINEAR_MAP` node with single argument.

In the second pass, we complete the transformation to prox-affine form by applying a set of prioritized rules, preferring to map ASTs onto a high-level proximal operator implementation when available but falling back to conic transformations when necessary. In short, given an input tree (or subtree) along with a set of rules for proximal operator transformations, we match the input against these rules. If a matching proximal operator is found, we then transform the function arguments (represented by subtrees of the original input tree) so that they have valid form for composition with the proximal operator in question. In doing so, the process may introduce auxiliary variables and additional indicator functions; the behavior of `ConvertProxArguments` depends on the requirements of the particular proximal operator, but some common examples include:

- *No-op.* The expression $\max\{-x, 0\}$ with variable x is the hinge function $f(x) = \max\{x, 0\}$, composed with the linear transformation $-I$. This represents a valid proximal operator so no further transformations are necessary and the argument $-x$ is returned as-is.
- *Epigraph transformation.* The expression $\|Ax - b\|_1$ with constants A, b and variable x matches a the proximal operator for the ℓ_1-norm $\|\cdot\|_1$, but it cannot be composed with an arbitrary affine function given the set of proximal operators available in the Epsilon solver. Therefore, a new variable y is introduced along with the constraint $I_0(Ax - b - y)$; y is returned as argument, resulting in the new expression $\|y\|_1$.
- *Kronecker product splitting.* The expression $\|AXB - C\|_F^2$ with constants A, B, C and variable X matches the proximal operator for sum-of-squares, but evaluation would require factoring $F^TF + I$ where $F = B^T \otimes A$ which cannot be done efficiently given the linear operators available. Therefore, the compiler introduces a new variable Z, modifies the argument to be $\|AZ - C\|_F^2$ and introduces the constraint $I_0(XB - Z)$.

Once the arguments have been transformed to the proper form, we create a `PROX_FUNCTION` node (with attribute specifying *which* proximal operator implementation) and the transformed arguments as children. Any indicators that were added by the argument conversion process are themselves recursively converted to prox-affine form and the result is accumulated in the output

under an `ADD` node.

3.1.3 Optimization and separation of prox-affine form

Once the problem has been put in prox-affine form, the next stage of the compiler transforms it to be separable in preparation for the solver. Although we previously described the prox-affine form generically, where we were minimizing over a single variable x and each term could potentially depend on all variable $f_i(H_i(x))$, the reality is that for many problems the x variables are already "naturally" partitioned to some extent (for instance, this arises in epigraph transformations, where one function in the prox-affine form will only depend on epigraph variables). Thus, to be more concrete, we introduce a partitioning of the variables $x = (x_1, \ldots, x_k)$ (where here each x_j is itself a vector of appropriate size, and let $\mathcal{J}_i$ denote the set of all variables that are used in the ith prox operator, i.e. our optimization problem becomes

$$\underset{x_1,\ldots,x_k}{\text{minimize}} \sum_{i=1}^{n} f_i(H_i(x_{\mathcal{J}_i})). \tag{3.6}$$

The process of separation is effectively one of introducing "copies" of variables until we reach a point that each objective term f_i has a unique set of variables, and the interactions between variables are captured entirely by the explicit equality constraints.

Given the form above, we describe the optimization problem via a bipartite graph, between nodes corresponding to objective functions $f_1, \ldots, f_N$ (plus additional equality constraints, in the final form) and nodes corresponding to variables $x_1, \ldots, x_k$. An edge exists between f_i and x_j if $j \in \mathcal{J}_i$, i.e. if the function uses that variable. By applying a sequence of transformations, we will introduce new variables and new equality constraints that will put the problem into a separable prox-affine form.

Definition of equivalence transformations. Specifically, the compiler sequentially executes a series of transformations to put the problem in separable prox-affine form:

1. *Move equality indicators.* The first compiler stage ("Conversion to prox-affine form", see Section 3.1.2) produces a single expression for the objective which includes all constraints via indicators; due to the nature of the transformations, many equality constraints are "simple" (e.g. involving I or $-I$) and can thus be moved to actual constraints in the separable form. This pass performs these modifications based on the linear map associated with the edges corresponding to variables in each equality constraint, splitting expressions when necessary. For example, an objective term $I_0(Ax + y + z)$ is transformed to a new objective term $I_0(Ax - w)$ and the constraint $w + y + z = 0$.
2. *Combine objective terms.* The basic properties of proximal operators (see e.g. [97]) allow simple functions like $c^T x$ and $\|x - b\|_2^2$ to be combined with other terms, reducing the number of proximal operators needed in the separable prox-affine form. This pass combines these terms assuming there is another objective term which includes the same variable.
3. *Add variable copies and consensus constraints.* The final pass guarantees that the objective is separable by introducing variable copies and consensus constraints. For example, the objective $f(x) + g(x)$ is transformed to $f(x_1) + g(x_2)$ and the constraint $x_1 = x_2$ is added.

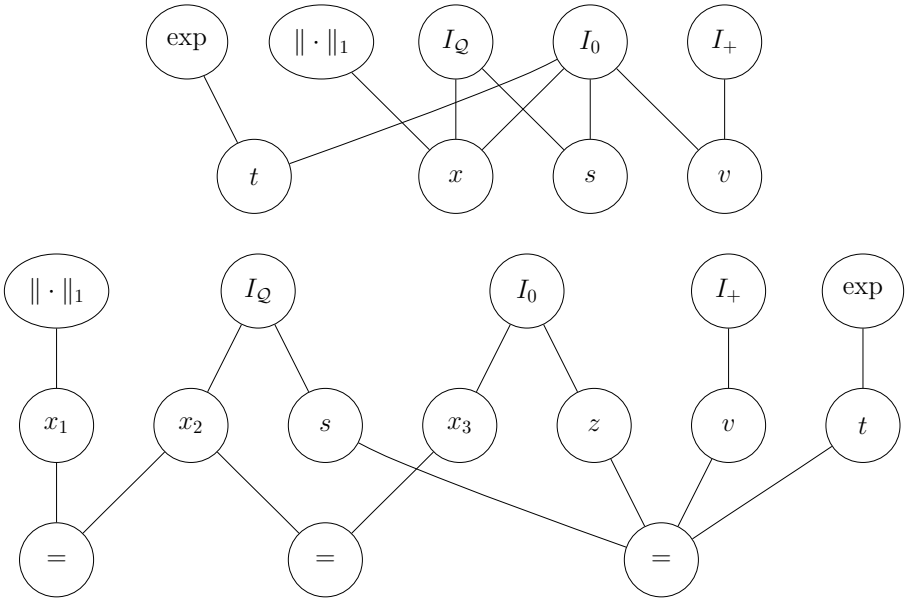

Figure 3.3: Compiler representation of prox-affine form as bipartite graph for $\exp(\|x\|_2 + c^T x) + \|x\|_1$ after conversion pass (top, see equation 3.8) and after separation pass (bottom, see equation 3.9).

For illustration purposes, consider the problem

$$\text{minimize} \quad \exp(\|x\|_2 + c^T x) + \|x\|_1. \tag{3.7}$$

The first compiler stage converts this problem to prox-affine form by introducing auxiliary variables t, s, v, along with three cone constraints and two prox-friendly functions:

$$\text{minimize} \quad \exp(t) + \|x\|_1 + I_{\mathcal{Q}}(x, s) + I_0(s + c^T x - t - v) + I_+(v), \tag{3.8}$$

where $I_{\mathcal{Q}}$ denotes the indicator of the second-order cone, I_0 the zero cone and I_+ the nonnegative orthant. The problem in this form is the input for the second stage which constructs the bipartite graph shown in Figure 3.3 (top). The problem is then transformed to have separable objective with many of the terms in I_0 (those with simple linear maps) move to the constraint

$$\begin{array}{ll} \text{minimize} & \exp(t) + \|x_1\|_1 + I_{\mathcal{Q}}(x_2, s) + I_0(c^T x_3 - z) + I_+(v) \\ \text{subject to} & s + z - t - v = 0 \\ & x_1 = x_2 \\ & x_2 = x_3 \end{array} \tag{3.9}$$

and variables x_1, x_2, x_3 are introduced along with consensus constraints. The bipartite graph for the final output from the compiler, a problem in separable prox-affine form, is shown in Figure 3.3 (bottom).

3.1.4 Solving problems in prox-affine form

Once the problem has been put in separable prox-affine form, the Epsilon solver applies the ADMM-based operator splitting algorithm using the library of proximal and linear operators. The implementation details of each operator are abstracted from the high-level algorithm, which applies the operators only through a common interface providing the basic mathematical operations required. Next we give a mathematical description of the operator splitting algorithm itself while the computational details of individual proximal and linear operators are discussed in Section 3.2.

Given a problem in separable prox-affine form

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} f_i(H_i(x_i)) \\ \text{subject to} \quad & \sum_{i=1}^{N} A_i(x_i) = 0. \end{aligned} \tag{3.10}$$

This operator splitting algorithm is motivated by considering the augmented Lagrangian

$$L_\lambda(x_1, \ldots, x_N, y) = \sum_{i=1}^{N} f_i(H(x_i)) + y^T(Ax - b) + (1/2\lambda)\|Ax - b\|_2^2 \tag{3.11}$$

where y is the dual variable, $\lambda \geq 0$ is the augmented Lagrangian penalization parameter, and $Ax = \sum_{i=1}^{N} A_i(x_i)$. The ADMM method applied here results in the Gauss-Seidel updates[1] with

$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \lambda f_i(H_i(x_i)) + \frac{1}{2}\left\| \sum_{j<i} A_j(x_j^{k+1}) + A_i(x_i) + \sum_{j>i} A_j(x_j^k) - b + u^k \right\|_2^2 \\ u^{k+1} &:= u^k + Ax^{k+1} - b \end{aligned} \tag{3.12}$$

where we have $u = \lambda y$ is the scaled dual variable. Critically, the x_i-updates are applied using the (generalized) proximal operator: let

$$v_i^k = b - u^k - \sum_{j<i} A_j(x_j^{k+1}) - \sum_{j>i} A_j(x_j^k) \tag{3.13}$$

then we have

$$x_i^{k+1} := \operatorname*{argmin}_{x_i} \lambda f_i(H_i(x_i)) + (1/2)\|A(x_i) - v_i^k\|_2^2 = \text{prox}_{\lambda f_i \circ H_i, A_i}(v_i^k) \tag{3.14}$$

The ability of the solver to evaluate the generalized proximal operator efficiently will depend on f_i and A_i (in the most common case $A_i^T A_i = \alpha I$, a scalar matrix, which can be handled by any proximal operator); it is the responsibility of the compiler to ensure that the prox-affine problem has been put in the required form such that these evaluations map to efficient implementations from the proximal operator library.

[1] specifically, the update for $x_i^{(k+1)}$ depends on $x_j^{(k+1)}$ for $j < i$ and $x_j^{(k)}$ for $j > i$

3.2 Fast computational operators

The proximal operator library directly implements the atoms available to disciplined convex programming frameworks, reducing the need for extensive transformation before solving an optimization problem. As the evaluation of proximal operators as well as the operations required by high-level algorithms rely heavily on linear operators, Epsilon also provides a library of efficient linear operators (and a system for composing them), extending beyond the standard dense/sparse matrices typically found in generic convex solvers.

3.2.1 Linear operators

In general, the computation required for solving convex optimization problems often depends heavily on the application of linear operators. Most commonly these linear operators are implemented with sparse or dense matrices which explicitly represent the coefficients of the linear transformation. Clearly, in many cases this can be inefficient (see, e.g., [33] and the references therein) and as such, we abstract the notion of a linear operator allowing for other implementations which can often be far more efficient than direct matrix representation.

A motivating example that arises in many applications is the use of matrix-valued variables. As intermediate representations for convex programming (both prox-affine and conic forms) typically reduce optimization problems over matrices to ones over vectors, matrix products naturally give rise to the Kronecker product. For example, consider the expression AX where $A \in \mathbb{R}^{m \times n}$ is a dense constant and $X \in \mathbb{R}^{n \times k}$ is the optimization variable; we vectorize this product with $\operatorname{vec}(AX) = (I \otimes A)\operatorname{vec}(X)$ where $\otimes$ denotes the Kronecker product. Representing the Kronecker product as a sparse matrix is not only space inefficient (i.e. requiring A to be repeated k times) but can also be extremely costly to factor. In particular, the naive approach requires a sparse factorization of a $km \times kn$ matrix as opposed to factoring a dense $m \times n$ matrix A directly. Explicitly maintaining the Kronecker product structure provides a mechanism for avoiding this unnecessary computational cost.

Epsilon augments the standard sparse/dense matrices with dedicated implementations for diagonal matrices, scalar matrices, the Kronecker product as well as composite types representing a sum or product:

- *Dense matrix*. A dense matrix $A \in \mathbb{R}^{m \times n}$ with $O(mn)$ storage.
- *Sparse matrix*. A sparse matrix $A \in \mathbb{R}^{m \times n}$ with O(# nonzeros) storage.
- *Diagonal matrix*. A diagonal matrix $A \in \mathbb{R}^{n \times n}$ with $O(n)$ storage.
- *Scalar matrix*. A scalar matrix $\alpha I \in \mathbb{R}^{n \times n}$ with $\alpha \in \mathbb{R}$ and $O(1)$ storage.
- *Kronecker product*. The Kronecker product of $A \in \mathbb{R}^{m \times n}$ and $BS \in \mathbb{R}^{p \times q}$, representing a linear map $\mathbb{R}^{nq} \to \mathbb{R}^{mp}$ where A and B can themselves be any linear operator type.
- *Sum*. The sum of linear operators $A_1 + \cdots + A_N$.
- *Product*. The product of linear operators $A_1 \cdots A_N$.
- *Abstract*. An abstract linear operator which cannot be combined with any of the basic types. This is used to represent factorizations, see below.

Each linear operator type A supports the following operations:

- *Apply*. Given a vector x, return $y = Ax$.
- *Transpose*. Return the linear operator A^T.
- *Inverse*. Return the linear operator A^{-1}.

The transpose operation returns a linear operator of the same type whereas the inverse operation may return a linear operator of a different type. The inverse operation is undefined for non-invertible linear operators and in practice is intended to be used in contexts where the linear transformation is known to be invertible.

In addition, linear operators also support binary operations for sum $A + B$ and product AB. This requires a system for type conversion, the basic rules for which are described with an ordering of the types corresponding to their sparsity ($Dense > Sparse > Diagonal > Scalar$); in order to combine any two of these types, we first promote the sparser type to to the denser type. For Kronecker products, type conversion depends on the arguments: the sum of two Kronecker products can be combined if one of the arguments is equivalent, e.g.

$$A \otimes B + A \otimes C = A \otimes (B + C) \tag{3.15}$$

while the product can be combined if the arguments have matching dimensions, i.e.

$$(A \otimes B)(C \otimes D) = AC \otimes BD \tag{3.16}$$

if we can form AC and BD. In either case, if a combination is possible it will be performed along with the appropriate type conversion for the sum/product of the arguments themselves. If a combination is not possible according to these rules, the resulting type will instead be a Sum or $Product$ composed of the arguments.

3.2.2 Proximal operators

The second class of operators that form the basis for Epsilon are the proximal operators. As seen in the resulting description of the ADMM algorithm, each individual solution over the x_i variables can be represented via an operator

$$x_i^{k+1} := \underset{x_i}{\operatorname{argmin}}\, \lambda f(H(x_i)) + \frac{1}{2}\|A_i(x_i) - v_i^k\|_2^2 \tag{3.17}$$

for some value of v_i^k (which naturally depends generally on the dual variables for the equality constraints involving x_i as well as the other x_j variables). This is exactly the generalized proximal operator $x_i^{k+1} = \operatorname{prox}_{\lambda f \circ H_i, A_i}(v)$. Indeed, the Epsilon compiler uses precisely the set of available fast proximal operators to reduce the convex optimization problems to fast forms relative to the corresponding cone problem; while any of the problems can be solved by reducing everything to conic form (and thus using only proximal operators corresponding to cone projections), the speed of the solver crucially depends on the ability to evaluate a much wider range of these proximal operators efficiently.

It is well-known that many proximal operators have closed form solutions that can be solved much more quickly than general optimization problems (see e.g. [97] for a review). In this section, we highlight several of the operators included in Epsilon along with a general description

of the methods used to solve them. The proximal operators group roughly into three categories: elementwise, vector, and matrix operators, for functions f of scalar inputs, vector inputs, and matrix inputs respectively. Note that this is not a perfect division, because several vector or matrix functions are simply sums of corresponding scalar functions, e.g., $\|x\|_2^2 = \sum_{i=1}^n x_i^2$, but instead we use vector or matrix designations to refer to functions that *cannot* be decomposed over their inputs.

- **Exact linear, quadratic, or cubic equations.** Several prox operators can be solved using exact solutions to their gradient conditions. For instance, the prox operator of $f(x) = (ax - b)^2$ is given by the solution to the gradient equation $2a(ax - b) + x = 0$, which is a simple linear equation. Similarly, the prox operators for $f(x) = -\log x$ and $f(x) = 1/x$ have gradient conditions that are given by quadratic and cubic equations respectively. Here some care must be taken to ensure that we select the correct one of two or three possible solutions, but this can be ensured analytically in many cases or simply by checking all solutions in the worst case. For vector functions, the proximal operator of a least-squares objective is also an instance of this case, though here of course the complexity requires a matrix inversion rather than a scalar linear equation.
- **Soft thresholding.** The absolute value and related functions can be solved via the soft thresholding operator. For example, the proximal operator of $f(x) = |x|$ is given by

$$\text{prox}_{\lambda f}(v) = \begin{cases} v - \lambda & \text{if } v > \lambda \\ 0 & \text{if } -\lambda \leq v \leq \lambda \\ v + \lambda & \text{if } v < -\lambda. \end{cases} \tag{3.18}$$

- **Newton's method.** Several proximal operators for smooth functions have no easily computed closed form solution. Nonetheless, in this case Newton's method can be used to find a solution in reasonable time. For elementwise functions, for instance, these operations are relatively efficient, because in practice a very small number of Newton iterations are needed to reach numerical precision. For example, the proximal operator of the logistic function $f(x) = \log(1 + \exp(x))$ has no closed form solution, but can easily computed with Newton's method.
- **Projection approaches.** The proximal operator for several functions can be related to the projection on to a set. In the simplest case, the proximal operator for an indicator of a set is simply equal to the projection onto the set, giving prox operators for cone projections. However, additional proximal methods derive from Moreau decomposition [84], which states that

$$v = \text{prox}_f(v) + \text{prox}_{f^*}(v) \tag{3.19}$$

  where f^* denotes the Fenchel conjugate of f. For example, the proximal operator for the ℓ_∞-norm, $f(x) = \|x\|_\infty$ is given by

$$\text{prox}_f(v) = v - \mathcal{P}_{\|x\|_1 \leq 1}(v) \tag{3.20}$$

  using the relation that the Fenchel conjugate of a normal is equal to the indicator of of the dual norm ball. The projection on to the ℓ_1-ball can be accomplished in $O(n)$ time using a median of medians algorithm [38].

- **Orthogonally invariant matrix functions.** If a matrix input function is defined solely by the singular values (or eigenvalues) of a matrix, then the proximal operator can be computed using the singular value decomposition (or eigenvalue decomposition) of that matrix. Typically, the running time of these proximal operators are dominated by the cost of the decomposition itself, making them very efficient for reasonably-sized matrices. For example, the $\log\det$ function can be written as

$$\log\det(X) = \sum_{i=1}^{n} \log \lambda_i \tag{3.21}$$

so its proximal operator can be given by

$$\operatorname{prox}_{\log\det}(X) = U \operatorname{prox}_{\log}(\Lambda) U^T \tag{3.22}$$

where $\operatorname{prox}_{\log}(\Lambda)$ is shorthand for applying the proximal operator for the negative log function to each diagonal element of the eigenvalues Λ. The prox operator for the $\log$ function can itself be solved via a quadratic equation, so computing the inner prox term is only an $O(n)$ operation and the runtime is dominated by the $O(n^3)$ cost of computing the eigenvalue decomposition.

- **Special purpose algorithms.** Finally, though we cannot enumerate these broadly, several proximal operators have special purpose fast solvers for these particular types of operators. A particularly relevant example is the fused lasso $f(x) = \|Dx\|_1$ where D is the first order difference operator; this proximal operator can be solved efficiently via an $O(n)$ dynamic programming approach [65].

Function			Proximal operator	
Category	**Atom**	**Definition**	**Method**	**Complexity**
Cone	Zero	$f(x) = I_0(Ax - b),\ A \in \mathbb{R}^{m\times n}$	subspace projection	$O(mn^2 + n^3)$
	Nonnegative orthant	$f(x) = I_+(x)$	positive thresholding	$O(n)$
	Second-order cone	$f(x,t) = I_{\mathcal{Q}}(x,t),\ x \in \mathbb{R}^n,\ t \in \mathbb{R}$	projection	$O(n)$
	Semidefinite cone	$f(X) = I_{\succeq}(X),\ X \in \mathbb{S}^n$	positive thresholding on $\lambda(X)$	$O(n^3)$
Elementwise $x, y \in \mathbb{R}$	Absolute	$f(x) = \lvert x\rvert$	soft thresholding	$O(n)$
	Square	$f(x) = x^2$	linear equation	$O(n)$
	Hinge	$f(x) = \max\{x, 0\}$	soft thresholding	$O(n)$
	Deadzone	$f(x) = \max\{\lvert x\rvert - \epsilon, 0\},\ \epsilon \geq 0$	soft thresholding	$O(n)$
	Quantile	$f(x) = \max\{\alpha x, (\alpha - 1)x\},\ 0 \leq \alpha \leq 1$	asymmetric soft thresholding	$O(n)$
	Logistic	$f(x) = \log(1 + \exp(x))$	Newton	$O(n) \cdot$ (# Newton)
	Inverse positive	$f(x) = 1/x,\ x \geq 0$	Newton	$O(n) \cdot$ (# Newton)
	Negative log	$f(x) = -\log(x),\ x \geq 0$	quadratic equation	$O(n)$
	Exponential	$f(x) = \exp(x)$	Newton	$O(n) \cdot$ (# Newton)
	Negative entropy	$f(x) = x \cdot \log(x),\ x \geq 0$	Newton	$O(n) \cdot$ (# Newton)
	KL Divergence	$f(x,y) = x \cdot \log(x/y),\ x, y \geq 0$	Newton	$O(n) \cdot$ (# Newton)
	Quadratic over linear	$f(x,y) = x^2/y,\ y \geq 0$	cubic equation	$O(n)$
Vector $x \in \mathbb{R}^n$	ℓ_1-norm	$f(x) = \lVert x\rVert_1$	soft thresholding	$O(n)$
	Sum-of-squares	$f(x) = \lVert Ax - b\rVert_2^2,\ A \in \mathbb{R}^{m\times n}$	normal equations	$O(mn^2 + n^3)^\dagger$
	ℓ_2-norm	$f(x) = \lVert x\rVert_2$	group soft thresholding	$O(n)$
	ℓ_∞-norm	$f(x) = \lVert x\rVert_\infty$	median of medians	$O(n)$
	Maximum	$f(x) = \max_i x_i$	median of medians	$O(n)$
	Log-sum-exp	$f(x) = \log\left(\sum_{i=1}^n \exp(x_i)\right)$	Newton	$O(n) \cdot$ (# Newton)
	Fused lasso	$f(x) = \sum_{i=1}^{n-1} \lvert x_i - x_{i+1}\rvert$	dynamic programming [65]	$O(n)$
Matrix	Negative log det	$f(X) = -\log\det(X), X \in \mathbb{S}^n$	quadratic equation on $\lambda(X)$	$O(n^3)$
	Nuclear norm	$f(X) = \lVert\sigma(X)\rVert_1, X \in \mathbb{R}^{m\times n}$	soft thresholding on $\sigma(X)$	$O(n^3)$
	Spectral norm	$f(X) = \lVert\sigma(X)\rVert_\infty, X \in \mathbb{R}^{m\times n}$	median of medians on $\sigma(X)$	$O(n^3)$

†Or $O(nm^2 + m^3)$ if $m < n$, and often lower if A is sparse. Furthermore, the cost can be amortized over multiple evaluations for the same A matrix, we can compute a Cholesky factorization once in this time, and solve subsequent iterations in $O(mn + n^2)$ time.

3.3 Examples and numerical results

In this section we present several examples of convex problems from statistical machine learning and compare Epsilon to existing approaches on a library of examples from several domains. At present, Python integration is provided (Matlab, R, Julia versions are planned) with CVXPY [34] providing the environment for constructing the disciplined convex programs and performing DCP validation. Epsilon effectively serves as a solver for CVXPY although behind the scenes the CVXPY/Epsilon interface is somewhat different than for other solvers as Epsilon compiler implements its own transformations on the AST. Nonetheless, from a user perspective problems are specified in the same syntax, a high-level domain specific language of which we give several examples in this section. Epsilon is open source and available at `http://epopt.io/`, including the code for all examples and benchmarks discussed here.

As Epsilon integrates with CVXPY, we make the natural comparison between Epsilon and the existing solvers for CVXPY which use the conic form. In particular, we compare Epsilon to ECOS [35], an interior point method and SCS [91], the "splitting conic solver". In general, interior point methods achieve highly accurate solutions but have trouble scaling to larger problems and so it is unsurprising that Epsilon is able to solve problems to moderate accuracy several orders of magnitude faster than ECOS (this is also the case when comparing ECOS and SCS, see [91]). On the other hand, SCS employs an operator splitting method that is similar in spirit to the Epsilon solver, both being variants of ADMM; the main difference between the two is in the intermediate representation and set of available proximal operators: SCS solves the conic form produced by CVXPY with cone and subspace projections (using sparse matrices) while Epsilon solves the prox-affine form using the much larger library of fast proximal and linear operators described in Section 3.2. In practice, this has significant impact with Epsilon achieving the same level of accuracy as SCS an order of magnitude faster (or more) on many problems.

In what follows we examine several examples in detail followed by results on a library of 20+ problems common to applications in statistical machine learning and other domains. In the detailed examples, we start with a complete specification of the input problem (a few lines of Python in the CVXPY grammar), discuss the transformation by the Epsilon compiler to an abstract syntax tree representing a prox-affine problem and explore how the Epsilon solver scales relative to conic solvers. When printing the AST for individual problems, we adopt a concise functional form which is a serialized version of the abstract syntax trees shown in Figure 3.2 and is the internal representation of the Epsilon compiler, as well as input to the Epsilon solver.

3.3.1 Lasso

We start with the lasso problem

$$\underset{\theta}{\text{minimize}} \;\; (1/2)\|X\theta - y\|_2^2 + \lambda\|\theta\|_1, \tag{3.23}$$

where $X \in \mathbb{R}^{m\times n}$ contains input features, $y \in \mathbb{R}^m$ the outputs, and $\theta \in \mathbb{R}^n$ are the model parameters. The regularization parameter $\lambda \geq 0$ controls the tradeoff between data fit and sparsity in the solution—the lasso is especially useful in the high-dimensional case where $m \leq n$ as sparsity effectively controls the number of free parameters in the model, see [123] for details.

In CVXPY, the lasso can be written as

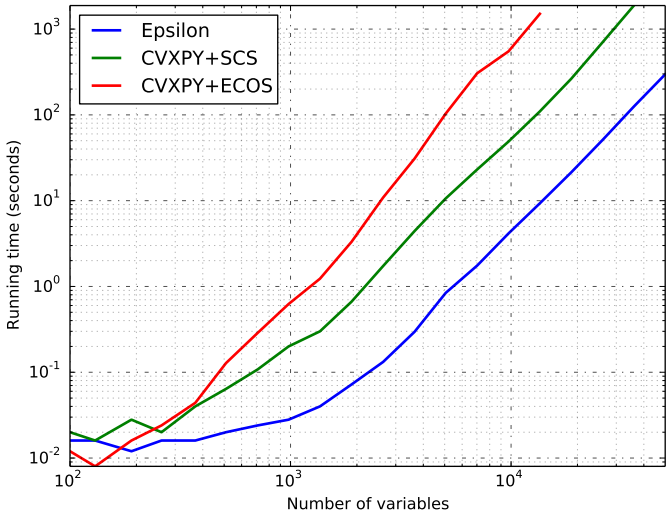

Figure 3.4: Comparison of running times on lasso example with dense $X \in \mathbb{R}^{m \times 10m}$.

```
theta = Variable(n)
f = sum_squares(X*theta - y) + lam*norm1(theta)
prob = Problem(Minimize(f))
```

where `sum_squares()`/`norm1()` functions correspond directly to the two objective terms. In essence this problem is already in prox-affine form with proximal operators for $\|X\theta - y\|_2^2$ and $\|\theta\|_1$; thus the prox-affine AST produced by the Epsilon compiler merely adds an additional variable and equality constraint to make the objective separable:

```
objective:
  add(
    sum_squares(add(const(a), scalar(-1.00)*dense(B)*var(x))),
    norm1(var(y)))

constraints:
  zero(add(var(y), scalar(-1.00)*var(x)))
```

Note that in the automatically generated serialization of the AST, variable/constant names are auto-generated and do not necessarily correspond to user input. In this particular example `a`, `B` correspond to the constants y, X from the original problem while `x`, `y` correspond to two copies of the optimization variable θ.

Computationally, it is the evaluation of the sum-of-squares proximal operator which dominates the running time for Epsilon as it requires solving the normal equations. However, the cost of the factorization required is amortized as this step can be performed once before the first iteration of the algorithm, as discussed in Section 3.2. In Figure 3.4, we compare the running time of Epsilon to CVXPY+SCS/ECOS on a sequence of problems with dense $X \in \mathbb{R}^{m \times 10m}$. Epsilon

(representing X as a dense linear operator) performs a dense factorization while SCS embeds X in a large sparse matrix and performs a sparse factorization (as it does with all problems). The difference in these factorizations explains the difference in running time as the time spent performing the actual iterations is negligible for both methods.

3.3.2 Multivariate lasso

In this example, we apply lasso to the multivariate regression setting where the output variable is now a vector as opposed to a scalar in univariate regression. In particular,

$$\underset{\Theta}{\text{minimize}} \;\; (1/2)\|X\Theta - Y\|_F^2 + \lambda\|\Theta\|_1 \tag{3.24}$$

where $X \in \mathbb{R}^{m\times n}$ are input features, $Y \in \mathbb{R}^{m\times k}$ represent the k-dimensional response variable and the optimization variable is now a matrix $\Theta \in \mathbb{R}^{n\times k}$, representing the parameters of the multivariate regression model. The Frobenius norm $\|\cdot\|_F$ is the ℓ_2-norm applied elementwise to a matrix and here $\|\cdot\|_1$ is also interpreted elementwise.

The CVXPY problem specification for the multivariate lasso is virtually identical to the standard lasso example

```
Theta = Variable(n,k)
f = sum_squares(X*Theta - Y) + lam*norm1(Theta)
prob = Problem(Minimize(f))
```

with the only change being the replacement of vectors `y`, `theta` with their matrix counterparts `Y`, `Theta` (by convention, we denote matrix-valued variables with capital letters). As a result, when the Epsilon compiler transforms this problem to the prox-affine AST,

```
objective:
  add(
    sum_squares(add(kron(scalar(1.00), dense(A))*var(X),
       scalar(-1.00)*const(B))),
    norm1(var(Y)))

constraints:
  zero(add(var(Y), scalar(-1.00)*var(X)))
```

the matrix-valued optimization variable results in the `kron` linear operator appearing as argument to the `sum_squares` proximal operator. This corresponds to the specialized Kronecker product linear operator implementation with (in this case) $O(mn)$ for the dense data matrix X.

Although it is a simple to change to the problem specification to apply Lasso in the multivariate case, the new problem results in a substantially different computational running times for the solvers considered. In Figure 3.5 we show the running times of each approach on a sequence of problems with $X \in \mathbb{R}^{m\times 10m}$ and $Y \in \mathbb{R}^{m\times 10}$; where as on standard Lasso Epsilon was roughly 10x faster than SCS, now the gap is closer to 100x, e.g. for a problem with 1.35×10^4 variables, SCS requires 2192 seconds vs. 27 seconds for Epsilon. The reason for this difference is due to

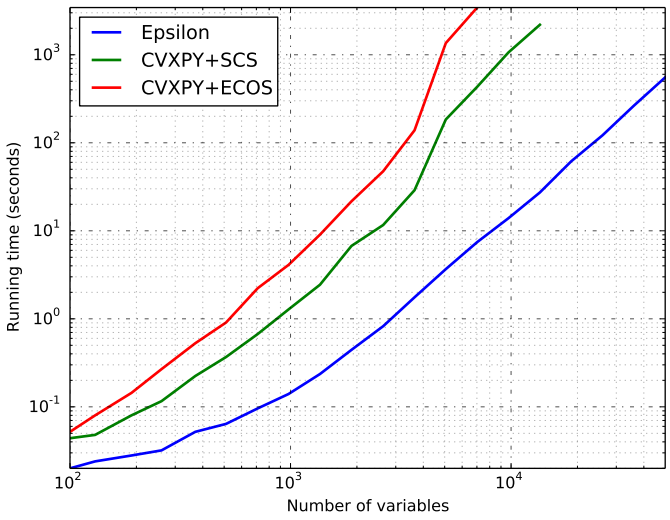

Figure 3.5: Comparison of running times on the multivariate Lasso example with dense $X \in \mathbb{R}^{m\times 10m}$ and $k = 10$.

the representation of the linear operator required for solving the normal equations for the least squares term

$$(1/2)\|(I_k \otimes X)\,\mathrm{vec}(\Theta) - \mathrm{vec}(Y)\|_2^2. \tag{3.25}$$

Since SCS is restricted to representing linear operators as sparse matrices, it must instantiate the Kronecker product explicitly (replicating the X matrix 10 times) and factor the resulting matrix with sparse methods. In contrast, Epsilon represents the Kronecker product directly (using the `kron` linear operator) and applies dense factorization methods without any unnecessary replication.

3.3.3 Total variation

In the previous lasso examples, we employ the ℓ_1-norm to estimate a sparse set of regression coefficients—a natural extension to this idea is to incorporate a notion of structured sparsity. The fused lasso problem (originally proposed in [124])

$$\underset{\theta}{\text{minimize}} \;\; (1/2)\|X\theta - y\|_2^2 + \lambda_1\|\theta\|_1 + \lambda_2 \sum_{i=1}^{n-1} \|\theta_{i+1} - \theta_i\|_1 \tag{3.26}$$

employs total variation regularization (originally proposed in [108]) to encourage the *differences* of the coefficient vector $\theta_{i+1} - \theta_i$ to be sparse. Such structure naturally arises in problems where the dimensions of the coefficient vector correspond to vertices in a chain or grid, see e.g. [3, 143] for example applications.

For total variation problems, CVXPY provides a function `tv()` which makes the problem specification concise:

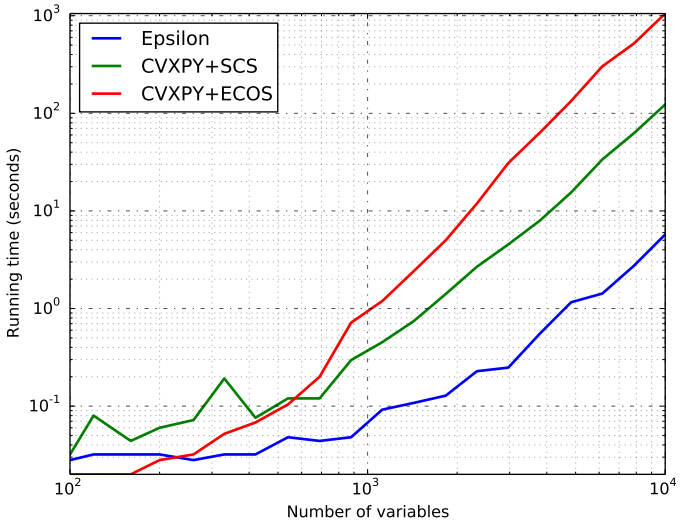

Figure 3.6: Comparison of running times on total variation example with $X \in \mathbb{R}^{m\times 10m}$, $y = X\theta_0 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.05^2)$ where θ_0 is piecewise constant with segments of length 10.

```
theta = Variable(n)
f = sum_squares(X*theta - y) + lam1*norm1(theta) + lam2*tv(
   theta)
prob = Problem(Minimize(f)).
```

In conic form this penalty is transformed to a set of a linear constraints which involve the first order differencing operator (to be defined shortly); however, Epsilon includes a direct proximal operator implementation of the total variation penalty and thus the compiler simply maps the problem specification onto three proximal operators

```
objective:
  add(
    least_squares(add(dense(C)*var(x), scalar(-1.00)*const(d))
       ),
    norm1(var(y)),
    tv_1d(var(z)))

constraints:
  zero(add(var(z), scalar(-1.00)*var(y)))
  zero(add(var(z), scalar(-1.00)*var(x)))
```

with the addition of the necessary variable copies and equality constraints to make the objective separable.

Figure 3.6 compares Epsilon to the conic solvers on a sequence of problems with $X \in \mathbb{R}^{m\times 10m}$. The main difference between the two approaches is that while Epsilon directly solves

	Epsilon		CVXPY+SCS		CVXPY+ECOS	
Problem	Time	Objective	Time	Objective	Time	Objective
basis_pursuit	1.35s	1.44×10^2	17.26s	1.45×10^2	217.68s	1.45×10^2
covsel	0.93s	3.74×10^2	25.09s	3.73×10^2	-	-
fused_lasso	3.87s	7.46×10^1	57.85s	7.41×10^1	641.34s	7.41×10^1
hinge_l1	3.71s	1.15×10^3	45.59s	1.15×10^3	678.47s	1.15×10^3
hinge_l1_sparse	14.26s	1.38×10^3	106.75s	1.38×10^3	183.65s	1.38×10^3
hinge_l2	3.58s	3.87×10^3	133.10s	3.87×10^3	1708.31s	3.87×10^3
hinge_l2_sparse	1.82s	8.08×10^3	28.40s	8.09×10^3	47.72s	8.08×10^3
huber	0.20s	2.18×10^3	3.17s	2.18×10^3	28.43s	2.18×10^3
lasso	3.69s	3.21×10^1	20.54s	3.21×10^1	215.68s	3.21×10^1
lasso_sparse	13.58s	4.37×10^2	56.94s	4.37×10^2	277.80s	4.37×10^2
least_abs_dev	0.10s	7.09×10^3	2.96s	7.10×10^3	11.46s	7.09×10^3
logreg_l1	3.70s	8.18×10^2	51.60s	8.18×10^2	684.86s	8.17×10^2
logreg_l1_sparse	6.69s	9.61×10^2	33.35s	9.63×10^2	310.02s	9.61×10^2
lp	0.33s	7.77×10^2	3.78s	7.75×10^2	7.58s	7.77×10^2
mnist	0.91s	1.75×10^3	219.63s	1.72×10^3	1752.97s	1.72×10^3
mv_lasso	7.14s	4.87×10^2	824.83s	4.88×10^2	-	-
qp	1.39s	4.30×10^3	3.20s	4.28×10^3	23.12s	4.24×10^3
robust_pca	0.59s	1.71×10^3	2.88s	1.71×10^3	-	-
tv_1d	0.13s	2.29×10^5	51.85s	2.95×10^5	-	-

Table 3.1: Comparison of running time and objective value between Epsilon and CVXPY with SCS and ECOS solvers; a value of "-" indicates a lack of result due to either solver failure, unsupported problem or 1 hour timeout.

the proximal operator for the total variation penalty (using the dynamic programming algorithm of [65]), while transforming to conic form requires reformulating the fused lasso penalty as a linear program using auxiliary variables and involving the finite differencing operator $D \in \{-1, 0, 1\}^{(n-1)\times n}$

$$D = \begin{bmatrix} 1 & -1 & 0 & \cdots \\ 0 & 1 & -1 & \cdots \\ 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.27}$$

containing 1 on the diagonal, -1 on the first super diagonal and 0 elsewhere. For even moderate n this linear operator (which corresponds to the edge incidence matrix for the chain graph) is poorly conditioned which can be problematic for general solvers, see e.g. [104] for further details. The dedicated proximal operator avoids these issues, reducing running times—on a problem with 10^4 variables, Epsilon requires 5.7 seconds vs. 123 seconds for SCS.

Solver	Problem	Epsilon time	Solver time
liblinear	`hinge_l1`	3.71s	0.49s
	`hinge_l1_sparse`	14.26s	4.26s
	`hinge_l2`	3.58s	0.16s
	`hinge_l2_sparse`	1.82s	0.83s
glmnet	`lasso`	3.69s	0.84s
	`lasso_sparse`	13.58s	0.67s
	`logreg_l1`	3.70s	2.31s
	`logreg_l1_sparse`	6.69s	1.96s
	`mv_lasso`	7.14s	7.40s
Gurobi	`lp`	0.33s	6.02s
	`qp`	1.39s	4.12s
QUIC	`covsel`	0.93s	6.24s

Table 3.2: Comparison of running times between Epsilon and specialized solvers.

3.3.4 Library of convex programming examples

In this final section we present results on a library of example convex problems from statistical machine learning appearing frequently in the literature (e.g. [19, 91]). In general, each example depends on a variety of factors such as dimensions of the constants, data generation scheme and setting of the hyperparameters—we have chosen reasonable defaults for these settings. The complete specification for each problem is available in the the submodule `epsilon.problems`, distributed with the Epsilon code base. On each example, we run each solver considered using the default tolerances which for Epsilon and SCS correspond to moderate accuracy and high accuracy for ECOS[2]. In practice, we observe that all solvers converge to a relative accuracy of 10^{-2} which is reasonable for the statistical applications under consideration.

The running times in Table 3.1 show that on all problem examples considered, Epsilon is faster than SCS and ECOS and often by a wide margin. In general, we observe that Epsilon tends to solve problems in fewer ADMM iterations and for many problems the iterations are faster due in part to operating on a smaller number of variables. There are numerous reasons for these differences, some of which we have explored in the more detailed examples appearing earlier in this section.

3.3.5 Comparison with specialized solvers

Next, we compare Epsilon to an assortment of specialized solvers which are available for the most common problems benchmarked in the previous section. Before doing so, we emphasize that the general convex programming approach offers many advantages to specialized algorithms in terms of reuse and extensibility. In addition, many of the relatively simple convex problems from the previous section do not have dedicated, mature software packages readily available in common mathematical programming environments (e.g. Matlab, R, Python). Furthermore,

[2]Modifying the tolerances for an interior point method does not materially affect the comparison due to the nature of the bottlenecks.

even when specialized solvers are available, translating problems to the interface provided by a particular package requires effort to understand and conform to the idiosyncrasies of each implementation. In contrast, general convex programming offers a uniform syntax and interface allowing problems to be easily formulated, extended and solved.

In terms of running times, Table 3.2 compares Epsilon to four dedicated software packages implementing specialized algorithms: liblinear [44], glmnet [48], Gurobi [95] and QUIC [62]. As in Table 3.1, the default stopping criteria is used corresponding to moderate accuracy for Epsilon and high accuracy for the specialized solvers. For the most part, Epsilon is competitive, although on a few problems liblinear and glmnet are significantly faster. This is due to the ability of these specialized algorithms to exploit sparsity in the solution which arises due to ℓ_1 regularization (lasso problems) or a small number of support vectors in the dual SVM formulation (hinge problems). At present, Epsilon does not take advantage of such structure and thus may have a disadvantage on sparse problems. We consider ℓ-regularized problems in more detail in Part II, developing several additional specialized algorithms exploiting sparsity in the solution.

On the other hand, Table 3.2 shows that Epsilon is significantly faster than Gurobi and QUIC in solving linear/quadratic programs and sparse inverse covariance estimation (`covsel`), respectively. However, it is important to highlight that the specialized algorithms solve these problems to high accuracy (e.g. tolerances of 10^{-8} or smaller) while Epsilon targets only moderate accuracy (e.g. 10^{-3}). For moderate accuracy, the operator splitting approach can be highly competitive, allowing Epsilon to be significantly faster on some problems. In general, accuracy requirements are problem-specific—however, solving problems to moderate accuracy is often sufficient in statistics and machine learning.

Part II

Specialized Newton Methods for Sparse Problems

Introduction to Specialized Newton Methods for Sparse Problems

In Part I of this thesis we argue strongly for the development of general convex programming methods, presenting a system that combines the declarative syntax of disciplined convex programming with fast operator splitting algorithms. We believe that this paradigm of convex programming, which provides a separation between problem formulation and the implementation of numerical algorithms, is highly beneficial and will over time become the preferred method of developing and deploying convex methods for a wide variety of problems.

Nonetheless, we recognize that at present specialized algorithms may enjoy many benefits not yet available to general convex programming systems. From this perspective, we present several specialized Newton methods in Chapters 4-6, comprising Part II of this thesis. These methods are focused on sparse problems, exploiting sparsity at the algorithmic level through optimization techniques such as coordinate descent, allowing them to be orders of magnitude faster in the special case that the optimal value of the objective variable is sparse. In addition, Chapter 5 considers optimizing a control problem with nonconvex objective, a problem formulation which does not fit into the convex focus of the DCP framework. In this problem as well as all of the problems considered in Part II, the proposed algorithms advance the size of problems that can be solved by a significant amount, often multiple orders of magnitude.

In addition, there are several connections between the specialized methods developed in Part II and the general framework proposed in Part I. At a high level, these algorithms represent state-of-the-art specialized solvers for narrow classes of problems, representing the performance we would like our general convex methods to achieve. In some cases this is trivially possible: with the addition of ℓ_2 regularization, the specialized algorithms could be used directly as proximal operators in the operator splitting framework presented in Chapter 3. More deeply, we envision future general convex programming methods which automatically recognize problem properties (e.g. separable objective with smooth and nonsmooth parts) and use this information to tailor solution methods appropriately, possibly with the aid of additional tools such as automatic differentiation. Such a system would (in theory) achieve performance comparable to the specific algorithms presented here as well as many other state-of-the-art specialized implementations for other problems.

Chapter 4

The Sparse Gaussian Conditional Random Field

The sparse Gaussian conditional random field (CRF) is discriminative extension of sparse inverse covariance estimation [9] also known as the graphical lasso [47]. Sparse inverse covariance estimation enables convex learning of high-dimensional undirected graphical models with entries in the inverse covariance corresponding to edges in a Gaussian Markov random field. However, in many prediction tasks we may not want to model correlations between input variables. This is the familiar generative/discriminative contrast in machine learning [89], where it has been repeatedly observed that in terms of predictive accuracy, discriminative approaches can be superior [121]. In recent years several researchers have proposed the sparse Gaussian conditional random field with applications in many fields [117, 138, 146]. Our contribution, which we develop in this Chapter, is a specialized Newton method which we demonstrate to be several orders of magnitutde faster on problems of interest than previously proposed algorithms. In Chapter 7, we apply this model to forecasting supply and demand of energy in the electrical grid.

4.1 Problem formulation

Let $x \in \mathbb{R}^n$ denote covariates and and $y \in \mathbb{R}^p$ denote response variables for a prediction task. A Gaussian CRF is a log-linear model with

$$p(y|x;\Lambda,\Theta) = \frac{1}{Z(x)} \exp\left\{-\frac{1}{2}y^T\Lambda y - y^T\Theta x\right\} \tag{4.1}$$

where the quadratic term models the conditional dependencies of y and the linear term models the dependence of y on x. The model is parameterized by $\Lambda \in \mathbb{R}^{p\times p}$, corresponding to the inverse covariance of $y|x$ and $\Theta \in \mathbb{R}^{p\times n}$, which captures the dependence of the response variables on the covariates; an illustration of the model is shown in Figure 4.1. It is straightforward to verify that the CRF is simply a reparameterization of a multivariate Gaussian distribution with mean $-\Lambda^{-1}\Theta x$, variance Λ^{-1}, and partition function

$$\frac{1}{Z(x)} = c|\Lambda|^{\frac{1}{2}} \exp\left\{-\frac{1}{2}x^T\Theta^T\Lambda^{-1}\Theta x\right\}. \tag{4.2}$$

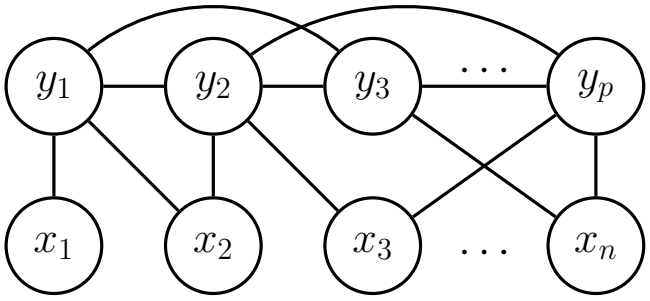

Figure 4.1: Illustration of sparse Gaussian CRF model.

For m data samples, arranged as the rows of $X \in \mathbb{R}^{m \times n}$ and $Y \in \mathbb{R}^{m \times p}$, the negative log-likelihood is given by

$$f(\Lambda, \Theta) = -\log|\Lambda| + \operatorname{tr}\left(S_{yy}\Lambda + 2S_{yx}^T\Theta + S_{xx}\Theta^T\Lambda^{-1}\Theta\right) \tag{4.3}$$

where the S terms are empirical covariances

$$S_{yy} = \frac{1}{m}Y^TY, \quad S_{yx} = \frac{1}{m}Y^TX, \quad S_{xx} = \frac{1}{m}X^TX. \tag{4.4}$$

Without regularization, it is straightforward to verify that maximum likelihood estimation is simply a reparameterization of the least squares problem. We can additionally add ℓ_2 regularization by adding λ_2 to the diagonal elements of S (formally, this corresponds to a Normal-Wishart prior on Λ and the columns of Θ), but again this just corresponds to the regularized least squares. However, the total number of parameters in this problem (for estimating both Θ and Λ) is $np + p(p+1)/2$, and thus model can overfit in the high-dimensional setting when the number of examples m is small relative to dimension of the variables.

To address this concern, we regularize the maximum likelihood estimate by adding ℓ_1 regularization to Θ and the off-diagonal elements of Λ; since the ℓ_1-norm encourages sparsity of the parameters, this directly corresponds to learning a sparse set of edges in our graphical model. Our final optimization problem is then given by minimizing the composite objective

$$\text{minimize} \quad f(\Lambda, \Theta) + \lambda(\|\Lambda\|_{1,\star} + \|\Theta\|_1) \tag{4.5}$$

where $\|\cdot\|_1$ denotes the elementwise ℓ_1-norm, $\|\cdot\|_{1,\star}$ denotes the elementwise ℓ_1-norm on off-diagonal entries, and $\lambda \geq 0$ is the regularization parameter.[1] This is a convex objective, following from the convexity of the ℓ_1-norm and the fact that the log-partition function of an exponential family graphical model is concave.

4.2 The Newton coordinate descent method

[1]It is also possible to introduce different regularization parameters for Λ and Θ, though we have found through our experiments that the optimal settings for these regularization parameters are typically quite similar, so we use only one for simplicity.

Algorithm 4.1: Newton coordinate descent

Input: Smooth function $f : \mathbb{R}^n \to \mathbb{R}$; ℓ_1 regularization parameter λ
Output: $x^\star$ minimizes $f(x) + \lambda\|x\|_1$
Initialize: $x \in \operatorname{dom} f$
while (not converged) **do**
1. *Compute the active set.* $\mathcal{A} = \{i \mid |(\nabla f(x))_i| > \lambda\} \cup \{i \mid x_i \neq 0\}$
2. *Compute the regularized Newton step with coordinate descent.*
$\Delta x = \operatorname{argmin}_{d \in \mathcal{D}_\mathcal{A}} \left(\nabla f(x)^T d + (1/2) d^T \nabla^2 f(x) d + \lambda\|x + d\|_1\right)$
3. *Line search.* Compute step size α using backtracking line search.
4. *Update.* $x \leftarrow x + \alpha\Delta x$
end while

In this section, we describe a general second-order method for solving ℓ_1-regularized optimization problems of the form

$$\text{minimize} \quad f(x) + \lambda\|x\|_1 \tag{4.6}$$

where $f(x)$ is a smooth function. The approach follows the overall structure of a "Newton-Lasso" method [126], also sometimes called proximal Newton methods [21]. In particular, we repeatedly form a second-order approximation to the smooth component of the objective

$$f(x + \Delta) \approx f(x) + \nabla f(x)^T \Delta + \frac{1}{2}\Delta^T \nabla^2 f(x) \Delta, \tag{4.7}$$

which we minimize with the addition of the ℓ_1 regularization term. In other words, we reduce the problem of solving an arbitrary smooth objective with ℓ_1 regularization to repeatedly solving a *quadratic* approximation with ℓ_1 regularization which is typically much easier. The overall procedure requires a relatively small number of iterations to converge to high accuracy, as is expected for second-order methods.

However, the time complexity of this approach depends critically on our ability to compute the ℓ_1-regularized Newton step which cannot be done in closed form. Here we employ coordinate descent, an approach known to perform well for the standard lasso [48] and other ℓ_1-regularized quadratic problems (see e.g. [62, 126, 138]). The strength of this approach is in exploiting sparsity at the algorithmic level: at each iteration we optimize over an active set $\mathcal{A}$ which tends to be much smaller than the overall variable dimension. Formally, define define $\mathcal{D}_\mathcal{A} \equiv \{d : d_i = 0, \forall i \notin \mathcal{A}\}$, the set of candidate directions. Intuitively, these correspond to only those coordinates that are either nonzero or violate the current optimality conditions of the optimization problem. Since the ℓ_1 regularization on the Newton direction tends to push the Newton updates in a direction that increases sparsity and since the line search provably converges to step sizes with $\alpha = 1$ [126], this method tends to generate sparse iterates. Thus, in problems with sparse solutions (the high-dimensional setting), the size of the active set tends to remain small which gives considerable computation advantage. We refer to this method as Newton coordinate descent (Newton-CD), shown in Algorithm 4.1.

In order to apply Newton-CD to a particular $f(x)$ we must be able to efficiently solve the coordinate descent problem for the second-order Taylor expansion which can be somewhat involved depending on the nature of $\nabla^2 f(x)$. Nonetheless, in the presence of a small active set,

solving for the regularized Newton direction with coordinate descent can be significantly cheaper than explicitly forming and inverting the Hessian. In order to achieve an efficient method, attention must be paid to implementing efficient per-coordinate updates using computational tricks such as caching certain matrix-matrix or matrix-vector products.

4.3 Newton-CD for the sparse Gaussian CRF

We discuss the details of applying the Newton-CD method to the sparse Gaussian CRF in this section. Such algorithms have previously been applied to the Gaussian MRF [61, 93], and a general analysis of such methods (showing quadratic convergence) is presented in [126]. The method here largely mirrors the approach in [61] for the Gaussian MRF, but the precise formulation is significantly more involved, owing to the complexity of gradient term of the $\Theta^T \Lambda^{-1} \Theta S_{xx}$ term in the objective. Previous work on the sparse Gaussian CRF model has proposed using off-the-shelf algorithms to solve the above optimization problem, including orthantwise quasi-Newton methods [118] (specifically, the OWL-QN method of [8]), and accelerated proximal gradient methods [146] (specifically, the FISTA algorithm of [11]). These methods are attractive due to their simplicity, but the algorithms suffer from relatively slow convergence and thus quickly become computationally impractical in the high-dimensional setting.

In the sparse Gaussian CRF problem, the gradients of the smooth component of the objective are

$$\begin{aligned} \nabla_\Lambda f(\Lambda, \Theta) &= S_{yy} - \Lambda^{-1} - \Lambda^{-1} \Theta S_{xx} \Theta^T \Lambda^{-1} \\ \nabla_\Theta f(\Lambda, \Theta) &= 2 S_{yx} + 2 \Lambda^{-1} \Theta S_{xx}. \end{aligned} \tag{4.8}$$

The precise formulation of the Hessian terms is cumbersome, due to the fact that all parameters involved are matrices, but we can express the second-order Taylor expansion using differentials.

$$\begin{aligned} &f(\Lambda + \Delta_\Lambda, \Theta + \Delta_\Theta) \approx g(\Delta_\Lambda, \Delta_\Theta) \equiv f(\Lambda, \Theta) + \\ &\quad \operatorname{tr} S_{yy} \Delta_\Lambda + 2 \operatorname{tr} S_{yx} \Delta_\Theta - \operatorname{tr} \Lambda^{-1} \Delta_\Lambda + \\ &\quad 2 \operatorname{tr} \Lambda^{-1} \Theta^T S_{xx} \Delta_\Theta - \operatorname{tr} \Lambda^{-1} \Theta^T S_{xx} \Theta \Lambda^{-1} \Delta_\Lambda + \\ &\quad \operatorname{tr} \Lambda^{-1} \Delta_\Lambda \Lambda^{-1} \Theta^T S_{xx} \Theta \Lambda^{-1} \Delta_\Lambda + \frac{1}{2} \operatorname{tr} \Lambda^{-1} \Delta_\Lambda \Lambda^{-1} \Delta_\Lambda + \\ &\quad \operatorname{tr} \Lambda^{-1} \Delta_\Theta^T S_{xx} \Delta_\Theta - 2 \operatorname{tr} \Lambda^{-1} \Delta_\Lambda \Lambda^{-1} \Theta^T S_{xx} \Delta_\Theta. \end{aligned} \tag{4.9}$$

As above, we compute the regularized Newton steps Δ_Λ, Δ_Θ by

$$\Delta_\Lambda, \Delta_\Theta = \operatorname*{argmin}_{D_\Lambda, D_\Theta} g(D_\Lambda, D_\Theta) + \lambda \left(\|\Lambda + D_\Lambda\|_{1,\star} + \|\Theta + D_\Theta\|_1 \right) \tag{4.10}$$

where we use a coordinate descent algorithm to optimize this ℓ_1-regularized QP. We must also incorporate the domain of f, ensuring that Λ is positive definite which is implemented in the line search by defining $-\log|X| = \infty$ for $X \not\succ 0$.

4.3.1 Fast coordinate updates

In this section, we derive the coordinate descent method for the regularized Newton Step (4.10) and highlight the key optimizations that are used in order to achieve fast performance. We split the per-coordinate updates for D_Λ and D_Θ into three cases. First, consider optimizing over a diagonal element of D_Λ

$$\begin{aligned}
&\underset{\mu}{\text{argmin}} \quad \left(g(D_\Lambda + \mu e_i e_i^T, D_\Theta) + \lambda\left(\|\Lambda + D_\Lambda + \mu e_i e_i^T\|_{1,\star} + \|\Theta + D_\Theta\|_1\right)\right) \\
&= \underset{\mu}{\text{argmin}} \quad ((1/2)\mu^2\left[\Sigma_{ii}^2 + 2\Sigma_{ii}\Psi_{ii}\right] + \\
&\qquad \mu\left(-\Sigma_{ii} + (S_{yy})_{ii} - \Psi_{ii} + (\Sigma D_\Lambda \Sigma)_{ii} - 2(\Sigma D_\Theta S_{xx}\Theta^T\Sigma)_{ii} + 2(\Psi D_\Lambda \Sigma)_{ii}\right) + \\
&\qquad \lambda|\Lambda_{ii} + (D_\Lambda)_{ii} + \mu|)
\end{aligned} \tag{4.11}$$

where $\Sigma = \Lambda^{-1}$ and $\Psi = \Sigma\Theta S_{xx}\Theta^T\Sigma$.

Next, note that for two symmetric matrices A, B the symmetric update is given by

$$\begin{aligned}
&\underset{\mu}{\text{argmin}} \quad \operatorname{tr} A(D_\Lambda + \mu(e_i e_j^T + e_j e_i^T))B(D_\Lambda + \mu(e_i e_j^T + e_j e_i^T)) \\
&= \underset{\mu}{\text{argmin}} \quad \mu^2 \operatorname{tr} A(e_i e_j^T + e_j e_i^T)B(e_i e_j^T + e_j e_i^T) + \mu \operatorname{tr} AD_\Lambda B(e_i e_j^T + e_j e_i^T) + \mu \operatorname{tr} A(e_i e_j^T + e_j e_i^T)BD_\Lambda \\
&= \underset{\mu}{\text{argmin}} \quad \mu^2(A_{ii}B_{jj} + 2A_{ij}B_{ij} + A_{jj}B_{ii}) + 2\mu((AD_\Lambda B)_{ij} + (AD_\Lambda B)_{ji})
\end{aligned}$$

Applying this equivalence twice, once with $A = B = \Sigma$ and again with $A = \Sigma$, $B = \Psi$ the symmetric update for an off-diagonal element of matrix D_Λ

$$\begin{aligned}
&\underset{\mu}{\text{argmin}} \quad \left(g(D_\Lambda + \mu(e_i e_j^T + e_j e_i^T), D_\Theta) + \lambda\left(\|\Lambda + D_\Lambda + \mu(e_i e_j^T + e_j e_i^T)\|_{1,\star} + \|\Theta + D_\Theta\|_1\right)\right) \\
&= \underset{\mu}{\text{argmin}} \quad (\mu^2\left(\Sigma_{ij}^2 + \Sigma_{ii}\Sigma_{jj} + \Sigma_{ii}\Psi_{jj} + 2\Sigma_{ij}\Psi_{ij} + \Sigma_{jj}\Psi_{ii}\right) + \\
&\qquad 2\mu\left(-\Sigma_{ij} + (S_{yy})_{ij} - \Psi_{ij} + (\Sigma D_\Lambda \Sigma)_{ij} - \Phi_{ij} - \Phi_{ji} + (\Psi D_\Lambda \Sigma)_{ij} + (\Psi D_\Lambda \Sigma)_{ji}\right) + \\
&\qquad 2\lambda|\Lambda_{ij} + (D_\Lambda)_{ij} + \mu|)
\end{aligned} \tag{4.12}$$

where $\Phi = \Sigma D_\Theta S_{xx}\Theta^T\Sigma$. Finally, we consider optimizing over an element of D_Θ

$$\begin{aligned}
&\underset{\mu}{\text{argmin}} \quad \left(g(D_\Lambda, D_\Theta + \mu(e_i e_j^T)) + \lambda\left(\|\Lambda + D_\Lambda\|_{1,\star} + \|\Theta + D_\Theta + \mu e_i e_j^T\|_1\right)\right) \\
&= \underset{\mu}{\text{argmin}} \quad (\mu^2\left(\Sigma_{jj}(S_{xx})_{ii}\right] + \mu\left[2(S_{yx})_{ij} + 2(S_{xx}\Theta\Sigma)_{ij} + 2(S_{xx}D_\Theta\Sigma)_{ij} - 2(S_{xx}\Theta\Sigma D_\Lambda\Sigma)_{ij}\right] + \\
&\qquad \lambda|\Theta_{ij} + (D_\Theta)_{ij} + \mu|
\end{aligned} \tag{4.13}$$

Although the notation is heavy due to the number of matrices involved, each update is simply minimizing an ℓ_1-regularized quadratic function over a scalar variable which can be solved in closed form, i.e.

$$\underset{\mu}{\text{argmin}} \frac{1}{2}a\mu^2 + b\mu + \lambda|c + \mu| = -c + S_{\lambda/a}\left(c - \frac{b}{a}\right) \tag{4.14}$$

where S_λ is the soft-thresholding operator.

4.3.2 Computational speedups

In order to make the Newton method computationally efficient, there are a number of needed optimizations. Again, these mirror similar optimization presented in [61], but require adaptations for the CRF case. In practice, the majority of the computational work of the Newton CD method comes from computing the regularized Newton step via coordinate descen. In particular it is important to cache and incrementally update certain matrix products, such that we can evaluate subsequent coordinate updates efficiently. This requires that we maintain an explicit form of the matrix products $\Lambda^{-1}\Delta_\Lambda$ and $\Lambda^{-1}\Delta_\Theta$; crucially, when we update a single coordinate of the Δ_Θ or Δ_Λ, we only need to update a single row of these matrix products, and we can subsequently use only certain elements of these products to compute each coordinate descent step. The algorithm for computing the regularized Newton step via coordinate descent using this matrix caching is shown in Algorithm 4.2.

Algorithm 4.2: Coordinate descent for regularized Newton step in SGCRF

Input: Current iterates Λ, Θ, active sets $\mathcal{A}_\Lambda$, $\mathcal{A}_\Theta$
Output: Regularized Newton direction D_Λ, D_Θ
Initialize: $D_\Lambda \leftarrow 0$, $D_\Theta \leftarrow 0, U \leftarrow 0, V \leftarrow 0$
while (not converged) **do**
  for coordinate (i, j) in $\mathcal{A}_\Lambda$ **do**
    1. *Minimize over coordinate*. Find μ by solving (4.11) or (4.12), using $U = \Lambda^{-1}\Delta_\Lambda$ and $V = \Lambda^{-1}\Delta_\Theta$.
    2. *Update*.

$$\begin{aligned}(D_\Lambda)_{ij}, (D_\Lambda)_{ij} &\leftarrow (D_\Lambda)_{ij} + \mu \\ U_i &\leftarrow U_i + \mu\Sigma_j \\ U_j &\leftarrow U_j + \mu\Sigma_i\end{aligned}$$

    where X_i denotes the ith row of matrix X.
  end for
  for coordinate (i, j) in S_Θ **do**
    1. *Minimize over coordinate*. Find μ by solving (4.13), using U and D_Θ.
    2. *Update*.

$$\begin{aligned}(D_\Theta)_{ij} &\leftarrow (D_\Theta)_{ij} + \mu \\ V_i &\leftarrow V_i + \mu\Sigma_j\end{aligned}$$

  end for
end while

In addition, since each step of our Newton method involves solving an ℓ_1-regularized problem itself, it is important that we solve for this regularized Newton step only to an accuracy that is

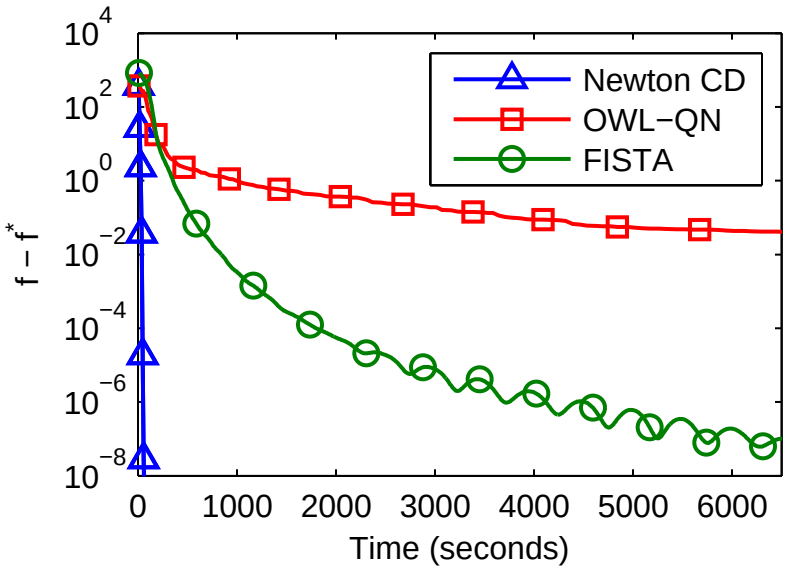

Figure 4.2: Sparse Gaussian conditional random field: comparison of specialized Newton coordinate descent to existing methods.

warranted by the overall accuracy of algorithm as a whole. Although more involved approaches are possible, we simply require that the inner loop makes no more than $O(t)$ passes over the data, where t is the iteration number, a heuristic that is simple to implement and works well in practice.

Finally, in cases where $n \gg m$ (the high-dimensional setting of interest), by not forming the $S_{xx} \in \mathbb{R}^{n \times n}$ matrix explicitly, we can reduce the computation for products involving $X^T X$ from $O(n^2)$ to $O(mn)$. Note that the same considerations do not apply to S_{yy}, since we need to form an invert the $p \times p$ matrix Λ to compute the gradients. Thus, the algorithm still has complexity $O(p^3)$, as in the MRF case. However, this highlights another advantage of the CRF over the MRF: when n is large, just forming a generative model over x and y jointly is prohibitively expensive. Thus, the sparse Gaussian CRF significantly improves both the performance and the computational complexity of the generative model.

4.4 Numerical results

In the numerical examples that follow we generate data following a similar procedure as [146]: Λ and Θ are sampled with $5(n+p)$ random unity entries (the rest being zero), and the diagonal of Λ is such that the condition number is $n+p$. We sample x from a zero-mean Gaussian with full covariance, square half the entries, and then normalize the columns to have unit variance.

4.4.1 Timing results

Figure 4.2 shows the suboptimality of each method in terms of the objective function $f - f^\star$ versus execution time on a problem with dimensions $p = 1000$, $n = 4000$, and $m = 2500$.

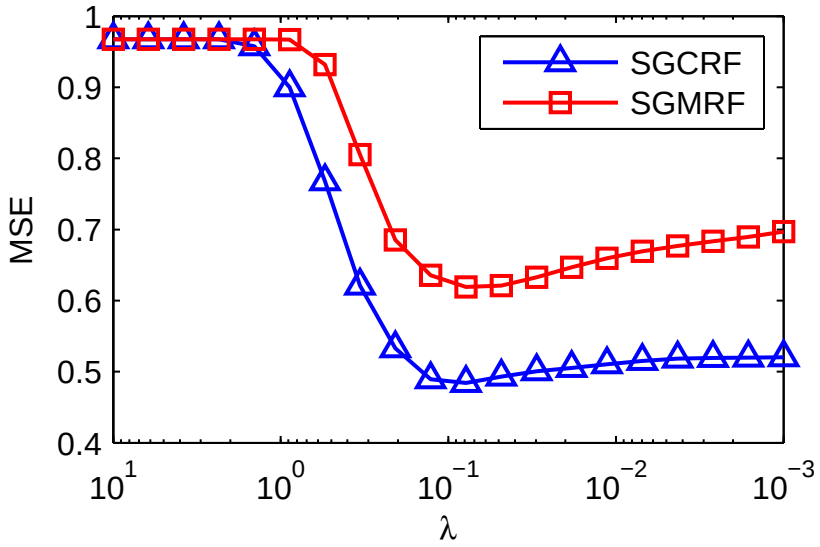

Figure 4.3: Generalization performance (measured by mean squared error of the predictions) for the Gaussian MRF versus CRF.

On this problem, the Newton CD approach converges to high numerical precision within about 81 seconds, while FISTA and OWL-QN still don't approach this level of precision after two hours. It is also important to note that the Newton CD approach also reaches all intermediate levels of accuracy faster than the alternative approaches, so that the algorithm is preferable even if only intermediate precision is desired. Indeed, we note previous works [118, 146] considered maximum problem sizes of $np \approx 10^5$; since much of the appeal of ℓ_1 approaches lies precisely in the ability to use large feature sizes, this has significantly limited the applicability of the approach. We thus believe that our proposed algorithms opens the possibility of substantial new applications of this sparse Gaussian CRF model.

4.4.2 Comparison to MRF

Our next experiment compares the discriminative CRF modeling to a generative MRF model. In particular, an alternative approach to our framework is to use a sparse Gaussian MRF to jointly model x, y as a Gaussian, then compute $y|x$. Figure 4.3 shows the performance of the Gaussian MRF versus CRF (problem dimensions $n = 200$, $p = 50$, $m = 50$), measured by mean squared error in the predictions on a test set, over a variety of different λ parameters. The CRF substantially outperforms the MRF in this case, due to two main factors: 1) the x variables as generated by the above process are not Gaussian, and thus any Gaussian distribution will model them poorly; and 2) the x variables are correlated and have dense inverse covariance, making it difficult for the MRF to find a sparse solution.

We also note that in addition to the modeling benefits, the CRF has substantial computational benefits. Modeling x and y jointly requires computing and inverting their joint covariance, which takes time $O((n+p)^3)$; in contrast, the corresponding operations for the CRF case are

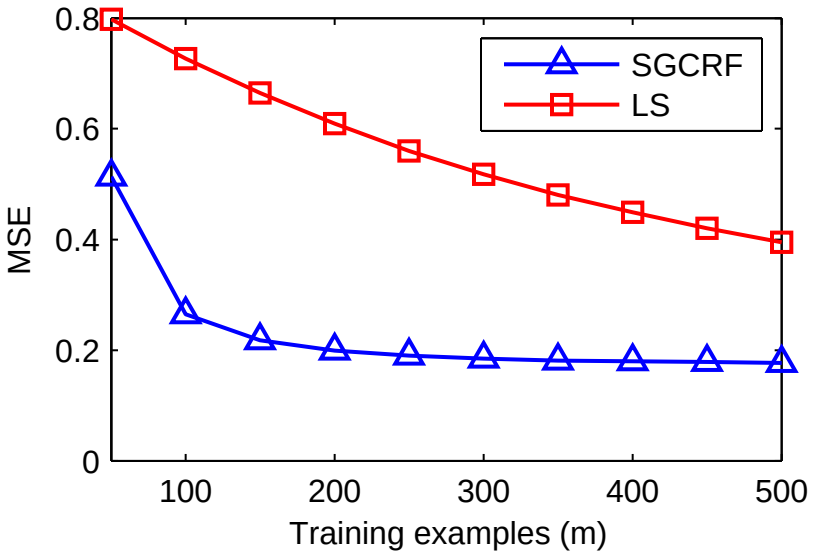

Figure 4.4: Generalization performance (MSE for the best λ chosen via cross-validation), for the sparse Gaussian CRF versus ℓ_2-regularized least squares.

$O(np^3)$, which is substantially faster for even modestly large n. Indeed, in real-world experiments, we were unable to successfully optimize a joint MRF using the QUIC algorithm of [61] (itself amongst the fastest for solving the sparse Gaussian MRF), after running the algorithm for 24 hours.

4.4.3 ℓ_1 and ℓ_2 regularization vs. sample size

Finally, to illustrate the benefit of ℓ_1 regularization over traditional (ℓ_2-regularized) multiple least squares estimation, we evaluate generalization performance versus sample size, shown in Figure 4.4 (here $n = 800$, $p = 200$). This figure shows performance measured by mean squared error of the ℓ_1-regularized sparse Gaussian CRF versus traditional least squares with ℓ_2 regularization; for each m we choose the ℓ_1 and ℓ_2 regularization parameters using validation data, then evaluate the MSE on separate test data. As the sample size increases, the performance of the two methods becomes similar (in the limit of infinite data with fixed n and p, they will of course be equivalent); however, as expected, for small samples sizes the ℓ_1 regularization method performs much better, being able to take advantage of the sparsity in the underlying model.

Chapter 5

The Sparse Linear-Quadratic Regulator

This chapter considers the task of designing sparse linear control laws for large-scale linear systems. Sparsity and decentralized control have a long history in the control literature: unlike the centralized control setting, which in the $\mathcal{H}_2$ and $\mathcal{H}_\infty$ settings can be solved optimally [7], it has been known for some time that the task of finding an optimal control law with structure constraints is a hard problem [17]. Witenhausen's counterexample [135] famously demonstrated that even for a simple linear system, a linear control law is no longer optimal, and subsequent early work focused on finding effective decentralized controllers for specific problem instances [72] or determining the theoretical existence of stabilizing distributed control methods [134]; they survey of Sandell et al. [109] covers many of these earlier approaches in detail.

In recent years, there has been an increasing interest in sparse and decentralized control methods, spurred 1) by increasing interest in large-scale systems such as the electrical grid, where some form of decentralization seems critical for practical control strategies, and 2) by increasing computational power that can allow for effective controller design methods in such systems. Generally, this work has taken one of two directions. On the one hand, several authors have looked at restricted classes of dynamical systems where the true optimal control law is provably sparse, resulting in efficient methods for computing optimal decentralized controllers [43, 103, 106, 114]. A notable recent example of such work has been the characterization of all systems that admit convex constraints on the controller (and thus allow for exact sparsity-constrained controller design) using the notion of quadratic invariance [106, 122]. On the other hand, an alternative approach has been to search for *approximate* (suboptimal) decentralized controllers, either by directly solving a nonconvex optimization problem [75], by constraining the class of allowable Lyapunov functions in a convex parameterization of the optimal $\mathcal{H}_2$ or $\mathcal{H}_\infty$ controllers [112, 113] or by employing a convex, alternative optimization objective as opposed to the typical infinite horizon cost [39, 40]. We build upon this second line of work, specifically the framework established in [75], which uses ℓ_1 regularization (amongst other possible regularizers) to *discover* the good sparsity patterns in the control law (though our method also applies directly to the case of a fixed sparsity pattern).

Despite the aforementioned work, an element that has been notably missing from past work in the area is a focus on the *algorithmic* approaches that can render these methods practical for large-scale systems, such as those with thousands of states and controls or more. Indeed, it is precisely for such systems that sparse and decentralized control is most appealing, and yet

most past work we are aware of has focused solely on semidefinite programming formulation of the resulting optimization problems [112, 113] (which scale poorly in off-the-shelf solvers), or very approximate first-order or alternating minimization methods [75] such as the alternating direction method of multipliers [19]. As a result, most of the demonstrated performance of the methods in these past papers has focused solely on relatively small-scale systems. In contrast, sparse methods in fields like machine learning and statistics, which have received a great deal of attention in recent years [23, 36, 123], evolved simultaneously with efficient algorithms for solving these statistical estimation problems [41, 68]. The goal of this work is to push this algorithmic direction in the area of sparse control, developing methods that can handle large-scale sparse controller design.

5.1 Problem formulation

Here we formally define our control and optimization framework, based upon the setting in [75]. Formally, we consider the linear Gaussian system

$$\dot{x}(t) = Ax(t) + Bu(t) + W^{1/2}\epsilon(t) \tag{5.1}$$

where $x \in \mathbb{R}^n$ denotes the state variables, $u \in \mathbb{R}^m$ denotes the control inputs, ϵ is a zero-mean Brownian motion process, $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are system matrices, and $W \in \mathbb{R}^{n\times n}$ is a noise covariance matrix. We seek to optimize the infinite horizon LQR cost for a linear state-feedback control law $u(t) = Kx(t)$ for $K \in \mathbb{R}^{m\times n}$,

$$J(K) = \lim_{T\to\infty} \frac{1}{T}\mathbf{E}\left[\int_0^T \left(x(t)^T Q x(t) + u(t)^T R u(t)\right) dt\right] \tag{5.2}$$

which can also be written in the alternative form (see e.g. [105] for details)

$$J(K) = \begin{cases} \operatorname{tr} PW = \operatorname{tr} L(Q + K^T R K) & A + BK \text{ stable} \\ \infty & \text{otherwise.} \end{cases} \tag{5.3}$$

where $L = L(K)$ and $P = P(K)$ are the unique solutions to the Lyapunov equations

$$\begin{aligned} (A + BK)L + L(A + BK)^T + W = 0 \\ (A + BK)^T P + P(A + BK) + Q + K^T R K = 0. \end{aligned} \tag{5.4}$$

When K is unconstrained, it is well-known that the problem can be solved by the classical LQR algorithm, though this results in a dense (i.e., centralized) control law, where each control will tend to depend on each state.

To encourage sparsity in the controller, [75] proposed to add an additional penalty to the ℓ_1-norm of the controller K. Here we will consider this framework with a weighted ℓ_1-norm: we are concerned with solving the optimization problem

$$\text{minimize } J(K) + g(K) \tag{5.5}$$

where $J(K)$ defined in (5.2) is the LQR cost (or the $\mathcal{H}_2$-norm for output $z(t) = (Q^{1/2}x(t), R^{1/2}u(t))$ and treating $\epsilon(t)$ as a disturbance input) and $g(K)$ is the sparsity-promoting penalty which we take to be

$$g(K) = \|\Lambda \circ K\|_1 = \sum_{ij} \Lambda_{ij}|K_{ij}|, \tag{5.6}$$

a weighted version of the ℓ_1-norm. This formulation allows us to use this single algorithmic framework to capture both ℓ_1 regularization of the control matrix, as well as optimization over a fixed pattern of nonzeros, by setting the appropriate elements of Λ to 0 or ∞.

Our algorithm uses gradient and Hessian information extensively. Following standard results [105], the gradient of $J(K)$ is given by

$$\nabla J(K) = 2(B^T P + RK)L \tag{5.7}$$

when $A + BK$ is stable. The Hessian is somewhat cumbersome to formulate directly, but we can concisely write its inner product (see [105]) with a direction $D \in \mathbb{R}^{m\times n}$ as

$$\begin{aligned} &\operatorname{vec}(D)^T \nabla^2 J(K) \operatorname{vec}(D) = \\ &\qquad 2\operatorname{tr}\left(\tilde{L}(PB + K^T R) + L(\tilde{P}B + D^T)R\right) D \end{aligned} \tag{5.8}$$

where $\operatorname{vec}(\cdot)$ denotes the vectorization of a matrix, and $\tilde{L} = \tilde{L}(K, D)$ and $\tilde{P} = \tilde{P}(K, D)$ are the unique solutions to two Lyapunov equations

$$\begin{aligned} (A + BK)\tilde{L} + \tilde{L}(A + BK)^T + BDL + LD^T B^T = 0 \\ (A + BK)^T \tilde{P} + \tilde{P}(A + BK) + ED + D^T E^T = 0, \end{aligned} \tag{5.9}$$

denoting $E = PB + K^T R$ for brevity. Since solving these Lyapunov equations takes $O(n^3)$ time, evaluating the function and gradient, or evaluating a single inner product with the Hessian, are all $O(n^3)$ operations.

Traditionally, direct second-order Newton methods have seen relatively little application in Lyapunov-based control, precisely because computing these Hessian terms is computationally intensive. Instead, typical approaches have focused on approaches that use gradient information only, either in a quasi-Newton setup [105], or by including only certain terms from the Hessian as in the Anderson-Moore method [7]. However, since a single iteration of any first-order approach is already a reasonably expensive $O(n^3)$ operation, the significantly reduced iteration count of typical Newton methods is appealing, provided we have a way to efficiently compute the Newton step. The algorithm we propose in the next section does precisely that, bringing down the complexity of a Newton step to a computational cost similar to that of a single evaluation of the objective function.

5.2 Newton-CD for sparse LQR

We solve the sparse LQR optimization problem with the Newton-CD method introduced in Section 4.2, an approach that works particularly well in this setting due to two elements:

1. Coordinate descent methods allow us to optimize only over a relatively small "active set" $\mathcal{A}$ of size $k \ll mn$, which includes only the nonzero elements of K plus elements with large gradient values. For problems that exhibit substantial sparsity in the solution, this often lets us optimize over much fewer elements than would be required if we considered all the elements of K at each iteration.
2. By properly precomputing certain terms, caching intermediate products, and exploiting problem structure, we can reduce the per-coordinate-update computation in coordinate descent from $O(n^3)$ (the naive solution, since each coordinate update requires computing an inner product with the Hessian matrix) to $O(n)$.

For the sparse LQR, we form the second-order Taylor expansion

$$\begin{aligned} J(K+\Delta) &\approx \operatorname{tr} \nabla J(K)^T \Delta + \frac{1}{2} \operatorname{vec}(\Delta)^T \nabla^2 J(K) \operatorname{vec}(\Delta) \\ &\equiv \tilde{J}_K(\Delta) \end{aligned} \tag{5.10}$$

which in our problem takes the form

$$\tilde{J}_K(\Delta) = 2 \operatorname{tr} LE\Delta + \operatorname{tr} \tilde{L}E\Delta + \operatorname{tr} L\tilde{P}B\Delta + \operatorname{tr} L\Delta^T R\Delta \tag{5.11}$$

where $\tilde{L}$ and $\tilde{P}$ are defined implicitly as the unique solutions to the Lyapunov equations given in (5.9).

As discussed in Section 4.2, in contrast to the standard Newton method, Newton-CD minimizes the second-order approximation with the addition of (weighted) ℓ_1 regularization

$$\Delta = \arg \min_{D \in \mathcal{D}_{\mathcal{A}}} \tilde{J}_K(D) + \|(K+D) \circ \Lambda\|_1 \tag{5.12}$$

and updates $K \leftarrow K + \alpha\Delta$ where α is chosen using backtracking line search. Intuitively, it can be shown that $\alpha \to 1$ as the algorithm progresses, causing the weighted ℓ_1 penalty on $K + D$ to shrink D in the direction that promotes sparsity in $K+\Delta$; the weights Λ control how aggressively we shrink each coordinate: $\Lambda_{ij} \to \infty$ forces $(K + \Delta)_{ij} \to 0$. Importantly, we note that with this formulation we will never choose a K such that $A + BK$ is unstable; this would make the resulting objective infinite, and a smaller step size would be preferred by the backtracking line search.

In order to find the regularized Newton step efficiently, we use coordinate descent which is appealing for ℓ_1-regularized problems as each coordinate update can be computed in closed form. This reduces (5.12) to iteratively minimizing each coordinate

$$\hat{\mu} = \operatorname*{argmin}_{\mu} \left(\tilde{J}_K(D + \mu e_i e_j^T) + \Lambda_{ij} |K_{ij} + D_{ij} + \mu| \right) \tag{5.13}$$

where e_i denotes the ith basis vector, and then setting $D \leftarrow D + \hat{\mu} e_i e_j^T$. Since the second-order approximation $\tilde{J}_K(D)$ is quite complex (it depends on the solution to four Lyapunov equations, two of which depend on D), deriving efficient coordinatewise updates is somewhat involved. In the next section we describe how each coordinate descent iteration can be computed in $O(n)$ time by precomputing a single eigendecomposition and solving the Lyapunov equations explicitly.

Algorithm 5.1: Coordinate descent for regularized Newton step in SLQR

Input: Stochastic linear system A, B, W; regularization parameters Q, R, Λ; current iterate K; active set $\mathcal{A}$; solution to Lyapunov equations L, P

Output: Regularized Newton step Δ

Initialize: $D \leftarrow 0$, $\Psi \leftarrow 0$

1. Compute the eigendecomposition $A + BK = USU^{-1}$

2. Let $\Theta_{ij} = 1/(S_{ii} + S_{jj})$ and compute $\Theta = XX^T$

3. Precompute matrix products as in (5.27)

while (not converged) **do**

  for coordinate (i, j) in $\mathcal{A}$ **do**

    1. Compute a, b, c according to (5.29) and set

$$\mu = -c + S_{\lambda/a}\left(c - \frac{b}{a}\right)$$

    2. Update solution

$$D_{ij} \leftarrow D_{ij} + \mu$$

    3. Update the cached matrix products

$$\begin{aligned}
(\Psi^0)_j &\leftarrow (\Psi^0)_j + \mu R_i \\
(\Psi^1_k)_i &\leftarrow (\Psi^1_k)_i + \mu(\Phi^1_k)_j \\
(\Psi^2_k)_j &\leftarrow (\Psi^2_k)_j + \mu(\Phi^4_k)_i \\
(\Psi^3_k)_j &\leftarrow (\Psi^3_k)_j + \mu(\Phi^2_k)_i \\
(\Psi^4_k)_j &\leftarrow (\Psi^4_k)_j + \mu(\Phi^3_k)_i
\end{aligned}$$

    where (in general) A_j denotes the jth column of A

  end for

end while

5.2.1 Fast coordinate updates

To begin, we consider the explicit forms for $\tilde{L}(K, D)$ and $\tilde{P}(K, D)$, the unique solutions to the Lyapunov equations depending on D. Since each coordinate descent update changes an element of D, a naive approach would require re-solving these two Lyapunov equations and $O(n^3)$ operations per iteration. Instead, assuming that $A+BK$ is diagonalizable, we precompute a single eigendecomposition $A + BK = USU^{-1}$ and use this to compute the solutions to the Lyapunov equations directly. For example, the equation describing $\tilde{L}$ can be written as

$$USU^{-1}\tilde{L} + \tilde{L}U^{-T}SU^T + BDL + LD^TB^T = 0 \tag{5.14}$$

and pre- and post-multiplying by U^{-1} and U^{-T} respectively gives

$$S\tilde{L}_U + \tilde{L}_U S = -U^{-1}(BDL + LD^TB^T)U^{-T} \tag{5.15}$$

where $\tilde{L}_U = U^{-1}\tilde{L}U^{-T}$. Since S is diagonal, this equation has the solution

$$(\tilde{L}_U)_{ij} = -\frac{(U^{-1}(BDL + LD^TB^T)U^{-T})_{ij}}{S_{ii} + S_{jj}} \tag{5.16}$$

which we rewrite as the Hadamard product

$$\tilde{L}_U = U^{-1}(BDL + LD^TB^T)U^{-T} \circ \Theta \tag{5.17}$$

with $\Theta_{ij} = -1/(S_{ii} + S_{jj})$.

Fast Θ multiplication via the Fast Multiple Method. Precomputing the eigendecomposition of $A+BK$ in this manner immediately allows for an $O(n^2)$ algorithm for evaluating Hessian products, but reducing this to $O(n)$ requires exploiting additional structure in the problem. In particular, we consider the form of the Θ matrix above, which is an example of a Cauchy matrix, that is, matrices with the form $C_{ij} = 1/(a_i - b_j)$. Like several other special classes of matrices, matrix-vector products with a Cauchy matrix can be computed more quickly than for a standard matrix. In particular, the Fast Multiple Method (FMM) [54], specifically the 2D FMM using the Laplace kernel, provides an $O(n)$ algorithm (technically $O(n \log \frac{1}{\epsilon})$ where ϵ is the desired accuracy) for computing the matrix vector product between a Cauchy matrix and an arbitrary vector.[1]

Although the FMM provides a theoretical method for quickly computing Hessian inner products, in our setting the overhead involved with actually setting up the factorization (which also takes $O(n)$ time, but with a relatively larger constant) would make using an off-the-shelf implementation of the FMM quite costly. However, our setting in fact is somewhat easier as Θ is fixed per outer Newton iteration; thus we can factor Θ once at (relatively) high computation cost and then directly use this factorization is subsequent iterations. Each FMM operation implicitly factors Θ in a hierarchical manner with blocks of low-rank structure, though here the situation is

[1] In theory, such matrix-vector products for Cauchy matrices can be computed exactly in time $O(n \log^2 n)$ [49], but these approaches are substantially less numerically robust than the FMM, so the FMM is typically preferred in practice [96].

simpler: since we maintain $A + BK$ to be stable at each iteration, all the eigenvalues are in the left half plane and representing Θ as a Cauchy matrix

$$\Theta_{ij} = \frac{1}{a_i - b_j}, \tag{5.18}$$

i.e., $a_i = S_{ii}$, $b_j = -S_{jj}$ leads to points, $a_i, b_j \in \mathbb{C}$ that are *separated* in the context of the FMM. This means that Θ in fact simply admits a low-rank representation (though the actual rank will be problem-specific, and depend on how close the eigenvalues of $A + BK$ are to the imaginary axis). Thus, while slightly more advanced factorizations may be possible, for the purposes of this work we simply use the property, based upon the FMM, that Θ will typically admit a low-rank factorization.

The Autonne-Kagaki factorization of Θ. Using the above property, we can compute the optimal low rank factorization of Θ using the (complex) singular value decomposition to obtain a factorization $\Theta = XY^*$. But since Θ is a complex symmetric (but not Hermitian) matrix, it also can be factored as $\Theta = VSV^T$ where S is a diagonal matrix of the singular values of Θ and V is a complex unitary matrix [59, Corollary 2.6.6]. This factorization lets us speed up the resulting computations by 2-fold over simply using an SVD, as we have significantly fewer matrices to precompute in the sequel.

Specifically, writing $\Theta = \sum_{i=1}^n x_i x_i^T$, and using the fact that for a Hadamard product

$$A \circ ab^T = \operatorname{diag}(a) A \operatorname{diag}(b). \tag{5.19}$$

we can write the Lyapunov solution $\tilde{L}$ analytically as

$$\tilde{L} = U\tilde{L}_U U^T = \sum_{k=1}^{r} X_i (BDL + (BDL)^T) X_i^T \tag{5.20}$$

where we let $X_i = U \operatorname{diag}(x_i) U^{-1}$, the transformed version of the diagonal matrix corresponding to the ith column of X. With the same approach, we write the explicit form for $\tilde{P}$ as

$$\tilde{P} = \sum_{i=1}^{r} X_i^T ((ED)^T + ED) X_i. \tag{5.21}$$

Using these explicit forms for $\tilde{L}$ and $\tilde{P}$ we observe that $\operatorname{tr} \tilde{L} ED = \operatorname{tr} \tilde{P} BDL$ and the second-order Taylor expansion simplifies to

$$\begin{aligned} \tilde{J}_K(D) &= 2 \operatorname{tr} LED + \operatorname{tr} LD^T RD \\ &\quad + 2 \operatorname{tr} \sum_{i=1}^{r} X_i^T ((ED)^T + ED) X_i BDL. \end{aligned} \tag{5.22}$$

Closed-form coordinate updates. Next, we consider coordinatewise updates to minimize $\tilde{J}_K(D)$ with the addition of ℓ_1 regularization. In particular, consider optimizing over μ the rank one update $D + \mu e_i e_j^T$; for each term we get a quadratic function in μ with coefficients that depend on several matrix products. For example, the second term of $LD^T RD$ yields

$$\begin{aligned} &\operatorname{tr} L(D + \mu e_i e_j^T)^T R (D + \mu e_i e_j^T) \\ &= \operatorname{tr} LD^T RD + 2\mu (RDL)_{ij} + \mu^2 R_{ii} L_{jj}. \end{aligned} \tag{5.23}$$

For each term in $\tilde{J}_K(D)$, we repeat these steps to derive

$$\begin{aligned}&\arg\min_{\mu} \tilde{J}_K(D + \mu e_i e_j^T) + \|(K + D + \mu e_i e_j^T) \circ \Lambda\|_1 \\ =&\arg\min_{\mu} \frac{1}{2} a\mu^2 + b\mu + \|c + \mu\|_1\end{aligned} \tag{5.24}$$

where

$$\begin{aligned}a &= 2R_{ii}L_{jj} \\ &\quad + 4\left(\sum_{k=1}^{r}(E^T X_k B)_{ii}(LX_k^T)_{jj} + (LX^T E)_{ji}(XB)_{ji}\right) \\ b &= 2(E^T L)_{ij} + 2(RDL)_{ij} \\ &\quad + 2\left(\sum_{k=1}^{r}(E^T X_k BDLX_k^T)_{ij} + (B^T X_k^T EDX_k L)_{ij}\right) \\ &\quad + 2\left(\sum_{k=1}^{r}(X_k BDLX_k^T E)_{ji} + (LX_k^T EDX_k B)_{ji}\right) \\ c &= K_{ij} + D_{ij}.\end{aligned} \tag{5.25}$$

This has the closed form solution

$$\mu = -c + S_{\lambda/a}\left(c - \frac{b}{a}\right) \tag{5.26}$$

where S_λ is the soft-thresholding operator.

Caching matrices. Naive computation of these matrix products for a, b and c still requires $O(n^3)$ operations; however, all matrices except D remain fixed over each iteration of the inner loop, allowing us to precompute many matrix products. Let

$$\begin{aligned}\Phi^0 &= LE \\ \Phi_k^1 &= X_k L \\ \Phi_k^2 &= X_k B \\ \Phi_k^3 &= LX_k^T E \\ \Phi_k^4 &= B^T X^T E.\end{aligned} \tag{5.27}$$

In addition as we iteratively update D, we also maintain the matrix products

$$\begin{aligned}\Psi^0 &= RD \\ \Psi_k^1 &= \Phi_k^1 D^T \\ \Psi_k^2 &= \Phi_k^4 D \\ \Psi_k^3 &= \Phi_k^2 D \\ \Psi_k^4 &= \Phi_k^3 D\end{aligned} \tag{5.28}$$

which allows us to efficiently compute

$$
\begin{aligned}
a &= 2R_{ii}L_{jj} + 4\left(\sum_{k=1}^{r}(\Phi_k^4)_{ii}(\Phi_k^1)_{jj} + (\Phi_k^3)_{ij}(\Phi_k^2)_{ji}\right) \\
b &= 2\left((\Phi^0)_{ji} + (\Psi^0 L)_{ij}\right) \\
&\quad + 2\left(\sum_{k=1}^{r}(\Psi_k^1\Phi_k)_{ji} + (\Psi_k^2\Phi_k^1)_{ij}\right) \\
&\quad + 2\left(\sum_{k=1}^{r}(\Psi_k^3\Phi_k^3)_{ji} + \Psi_k^4\Phi_k^2)_{ji}\right) \\
c &= K_{ij} + D_{ij}
\end{aligned}
\tag{5.29}
$$

resulting in an $O(n)$ time per iteration (in general we can compute an element of a matrix product $(AB)_{ij}$ as the dot product between the ith row of A and the jth column of B). Updating the cached products Ψ also requires $O(n)$ time as a change to a single coordinate of D requires modifying a single row or column of one of the products Ψ. The complete algorithm is given in Algorithm 5.1 and has $O(n)$ per iteration complexity as opposed to the $O(n^3)$ naive implementation.

5.2.2 Additional algorithmic elements

While the above algorithm describes the basic second-order approach, several elements are important for making the algorithm practical and robust to a variety of different systems.

Initial conditions. One crucial element that affects the algorithm's performance is the choice of initial K matrix. Since the objective $J(K)$ is infinite for $A + BK$ unstable, we require that the initial value must stabilize the system. We could simply choose the full LQR controller K^{LQR} as this initial point; it may take time $O(n^3)$ to compute the LQR solution, but since our algorithm is $O(n^3)$ overall, this is typically not a prohibitive cost. However, the difficulty with this strategy is that the resulting controller is not sparse, which leads to a full active set for the first step of our Newton-CD approach, substantially slowing down the method. Instead, a single soft-thresholding step on the LQR solution produces a good initial starting point that is both guaranteed to be stable, and which leads to a much smaller active set in practice. Formally, we compute

$$K^{(0)} = S_{\alpha\Lambda}(K^{\mathrm{LQR}}) \tag{5.30}$$

where $\alpha \leq 1$ is chosen by backtracking line search such that the regularized objective decreases and $K^{(0)}$ remains stable. In addition, if the goal is to sweep across a large range of possible regularization parameters, we can employ a "warm start" method that initializes the controller to the solution of previous optimization problems.

Unstable initial controllers. In the event where we do not want to start at the LQR solution, it is also possible to begin with some initial controller $K^{(0)}$ that is *not* stabilizing using a "deflation" technique. Specifically, rather than find an optimal control law for the linear system (A, B), we find a controller for the linear system $(A-\nu I, B)$ where ν is chosen such that $A-\nu I+BK^{(0)}$

is stable with some margin. The resulting controller $K^{(1)}$ will typically stabilize the system to a larger degree, and we can repeat this process until it produces a stabilizing control law. Further, we typically do not need to run the Newton-CD method to convergence, but can often obtain a better stabilizing control law after only a few outer iterations.

Handling non-convexity. As mentioned above, the objective $J(K)$ is not a convex function, and so can (and indeed, often does in practice) produce indefinite Hessian matrices. In such cases, the coordinate descent steps are not guaranteed to produce a descent direction, and indeed can cause the overall descent direction to diverge. Furthermore, since we never compute the full Hessian $\nabla_K^2 J(K)$ (even restricted to just the active set), it is difficult to perform typical operations to handle non-convexity such as projecting the Hessian onto the positive definite cone. Instead, we handle this non-convexity by a fallback to a simpler quasi-Newton coordinate descent scheme [126]. At each Newton iteration we also form a coordinate descent update based upon a *diagonal* PSD approximation to the Hessian,

$$\bar{J}(K+D) = \operatorname{tr} \nabla J(K)^T D + \frac{1}{2} \operatorname{vec}(D)^T H \operatorname{vec}(D) \tag{5.31}$$

where

$$H_{ii} = \min\{\max\{10^{-2}, (\nabla^2 J(K))_{ii}\}, 10^4\}. \tag{5.32}$$

The diagonal terms of the Hessian are precisely the a variables that we compute in the coordinate descent iterations anyway, so this search direction can be computed at little additional cost. Then, we simply perform a line search on both update directions simultaneously, and choose the next iterate with the largest improvement to the objective. In practice, the algorithm sometimes uses the fallback direction in the early iterations of the method until it converges to a convex region around a (local) optimum where the full Newton step causes much larger function decreases and so is nearly always chosen. This fallback procedure also provides a convergence guarantee for our method: the quasi-Newton coordinate descent approach was analyzed in [126] and shown to converge for both convex and non-convex objectives. Since our algorithm always takes at least as good a step as this quasi-Newton approach, the same convergence guarantees hold here.

Inner and outer loop convergence and approximation. Finally, a natural question that we do not address directly in the above algorithmic presentations involves how many iterations to use (both inner and outer), as well as what constitutes a sufficient approximation for certain terms like the rank of Θ. In practice, a strength of the Newton-CD methods is that it can be fairly insensitive to slightly less accurate inner loops [21]: in such cases, the approximate Newton direction is still typically much better than a gradient direction, and while additional outer loop iterations may be required, the timing of the resulting method is somewhat insensitive to choice of parameters for the inner loop convergence and for different low-rank approximations to Θ. In our implementation, we run the inner loop for at most $t/3$ iterations at outer loop iteration t or until the relative change in the direction D is less than 10^{-2} in the Frobenius norm.

5.3 Numerical results

In this section we evaluate the performance of the proposed algorithm on the task of finding sparse optimal controllers for a synthetic mass-spring system and wide-area control in power

systems. In both settings, the method finds sparse controllers that perform nearly optimally while only depending on a small subset of the state space; furthermore, as we scale to larger examples, we demonstrate that these optimal controllers become *more sparse*, highlighting the increased role of sparsity in larger systems.

Computationally, we compare the convergence rate of our algorithm to that of existing approaches for solving the sparse optimal control problem and demonstrate that the proposed method converges rapidly to highly accurate solutions, significantly outperforming previous approaches. Although the solution accuracy required for a "good enough" controller is problem-specific, since per-iteration complexity grows with the dimension of the state space as $O(n^3)$, faster methods that reach an accurate solution in a small number of iterations are strongly preferred. We also note that, if one uses the ℓ_1 regularization penalty solely as a heuristic for encouraging sparsity, then finding an exact (locally) optimal solution may be less important than merely finding a solution with a reasonable sparsity pattern, which can indeed be accomplished by a variety of algorithms. However, given that we are using the ℓ_1 heuristic in the first place, and since in practice the ℓ_1 penalty has a similar "shrinkage" effect as increasing the respective R penalty on the controls, it is reasonable to seek out as accurate a solution as possible to this optimization problem. We demonstrate that for all levels of accuracy and on both sets of examples considered, our second-order method is significantly faster than existing approaches. In particular, for large problems with thousands of states, our method reaches a reasonable level of accuracy in minutes whereas previous approaches take hours.

Specifically, we compare our algorithm to two other approaches: the original alternating direction method of multipliers (ADMM) method from [75] which has as an inner loop the Anderson-Moore method; and iterative soft-thresholding (ISTA), a proximal gradient approach which iterates between a gradient step and the soft-thresholding projection. In the ISTA implementation, in order to ensure that we maintain the stability of $A+BK$ we perform a line search to choose the step size for each iteration. Although it is also possible to add acceleration to ISTA, resulting in the FISTA algorithm, this is known to perform poorly on nonconvex problems, a behavior which we observed in our own experiments.

In each set of experiments, we first solve the ℓ_1 regularized objective with a weight λ placed on all elements of K. Once this has converged, we perform a second polishing pass with the sparsity of K fixed to the nonzero elements of the optimal solution to the ℓ_1 problem, optimizing performance on the LQR objective for a given level of sparsity. The polishing step can also be performed efficiently using the Newton-CD method and Λ with elements equal to 0 or ∞. Finally, when solving the ℓ_1 regularized problem in the first step, we soft-threshold the LQR solution as described in Section 5.2.2; this is relatively quick compared to the overall running time and we use the same initial controller $K^{(0)}$ as the starting point for all algorithms.

5.3.1 Mass-spring system

In our first example we consider the mass-spring system from [75] describing the displacement of N masses connected on a line. The state space is comprised of the position and velocity of

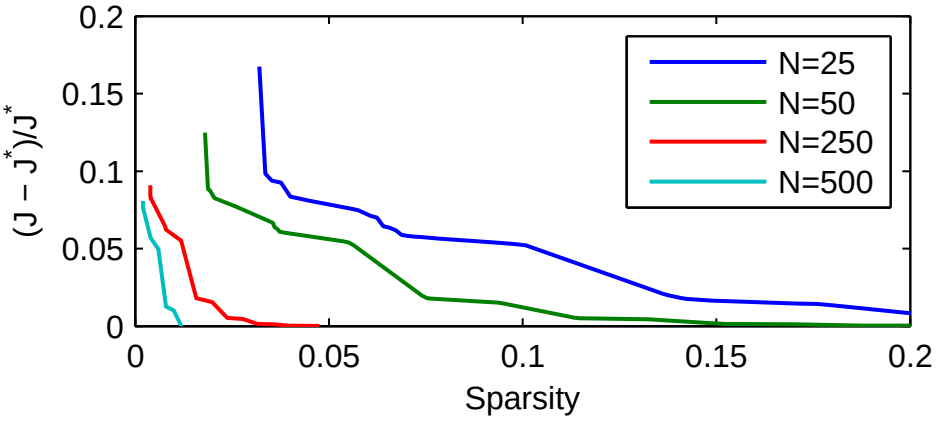

Figure 5.1: Comparison of sparse controllers to the optimal LQR control law J^* for varying levels of sparsity on the mass-spring system.

each mass with dynamics given by the linear system

$$A = \begin{bmatrix} 0 & I \\ T & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{5.33}$$

where I is the $N \times N$ identity matrix, and T is an $N \times N$ tridiagonal symmetric Toeplitz matrix of the form

$$T = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix}; \tag{5.34}$$

we take $Q = I, R = 10I, W = BB^T$ as in the previous paper.

We begin by characterizing the trade-off between sparsity and system performance by sweeping across 100 logarithmically spaced values of λ. For the system with $N = 50$ springs, the results shown in Figure 5.1 are nearly identical to those reported by [75], although their methodology includes an additional loop and iteratively solving a series of reweighted ℓ_1 problems. For all systems, the leftmost point represents a control law based almost entirely on local information—although the results shown penalize the elements of K uniformly, we also found that by regularizing just the elements of K corresponding to nonlocal feedback we were able to find stable local control laws in all examples. As λ decreases, the algorithm finds controllers that quickly approach the performance of LQR and in the smallest example we require a controller with 18% nonzero elements to be within 0.1% of the LQR performance; in the largest example we require only and 4.0% sparsity to reach this level. This demonstrates the trend that we anticipate: larger systems require comparatively sparser controllers for optimal performance.

Next we compare the running times of each algorithm by fixing λ and considering the convergence of the objective value f at each iteration to the (local) optimum f^*. Figure 5.2 shows three such fixed λ settings corresponding to the levels of sparsity of interest in the mass-spring system and we in all settings that the Newton-CD method converges far more quickly than other methods. In the largest system considered ($N = 500$) with $\lambda = 0.1$ (top left), it converges to a

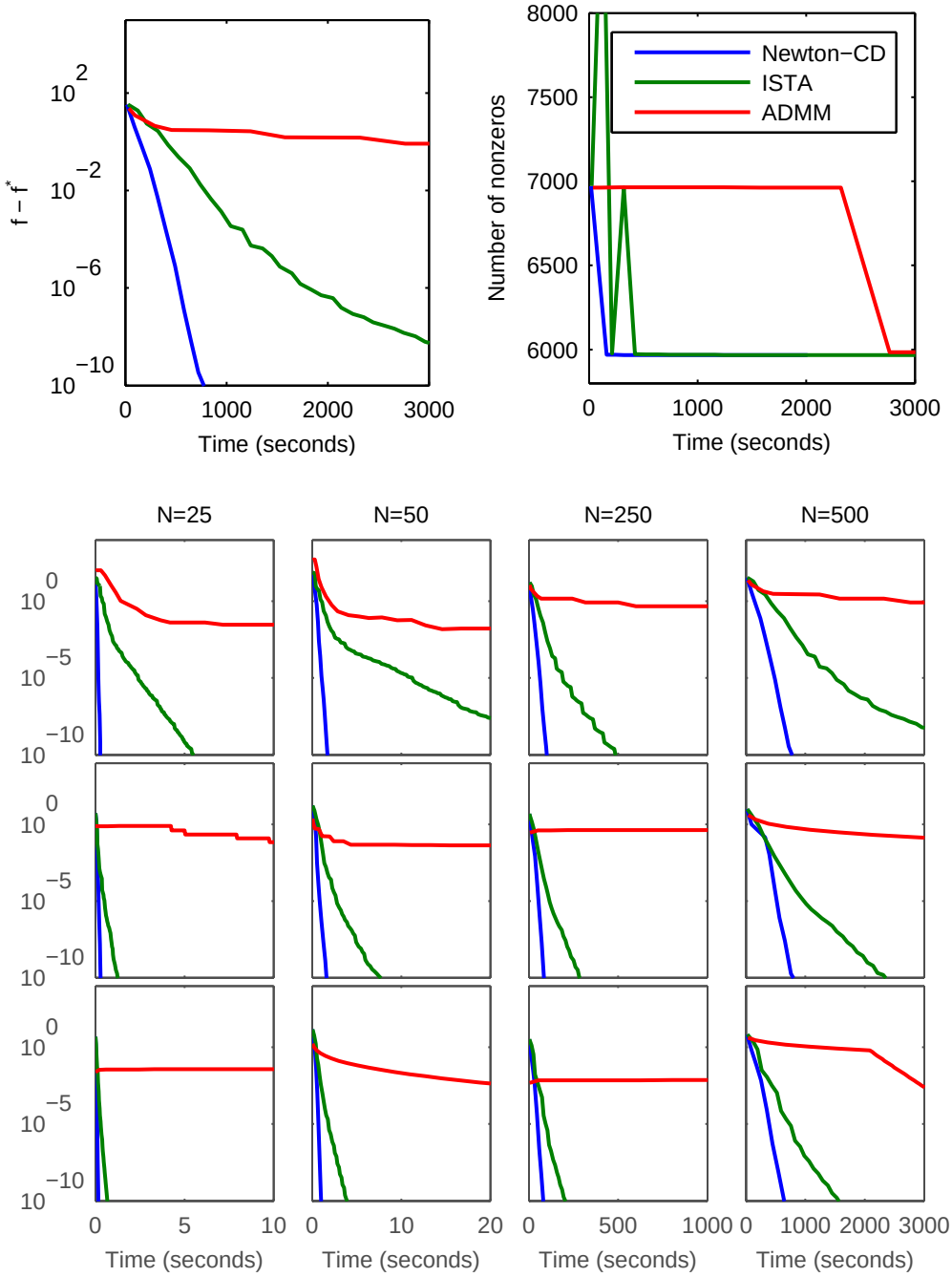

Figure 5.2: Convergence of algorithms on mass-spring system with $N = 500$ and $\lambda = 10$ (top left); the sparsity found by each algorithm for the same system (top right); and across many settings with one column per example and rows corresponding to different settings of λ with $\lambda_1 = [10, 10, 1, 1]$, $\lambda_2 = [1, 1, 0.1, 0.1]$ and $\lambda_3 = [0.1, 0.1, 0.01, 0.01]$ (bottom).

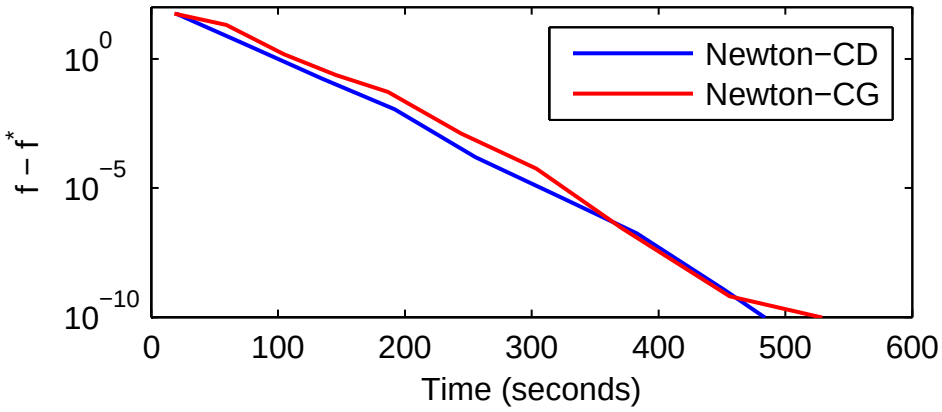

Figure 5.3: Convergence of Newton methods on the polishing step for the mass-spring system with $N = 500$ and $\lambda = 100$.

solution accurate to 10^{-8} in less than 11 minutes whereas ADMM has not reached an accuracy of 10^{-1} after over two hours. In addition, the sparsity pattern in the intermediate solutions do not typically correspond to that of the ℓ_1 solution, as can be seen in Figure 5.2 (top right). For smaller examples, ISTA is competitive but as the size of the system grows, the many iterations that it requires to converge become more expensive, a behavior that is highlighted in the convergence for the $N = 500$ system (rightmost column). Finally, we note that Newton-CD performs especially well on λ corresponding to sparse solutions (top row) due to the active set method exploiting sparsity in the solution.

In addition to solving the ℓ_1 problem, the Newton-CD method can also be used for the polishing step of finding the optimal controller with a fixed sparsity structure. In Figure 5.3, we compare Newton-CD to the conjugate gradient approach of [75] which can be seen as a Newton-Lasso method for the special case of Λ with entries 0 or ∞. Here we see that performance on the polishing step is comparable with both methods converging quickly and using the same number of outer loop iterations. We note that the conjugate gradient approach could also be extended to work for general Λ by using an orthant-based approach (for example, see [93]), but we do not pursue that direction in this work.

5.3.2 Wide-area control in power systems

Following [37], which applied the sparse optimal control framework and the ADMM algorithm to this same problem, our next examples consider the task of controlling inter-area oscillations in a power network via wide-area control. These examples highlight the computational benefits of our algorithmic approach even more so than the synthetic examples above.

To briefly introduce the domain ([37] explains the overall setup in more detail), we are concerned here with the problem of frequency regulation in a AC transmission grid. We employ a linearized approximation where for each generation the system state consists of the power angle θ_i, the mismatch between the rotational velocity and the reference rotational velocity $\omega_i - \omega^{\text{ref}}$, as well as a number of additional states x_i characterizing the exciters, governors, and/or power

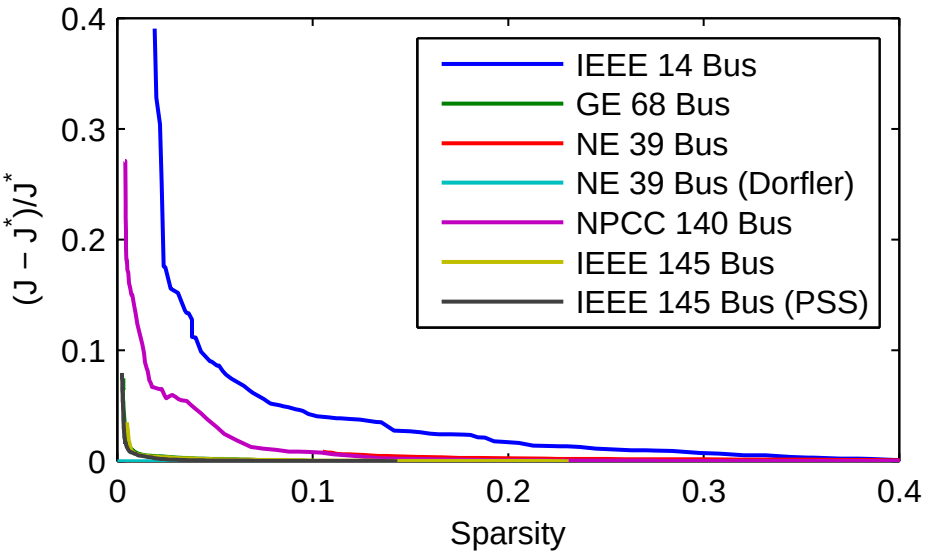

Figure 5.4: Comparison of performance to LQR solution for varying levels of sparsity on wide-area control in power networks.

system stabilizer (PSS) control loops at each generator (typically operating on much faster time scale). The system dynamics can be written generically as

$$\begin{aligned} \dot{\theta} &= \omega - \omega^{\text{ref}} \\ \dot{\omega} &= (Y_{GG} - Y_{GL}Y_{LL}^{-1}Y_{LG})\theta + f(x) \end{aligned} \tag{5.35}$$

where Y is an approximate DC power flow susceptance matrix; G and L are the generator and load nodes; and $f(x)$ denotes the local control dynamics. Importantly, there is a coupling between the generators induced by the network dynamics, which can create oscillatory modes that cannot easily be stabilized by local control alone. The control actions available to the system effectively involve setting the operating points for the inner loops of the power system stabilizers.

The examples we use here are all drawn from the Power Systems Toolbox, in particular the MathNetEig package, which provides a set of routines for describing power networks, generators, exciters and power system stabilizers, potentially at each generator node, and also has routines for analytically deriving the resulting linearized systems. We evaluate our approach on all the larger examples included with this toolbox, as well as the New England 39 bus system used in [37] (which is similar to the 39 bus system included in the power system toolbox, but which includes power system stabilizers at 9 of the 10 generators, limits the type of external control applied to each PSS, and which allows for no control at one of the generators). To create a somewhat larger system than any of those included in the toolbox, we also modify the PST 50 machine system to include power system stabilizers at each node, resulting in a $n = 500$ state system to regulate with our sparse control algorithm.

As in the previous example, we begin by considering the sparsity/performance trade-off by varying the regularization parameter λ, shown in Figure 5.4. Here we see that for several powers systems under consideration, near optimal performance is achieved by an extremely sparse

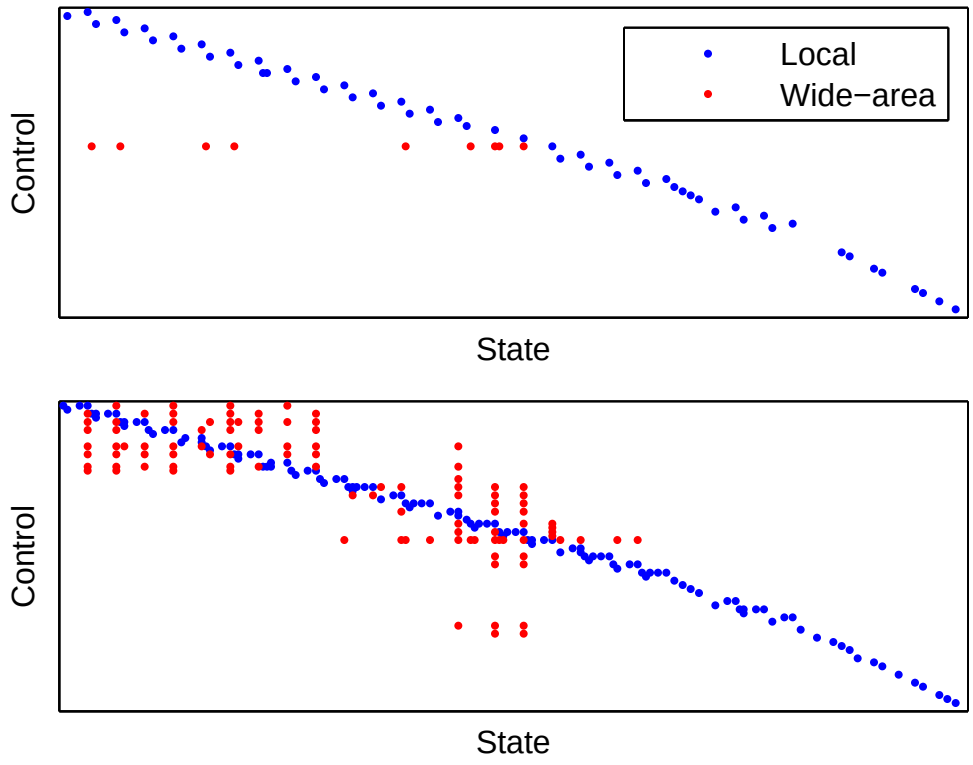

Figure 5.5: Sparsity patterns for wide-area control in the NPCC 140 Bus power system: the sparsest stable solution found (top) and the sparsest solution achieving performance within 10% of optimal (bottom)

.

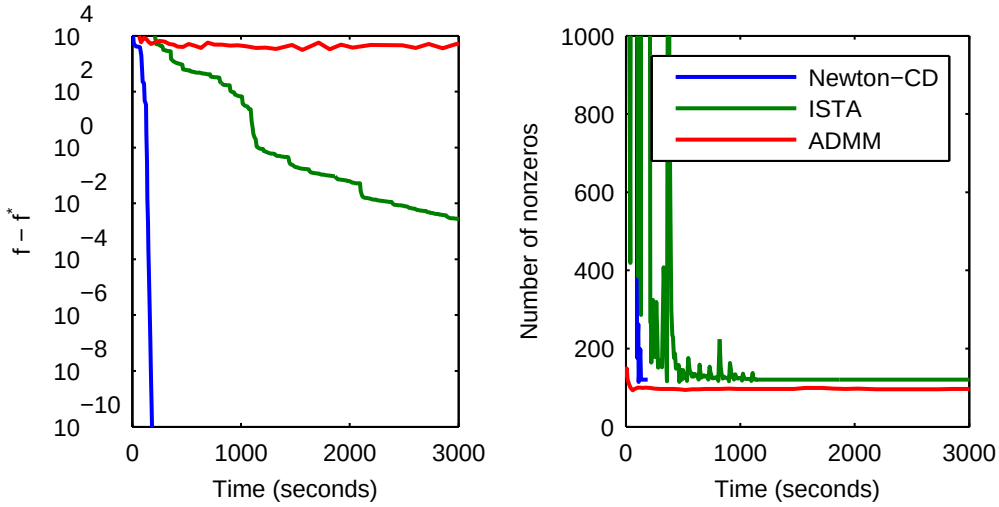

Figure 5.6: Convergence of algorithms on IEEE 145 Bus (PSS) wide-area control example with $\lambda = 100$ (left) and the number of nonzeros in the intermediate solutions (right).

controller depending almost exclusively on local information. As in the mass-spring system, in addition to Λ with uniform weights, we also used a structured Λ to find local controllers; here were able to find stable local control laws for every example with the exception of PST 48. For this power system, we show the sparsity pattern of the sparsest stable controller in Figure 5.5 along with that of the controller achieving performance within 10% of the full LQR optimum. Finally, we note that in general the larger power systems admit controllers with relatively more sparsity as was the case in the mass-spring system.

Computationally, we consider the convergence on the largest power system example in Figure 5.6 and observe a dramatic difference between Newton-CD and previous algorithms: Newton-CD has converged to an accuracy better than 10^{-8} in less than 173 seconds while ADMM is not within 10^3 after over an hour. In addition, the sparsity pattern of the intermediate solutions found by ADMM is significantly different than that of Newton-CD and ISTA which have converged to the ℓ_1 regularized solution with much higher accuracy. In Figure 5.7 we consider convergence across all power systems with three values of λ chosen such that the resulting controllers have performance within 10%, 1% and 0.1% of LQR. Here we see similar results as in the large system with Newton-CD converging faster across all examples and choices of λ and the differences being orders of magnitude in many cases. We note that algorithm benefits significantly from a high level of sparsity for these choices of λ since for most power systems considered, the controller achieving performance within 0.1% of LQR is still quite sparse.

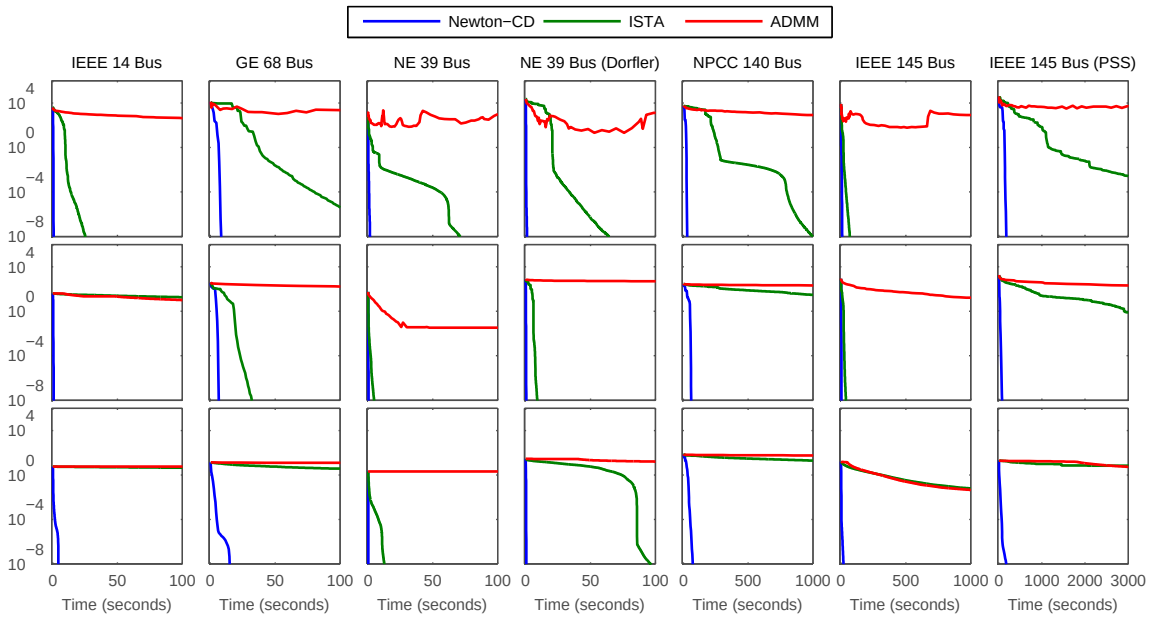

Figure 5.7: Convergence of algorithms on wide-area control across all power systems with three choices of λ corresponding to performance within 10%, 1% and 0.1% of LQR. Columns correspond to power systems and rows correspond to different choices of λ with largest on top.

Chapter 6

The Group Fused Lasso

Given a multivariate signal $y_1, y_2, \ldots, y_T$, with $y_t \in \mathbb{R}^n$, the (weighted) group fused lasso (GFL) estimator [5, 15] attempts to find a roughly "piecewise-constant" approximation to this signal. It determines this approximation by solving the optimization problem

$$\text{minimize} \quad \frac{1}{2}\sum_{t=1}^{T} w_i \|x_t - y_t\|_2^2 + \sum_{t=1}^{T-1} \lambda_t \|x_t - x_{t+1}\|_2 \tag{6.1}$$

where $x_1, x_2, \ldots, x_T$ are the optimization variables, $w \in \mathbb{R}_+^T$ are weights for each time point, $\lambda \in \mathbb{R}_+^{T-1}$ are regularization parameters, and $\|\cdot\|_2$ denotes the Euclidean norm. Intuitively, the ℓ_2-norm on the *difference* between consecutive points encourages sparsity in this difference: each $x_t - x_{t+1}$ will typically be either full or identically zero at the solution, i.e., the signal x will be approximately piecewise-constant. This approach generalizes the 1D total variation norm [10, 124], which considers only univariate signals. Owing to the piecewise-constant nature of the approximate signals formed by the group fused lasso, the approach has found applications in signal compression, multiple change-point detection, and total variation denoising. Though several algorithms have been proposed to solve (6.1), to the best of our knowledge these have involved, at their foundation, first-order methods such as projected gradient, block coordinate descent, or splitting methods. Although such algorithms can sometimes obtain reasonable performance, they often fail to quickly find accurate solutions, especially when one wants to solve (6.1) to high precision as a proximal operator (e.g. as in Section 2.2).

In this chapter, we develop a fast algorithm for solving the optimization problem (6.1), based upon a projected Newton approach. Our method can solve group fused lasso problems to high numerical precision, often several orders of magnitude faster than existing state-of-the-art approaches. At its heart, our method involves dualizing the optimization problem (6.1) *twice*, in a particular manner, to eliminate the non-differentiable ℓ_2-norm and replace it by simple nonnegativity constraints; we solve the reformulated problem to high accuracy via a projected Newton approach. In order to fully exploit the sparsity of large-scale instances, we combine the above ideas with a primal active-set method that iteratively solves reduced-size problems to find the final set of non-zero differences for the original GFL problem.

Although our fast fused group lasso method is valuable in its own right, its real power comes when used as a proximal subroutine in a more complex algorithm, an operation that often needs

to be solved thousands of times. With this motivation in mind, we apply our approach to two applications: segmenting linear regression models, and color total variation image denoising. We demonstrate the power of our approach in experiments with real and synthetic data, both for the basic group fused lasso and these applications, and show substantial improvement over the state of the art.

6.1 A fast Newton method for the GFL

We begin by adopting slightly more compact notation, and rewrite (6.1) (the *primal* problem) as

$$\text{minimize} \;\; (1/2)\|(X-Y)W^{1/2}\|_F^2 + \|XD\Lambda\|_{1,2} \tag{P}$$

where $X, Y \in \mathbb{R}^{n\times T}$ denote the matrices

$$X = \left[\begin{array}{ccc} x_1 & \cdots & x_T \end{array}\right], \;\; Y = \left[\begin{array}{ccc} y_1 & \cdots & y_T \end{array}\right]; \tag{6.2}$$

$W := \mathrm{diag}(w)$ and $\Lambda := \mathrm{diag}(\lambda)$; $\|\cdot\|_F$ denotes the Frobenius norm; $\|\cdot\|_{1,2}$ denotes the mixed $\ell_{1,2}$-norm

$$\|A\|_{1,2} := \sum\nolimits_i \|a_i\|_2, \tag{6.3}$$

where a_i is the ith column of A; and $D \in \mathbb{R}^{T,T-1}$ denotes the first-order differencing operator

$$D = \left[\begin{array}{cccc} 1 & 0 & 0 & \cdots \\ -1 & 1 & 0 & \cdots \\ 0 & -1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}\right] \tag{6.4}$$

so that XD takes the difference of the columns of X.

6.1.1 Dual problems

To solve (P), it is useful to look at its dual and (for our algorithm) a modified dual of this dual. To derive these problems, we transform (P) slightly by introducing the constraint $V = XD$, and corresponding dual variables $U \in \mathbb{R}^{n\times T-1}$. The Lagrangian is then given by

$$\mathcal{L}_P(X,U,V) := (1/2)\|(X-Y)W^{1/2}\|_F^2 + \|V\Lambda\|_{1,2} + \mathrm{tr}\, U^T(V-XD). \tag{6.5}$$

Minimizing (6.5) analytically over X and V gives

$$X^\star = Y - UD^TW^{-1}, \;\; V^\star = 0 \text{ iff } \|u_t\|_2 \le \lambda_t \tag{6.6}$$

where u_t is the t-th column of U; this leads to the dual

$$\begin{array}{ll} \text{maximize} & -(1/2)\|UD^TW^{-1/2}\|_F^2 + \mathrm{tr}\, UD^TY^T \\ \text{subject to} & \|u_t\|_2 \le \lambda_t, \qquad t = 1,\ldots,T-1. \end{array} \tag{D}$$

Indeed, several past algorithmic approaches have solved (D) directly using projected gradient methods, see e.g., [5].

The basis of our algorithm is to form the dual of (D), but in a manner that leads to a different problem than the original primal. In particular, noting that the constraint $\|u_t\|_2 \leq \lambda_t$ is equivalent to the constraint that $\|u_t\|_2^2 \leq \lambda_t^2$, we can remove the non-differentiable ℓ_2-norm, and form the Lagrangian

$$\mathcal{L}_D(U, z) = -(1/2)\|UD^T W^{-1/2}\|_F^2 + \operatorname{tr} UD^T Y^T + \sum_{t=1}^{T-1} z_t(\|u_t\|_2^2 - \lambda_t^2). \tag{6.7}$$

Minimizing over U analytically yields

$$U^\star = YD(D^T W^{-1} D + Z)^{-1}, \tag{6.8}$$

where $Z := \operatorname{diag}(z)$, and leads to the dual problem (the dual of the dual of (P))

$$\begin{array}{ll} \text{minimize} & (1/2)YD(D^T W^{-1} D + Z)^{-1} D^T Y^T + (1/2)(\lambda^2)^T z \\ \text{subject to} & z \geq 0 \end{array} \tag{DD}$$

where λ^2 denotes squaring λ elementwise. This procedure, taking the dual of the dual of the original optimization problem, has transformed the original, nonsmooth problem into a smooth optimization problem subject to a nonnegativity constraint, a setting for which there are several efficient algorithms. Although (DD) is not easily solved via a standard form semidefinite program—it involves a matrix fractional term, for which the standard semidefinite programming form is computationally unattractive—it can be solved efficiently by a number of methods for smooth, bound-constrained optimization. However, as we will see below, the Hessian for this problem is typically poorly conditioned, so the choice of algorithm for minimizing (DD) has a large impact in practice. Furthermore, because the z dual variables are non-zero only for the change points of the original X variables, we expect that for many regimes we will have very few non-zero z values. These points motivate the use of projected Newton methods [14], which perform Newton updates on the variables not bound ($z \neq 0$).

6.1.2 A projected Newton method for (DD)

Denote the objective of (DD) as $f(z)$; the gradient and Hessian of f are given by

$$\begin{aligned} \nabla f(z) &= -(1/2)(U^2)^T 1 + (1/2)\lambda^2, \\ \nabla^2 f(z) &= U^T U \circ (D^T W^{-1} D + Z)^{-1}, \end{aligned} \tag{6.9}$$

where as above $U = YD(D^T W^{-1} D + Z)^{-1}$, U^2 denotes elementwise squaring of U, and $\circ$ denotes the elementwise (Hadamard) product. The projected Newton method proceeds as follows: at each iteration, we construct the set of *bound* variables

$$\mathcal{I} := \{i : z_i = 0 \text{ and } (\nabla_z f(z))_i > 0\}. \tag{6.10}$$

We then perform a Newton update only on those variables that are *not* bound ($\bar{\mathcal{I}}$, referred to as the *free set*), and project back onto the feasible set

$$z_{\bar{\mathcal{I}}} \leftarrow \left[z_{\bar{\mathcal{I}}} - \alpha(\nabla_z^2 f(z))^{-1}_{\bar{\mathcal{I}},\bar{\mathcal{I}}}(\nabla_z f(z))_{\bar{\mathcal{I}}}\right]_+, \tag{6.11}$$

where α is a step size (chosen by backtracking, interpolation, or other line search), and $[\cdot]_+$ denotes projection onto the non-negative orthant. The full method is shown in Algorithm 6.1. Although the projected Newton method is conceptually simple, it involves inverting several (possibly $T \times T$ matrices), which is impractical if these were to be performed with general matrix operations. Fortunately, there is a great amount of structure that can be exploited in this problem.

Algorithm 6.1: Projected Newton for GFL

---

input signal $Y \in \mathbb{R}^{n\times T}$; weights $w \in \mathbb{R}^T_+$; regularization parameters $\lambda \in \mathbb{R}^{T-1}_+$; tolerance ϵ
output: optimized signal $X \in \mathbb{R}^{n\times T}$
initialization: $z \leftarrow 0$
repeat

1. Form dual variables and gradient

$$U \leftarrow YD(DW^{-1}D + Z)^{-1}$$
$$\nabla_z f(z) \leftarrow -(1/2)(U^2)^T 1 + (1/2)\lambda^2$$

2. Compute active constraints

$$\mathcal{I} \leftarrow \{i : z_i = 0 \text{ and } (\nabla_z f(z))_i > 0\}$$

3. Compute reduced Hessian and Newton direction

$$H \leftarrow U^T_{\bar{\mathcal{I}}} U_{\bar{\mathcal{I}}} \circ (D^T W^{-1} D + Z)^{-1}_{\bar{\mathcal{I}},\bar{\mathcal{I}}}$$
$$\Delta z_{\bar{\mathcal{I}}} \leftarrow -H^{-1}(\nabla_z f(z))_{\bar{\mathcal{I}}}$$

4. Update variables

$$z_{\bar{\mathcal{I}}} \leftarrow [z_{\bar{\mathcal{I}}} + \alpha \Delta z_{\bar{\mathcal{I}}}]_+$$

where α is chosen by line search
until $\|(\nabla_z f(z))_{\bar{\mathcal{I}}}\|_2 \le \epsilon$

Efficiently solving $YD(D^TW^{-1}D + Z)^{-1}$. One key operation for the weighted GFL problem is to solve linear systems of the form $D^TW^{-1}D + Z$, where W and Z are diagonal. Fortunately,

the first matrix is highly structured: it is a symmetric tridiagonal matrix

$$D^T W^{-1} D = \begin{bmatrix} \frac{1}{w_1} + \frac{1}{w_2} & -\frac{1}{w_2} & 0 & \cdots \\ -\frac{1}{w_2} & \frac{1}{w_2} + \frac{1}{w_3} & -\frac{1}{w_3} & \cdots \\ 0 & -\frac{1}{w_3} & \frac{1}{w_3} + \frac{1}{w_4} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{6.12}$$

and adding Z to it only affects the diagonal. LAPACK has customized routines for solving problems of this form: `dpttrf` (which computes the LDL^T factorization of the matrix) and `dptts2` (which computes the solution to $LDL^T X = B$ via backsubstitution). For our work, we modified this latter code slightly to solve systems with the unknown on the left hand size, as is required for our setting; this lends a slight speedup by exploiting the memory locality of column-based matrices. The methods factor $T-1 \times T-1$ matrix in $O(T)$ time, and solve n left hand sides in time $O(Tn)$.

Computing entries of $(D^T W^{-1} D + Z)^{-1}$. The projected Newton method also requires more than just solving equations of the form above: to compute the Hessian, we must actually also compute entries of the inverse $(D^T W^{-1} D + Z)^{-1}$ — we need to compute the entries with rows and columns in $\bar{\mathcal{I}}$. Naively, this would require solving $k = |\bar{\mathcal{I}}|$ left hand sides, corresponding to the unit bases for the entries in $\bar{\mathcal{I}}$; even using the fast solver above, this takes time $O(Tk)$. To speed up this operation, we instead use a fast method for computing the actual entries of the inverse of this tridiagonal, using an approach based upon [129]; this ultimately lets us compute the k^2 entries in $O(k^2)$ time, which can be much faster for small free sets.

Specifically, let $a \in \mathbb{R}^{T-1}$ and $b \in \mathbb{R}^{T-2}$ denote the diagonal and the negative off-diagonal entries of $D^T W^{-1} D + Z$ respectively (that is, $a_i = \frac{1}{w_i} + \frac{1}{w_{i+1}} + z_i$ and $b_i = \frac{1}{w_{i+1}}$), we can compute individual entries of $(D^T W^{-1} D + Z)^{-1}$ as follows (the following adapts the algorithm in [129], but has enough simplifications for our case that we state it explicitly here). Define $\theta, \phi \in \mathbb{R}^T$ via the recursions

$$\begin{aligned} \theta_{i+1} &= a_i \theta_i - b_{i-1}^2 \theta_{i-1},\ i = 2, \ldots, T-1 \\ \theta_1 &= 1,\ \theta_2 = a_1, \\ \phi_i &= a_i \phi_{i+1} - b_i^2 \phi_{i+2},\ i = T-2, \ldots, 1 \\ \phi_T &= 1,\ \phi_{T-1} = a_{T-1}. \end{aligned} \tag{6.13}$$

Then, the (i, j) entry of $(D^T W^{-1} D + Z)^{-1}$ for $j \leq i$ is given by

$$(D^T W^{-1} D + Z)^{-1}_{ij} = \frac{1}{\theta_T} \Big(\prod_{k=i}^{j-1} b_i \Big) \theta_i \phi_{j+1}. \tag{6.14}$$

Finally, we can compute all the needed running products $\prod_{k=i}^{j-1} b_i$ by computing a single cumulative sum of the logs of the b_i terms $c_i = \sum_{j=1}^{i} \log b_i$ and then using the equality $\prod_{k=i}^{j-1} b_i = \exp(c_j - c_i)$.

6.1.3 A primal active set approach

Using the two optimizations mentioned above, the projected Newton method can very quickly find a solution accurate to numerical precision for medium sized problems (T and n on the order of thousands). However, for problems with substantially larger T, which are precisely those we are most interested in for many GFL applications, the approach above begins to break down. There are two reasons for this: 1) The size of the free set $k = |\mathcal{I}|$, though often small at the final solution, can be significantly larger at intermediate iterations; since the Newton method ultimately does involve an $O(k^3)$ time to invert the Hessian restricted to the free set, this can quickly render the algorithm impractical. 2) Even with small free sets, the basic $O(Tn)$ cost required for a single pass over the data at each Newton iteration starts to dominate, especially since a significant number of iterations to find the correct free set may be required (only after finding the correct free set does one obtain quadratic convergence rates).

To overcome these problems, we consider a further layer to the algorithm, which wraps our fast projected Newton solver inside a primal active-set method. The basic intuition is that, at the optimal solution to the original GFL problem, there will typically be very few change points in the solution $X^\star$ (these correspond exactly to those z variables that are non-zero). If we knew these changes points ahead of time, we could solve a substantially reduced (weighted) GFL problem that was equivalent to the original problem. Specifically, let $\mathcal{J} \subseteq \{1, \ldots, T-1\}$ denote the optimal set of change point locations for the primal problem. By the relationship of dual problems, this will be identical to the set of free variables $\bar{\mathcal{I}}$ at the optimal solution, but since we treat these differently in the algorithmic design we use different notation. Then the original problem

$$\text{minimize} \quad \|(X - Y)W^{1/2}\|_F^2 + \|XD\Lambda\|_{1,2}, \tag{6.15}$$

where $X \in \mathbb{R}^{n \times T}$, is equivalent to the reduced problem

$$\text{minimize} \quad \|(X' - Y')W'^{1/2}\|_F^2 + \|X'D'\Lambda_{\mathcal{J},\mathcal{J}}\|_{1,2} \tag{6.16}$$

with optimization variable $X' \in \mathbb{R}^{n \times k+1}$ for $k = |\mathcal{J}|$, where $D' \in \mathbb{R}^{k+1 \times k}$ denotes the same first-order differences matrix but now over only $k+1$-sized vectors, and where Y' and $W' = \operatorname{diag}(w')$ are defined by

$$w'_i = \sum_{j \in \mathcal{J}'_i} w_j, \quad y'_i = \frac{1}{w'_i} \sum_{j \in \mathcal{J}'_i} w_j y_j, \tag{6.17}$$

where we define $\mathcal{J}'_i = \{\mathcal{J}_{i-1} + 1, \ldots, \mathcal{J}_i\}$ for $i = 1, \ldots, k+1$ (i.e., $\mathcal{J}'_i$ denotes the list of indices within the ith segment, there being $k+1$ segments for k change points). Furthermore, all these terms can be computed in time $O(nk)$ via cumulative sums similar to the cumulative sum used for b above (which take $O(Tn)$ to compute once, but which thereafter only require $O(kn)$ to form the reduced problem).

To see this equivalence, note first that since X only changes at the points $|\mathcal{J}|$, it immediately holds that $\|XD\Lambda\|_{1,2} = \|X'D'\Lambda_{\mathcal{J},\mathcal{J}}\|_{1,2}$. To show that the other term in the objective is also

equivalent, we have that

$$\begin{aligned}
&\|(X-Y)W^{1/2}\|_F^2 \\
&= \sum_{i=1}^{k+1} \|(x_i' 1^T - Y_{\mathcal{J}_i'}) W^{1/2}_{\mathcal{J}_i',\mathcal{J}_i'}\|_F^2 \\
&= \sum_{i=1}^{k+1} \left((w_{\mathcal{J}'}^T 1) x'^T_i x_i' - 2 x'^T Y_{\mathcal{J}_i'} w_{\mathcal{J}_i'} + \|Y_{\mathcal{J}_i'} W^{1/2}_{\mathcal{J}_i',\mathcal{J}_i'}\|_F^2 \right) \\
&= \sum_{i=1}^{k+1} w_i' \|x_i' - y_i'\|_2^2 + c.
\end{aligned}$$

This equivalence motivates a primal active set method where we iteratively guess the active set $\mathcal{J}$ (with some fixed limit on its allowable size), use the projected Newton algorithm to solve the reduced problem, and then use the updated solution to re-estimate the active set. This is essentially equivalent to a common "block pivoting" strategy for non-negative least squares [102] or ℓ_1 methods [73], and has been shown to be very efficient in practice [67]. The full algorithm, which we refer to as Active Set Projected Newton (ASPN, pronounced "aspen"), is shown in Algorithm 6.2. In total, the algorithm is extremely competitive compared to past approaches to GFL, as we show in Section 6.3, often outperforming the existing state of the art by orders of magnitude.

6.2 Applications

Although the ASPN algorithm for the group fused lasso is a useful algorithm in its own right, part of the appeal of a fast solver for this type of problem is the possibility of using it as a "subroutine" within solvers for more complex problems. In this section we derive such algorithms for two instances: segmentation of time-varying linear regression models and multi-channel total variance image denoising. Both models have been considered in the literature previously, and the method presented here offers a way of solving these optimization problems to a relatively high degree of accuracy using simple methods.

6.2.1 Linear model segmentation

In this setting, we observe a sequence of input/output pairs ($a_t \in \mathbb{R}^n$, $y_t \in \mathbb{R}$) over time and the goal is to find model parameters x_t such that $y_t \approx a_t^T x_t$ (it is more common to denote the input itself as x_t and model parameters θ_t, but the notation here is more in keeping with the rest of this chapter). Naturally, if x_t is allowed to vary arbitrarily, we can always find (an infinite number of) x_t's that fit the output perfectly, but if we constrain the sum of norms $\|x_t - x_{t-1}\|_2$, then we will instead look for piecewise constant segments in the parameter space; this model was apparently first proposed in [92].

This model may be cast as the optimization problem

$$\text{minimize} \quad \|A \operatorname{vec} X - y\|_2^2 + \|XD\Lambda\|_{1,2}, \tag{6.18}$$

Algorithm 6.2: Active Set Projected Newton (ASPN)

input signal $Y \in \mathbb{R}^{n \times T}$; weights $w \in \mathbb{R}_+^T$; regularization parameters $\lambda \in \mathbb{R}_+^{T-1}$; maximum active set size $k_{\max}$; tolerance ϵ
output: optimized signal $X \in \mathbb{R}^{n \times T}$
initialization: $z \leftarrow 0$
repeat

1. Form dual variables and gradient

$$U \leftarrow YD(DW^{-1}D + Z)^{-1}$$
$$\nabla_z f(z) \leftarrow -\frac{1}{2}(U^2)^T 1 + \frac{1}{2}\lambda^2$$

2. Compute active set, containing all non-zero z_i's and additional element with negative gradients, up to size $k_{\max}$

$$\mathcal{J}^0 \leftarrow \{i : z_i > 0\}$$
$$\mathcal{J}^1 \leftarrow \{i : z_i = 0, \nabla_z f(z) < 0\}$$
$$\mathcal{J} \leftarrow \mathcal{J}^0 \cup \mathcal{J}^1_{1:k_{\max}-|\mathcal{J}^0|}$$

3. Form reduced problem (Y', w') for $\mathcal{J}$ using (6.17) and solve using projected Newton

$$z_{\mathcal{J}} \leftarrow \text{Projected-Newton}(Y', w', \lambda_{\mathcal{J}})$$

until $\|(\nabla_z f(z))_{\mathcal{J}}\|_2 \leq \epsilon$

where $X \in \mathbb{R}^{n \times T}$ is the same optimization variable as previously, $y \in \mathbb{R}^T$ denotes the vector of outputs, vec denotes the vectorization of a matrix (stacking its columns into a single column vector), and $A \in \mathbb{R}^{T \times Tn}$ is the block diagonal matrix

$$A = \begin{bmatrix} a_1^T & 0 & 0 & \cdots \\ 0 & a_2^T & 0 & \cdots \\ 0 & 0 & a_3^T & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{6.19}$$

While this problem looks very similar to the ordinary GFL setting, the introduction of the additional matrix A renders it substantially more complex. While it is possible to adapt the Newton methods above to solve the problem directly, much of the special problem structure is lost, and it requires, for examples, forming $Tn \times Tn$ block tridiagonal matrices, which is substantially more computationally intensive, especially for large n (the methods scale like $O(n^3)$). While optimization may still be possible with such approaches, we instead adopt a different approach that builds on the alternating direction method of multipliers (see Section 2.2.3).

The "standard" ADMM algorithm. The simplest way to apply ADMM to (6.18), considered for the pure group fused lasso e.g., in [132], is to introduce variables $Z = XD$, and formulate the problem as

$$\begin{array}{ll} \text{minimize} & \|A \operatorname{vec} X - Y\|_2^2 + \|Z\Lambda\|_{1,2} \\ \text{subject to} & XD = Z. \end{array} \tag{6.20}$$

After some derivations, this leads to the updates

$$\begin{aligned} X^{k+1} &\leftarrow \operatorname*{argmin}_X \|A \operatorname{vec} X - y\|_2^2 + \tfrac{\rho}{2}\|XD - Z^k + U^k\|_F^2 \\ Z^{k+1} &\leftarrow \operatorname*{argmin}_Z \|Z\Lambda\|_{1,2} + \tfrac{\rho}{2}\|X^{k+1}D - Z + U^k\|_F^2 \\ U^{k+1} &\leftarrow U^k + X^{k+1}D - Z^{k+1}, \end{aligned} \tag{6.21}$$

where ρ acts effectively like a stepsize for the problem. This set of updates is particularly appealing because minimization over X and Z can both be computed in closed form: the minimization over X is unconstrained quadratic optimization, and has the solution

$$\left(A^T A + \rho F^T F\right)^{-1} \left(A^T y + \rho F^T \operatorname{vec}(Z^k - U^k)\right) \tag{6.22}$$

where $F = (D^T \otimes I)$. Furthermore, these updates can be computed very efficiently, since $\left(A^T A + \rho F^T F\right)$ is block tridiagonal, and since this matrix does not change at each iteration, we can precompute its (sparse) Cholesky decomposition once, and use it for all iterations; using these optimizations, the X update takes time $O(Tn^2)$. Similarly, the Z update is a proximal operator that can be solved by *soft thresholding* the columns of $X^{k+1}D + U^k$ (an $O(Tn)$ operation). Although these elements make the algorithm appealing, they hide a subtle issue: the matrix $(A^T A + \rho F^T F X)$ is poorly conditioned (owing to the poor conditioning of $D^T D$), even for large ρ. Because of this, ADMM needs a large number of iterations for converging to a reasonable solution; even if each iteration is quite efficient, the overall algorithm can still be impractical.

ADMM using the GFL proximal operator. Alternatively, we can derive a different ADMM algorithm by considering instead the formulation

$$\begin{array}{ll} \text{minimize} & \|A \operatorname{vec} X - Y\|_2^2 + \|ZD\Lambda\|_{1,2} \\ \text{subject to} & X = Z, \end{array} \tag{6.23}$$

which leads to the iterative updates

$$\begin{aligned} X^{k+1} &\leftarrow \operatorname*{argmin}_X \|A \operatorname{vec} X - y\|_2^2 + \tfrac{\rho}{2}\|X - Z^k + U^k\|_F^2 \\ Z^{k+1} &\leftarrow \operatorname*{argmin}_Z \|ZD\Lambda\|_{1,2} + \tfrac{\rho}{2}\|X^{k+1} - Z + U^k\|_F^2 \\ U^{k+1} &\leftarrow U^k + X^{k+1} - Z^{k+1}. \end{aligned} \tag{6.24}$$

The X update can still be computed in closed form

$$\operatorname{vec} X^{k+1} = \left(A^T A + \rho I\right)^{-1} \left(A^T y + \rho \operatorname{vec}(Z^k - U^k)\right),$$

which is even simpler to compute than in the previous case, since $A^TA + \rho I$ is block diagonal with blocks $a_i a_i^T + \rho I$, which can be solved for in $O(n)$ time; thus the entire X update takes times $O(Tn)$. The downside is that the Z update, of course, can no longer be solved with soft-thresholding. But the Z update here is precisely in the form of the group fused lasso; thus, we can use ASPN directly to perform the Z update. The main advantage here is that the matrix $A^TA + \rho I$ is much better conditioned, which translates into many fewer iterations of ADMM. Indeed, as we show below, this approach can be many orders of magnitude faster than straight ADMM, which is already a very competitive algorithm for solving these problems.

6.2.2 Color total variation denoising

Next, we consider color total variation denoising example. Given an $m \times n$ RGB image represented as a third order tensor, $Y \in \mathbb{R}^{3\times m\times n}$, total variation image denoising [16, 108] attempts to find an approximation $X \in \mathbb{R}^{3\times m\times n}$ such that differences between pixels in X favor being zero. It does this by solving the optimization problem

$$\text{minimize} \;\; (1/2)\|X - Y\|_F^2 + \lambda \sum_{i=1}^{m}\sum_{j=1}^{n-1} \|X_{:,i,j} - X_{:,i,j+1}\|_2 + \lambda \sum_{i=1}^{m-1}\sum_{j=1}^{n} \|X_{:,i,j} - X_{:,i+1,j}\|_2,$$

corresponding to an ℓ_2-norm penalty on the difference between all adjacent pixels, where each pixel $X_{:,i,j}$ is represented as a 3 dimensional vector. We can write this as a sum of $m + n$ group fused lasso problems

$$\text{minimize} \;\; \sum_{i=1}^{m} \left(\|X_{:,i,:} - Y_{:,i,:}\|_F^2 + \lambda \|X_{:,i,:}D\|_{1,2}\right) + \sum_{j=1}^{n} \left(\|X_{:,:,j} - Y_{:,:,j}\|_F^2 + \lambda \|X_{:,:,j}D\|_{1,2}\right),$$

where $X_{:,i,:} \in \mathbb{R}^{3\times n}$ denotes the slice of a single row of the image and $X_{:,:,j} \in \mathbb{R}^{3\times n}$ denotes the slice of a single column.

Unfortunately, this optimization problem cannot be solved directly via the group fused lasso, as the difference penalties on the rows and columns for the same matrix X render the problem quite different from the basic GFL. We can, however, adopt an approach similar to the one above, and create separate variables corresponding to the row and column slices, plus a constraint that they be equal; formally, we solve

$$\begin{aligned} \text{minimize} \;\; & \sum_{i=1}^{m} \left(\|X_{:,i,:} - Y_{:,i,:}\|_F^2 + \lambda \|X_{:,i,:}D\|_{1,2}\right) + \sum_{j=1}^{n} \left(\|Z_{:,:,j} - Y_{:,:,j}\|_F^2 + \lambda \|Z_{:,:,j}D\|_{1,2}\right) \\ \text{subject to} \;\; & X = Z. \end{aligned}$$

The major advantage of this approach is that it decomposes the problem into $m + n$ *independent* GFL tasks, plus a meta-algorithm that adjusts each sub-problem to make the rows and columns agree. Several such algorithms are possible, including ADMM; we present here a slightly simpler scheme known as the "proximal Dykstra" method [25], which has been previously applied to

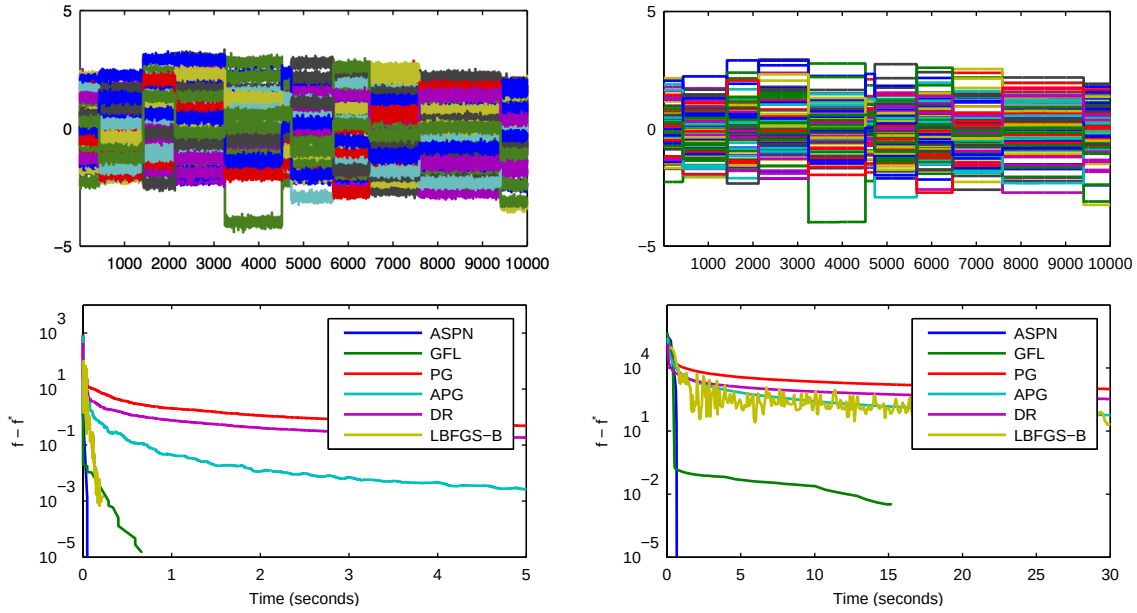

Figure 6.1: Top left: synthetic change point data, with $T = 10000$, $n = 100$, and 10 true change points. Top right: recovered signal. Bottom left: timing results on synthetic problem with $T = 1000$, $n = 10$. Bottom right: timing results on synthetic problem with $T = 10000$, $n = 100$.

the case of (single channel, i.e., black and white) total variation denoising [10]. Starting with $X^0 = Y$, $P^0 = 0$, $Q^0 = 0$, the algorithm iterates as follows:

$$\begin{aligned} Z^{k+1}_{:,:,j} &\leftarrow \mathrm{GFL}(X^k_{:,:,j} + P^k_{:,:,j}, \lambda), \ j = 1, \ldots, n \\ P^{k+1} &\leftarrow P^k + X^k - Z^{k+1} \\ X^{k+1}_{:,i,:} &\leftarrow \mathrm{GFL}(Z^{k+1}_{:,i,:} + Q^k_{:,i,:}, \lambda), \ i = 1, \ldots, m \\ Q^{k+1} &\leftarrow Q^k + Z^{k+1} - X^{k+1}. \end{aligned} \tag{6.25}$$

Typically, very few iterations (on the order of 10) of this outer loop are need to converge to high accuracy. Furthermore, because each of the m or n GFL problems solved in the first and third steps are independent, they can be trivially parallelized.

6.3 Numerical results

We present experimental results for our approaches, both on the basic group fused lasso problem, where we compare to several other potential approaches, and on the two applications of linear model segmentation and color total variation denoising.

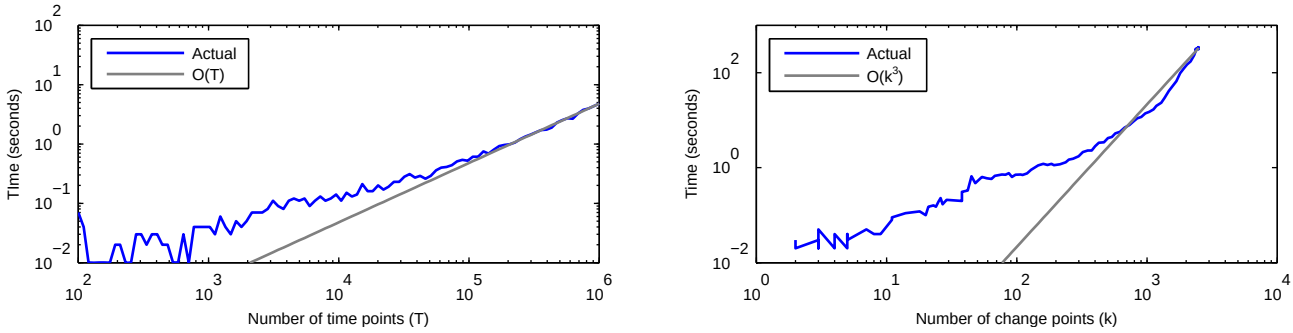

Figure 6.2: Left: timing results vs. number of change points at solution for synthetic problem with $T = 10000$ and $n = 10$. Right: timing results for varying T, $n = 10$, and sparse solution with 10 change points.

6.3.1 Group fused lasso

Here we evaluate the ASPN algorithm versus several alternatives to solving the group fused lasso problem, evaluated on both synthetic and real data. Figure 6.1 shows a synthetic time series with $T = 10,000$, $n = 100$, and 10 discrete change points in the data; the data was generated by uniformly sampling the change points, sampling the mean of each segment from $\mathcal{N}(0, I)$, and then additional Gaussian noise. Figure 6.1 shows the recovered signal using the group fused lasso with $w_t = 1$, $\lambda_t = 20$—we show timing results for this problem as well as a smaller problem with $T = 1000$ and $n = 10$; we compare ASPN to GFLseg [15] (which uses coordinate descent on the primal problem), an accelerated projected gradient on the dual (i.e., the FISTA algorithm) [11], Douglas-Rachford splitting [24] (a generalization of ADMM that performs slightly better here), a projected gradient on the dual [5], and LBFGS-B [22] applied to the dual of the dual. In all cases, ASPN performs as well as (often much better than) the alternatives.

Next, we evaluate how the ASPN algorithm scales as a function of the number of time points T and the number of change points at the solution, k. In Figure 6.2, the first set of experiments shows that when the number of change points at the solution is fixed ($k = 10$), the amount of time required for a highly accurate solution remains small even for large T, agreeing with analysis that shows the number of operations required is $O(T)$. In particular, a solution accurate to 10^{-6} is found in 4.8 seconds on a problem with $T = 10^6$ time points. However, in the next set of experiments, we see that compute time grows rapidly as a function of k due to the $O(k^3)$ operations required to compute the Newton step, suggesting that the proposed method is most appropriate for problems with sparse solutions.

Finally, we evaluate the algorithm on two real time series previously used with the group fused lasso [15], from DNA profiles of bladder and lung cancer sequences. Figure 6.3 shows one of these two series, along with the approximation produced by the group fused lasso. Figure 6.3 shows timing results for the above methods again on this problem: here we observe the same overall behavior, that ASPN typically dominates the other approaches.

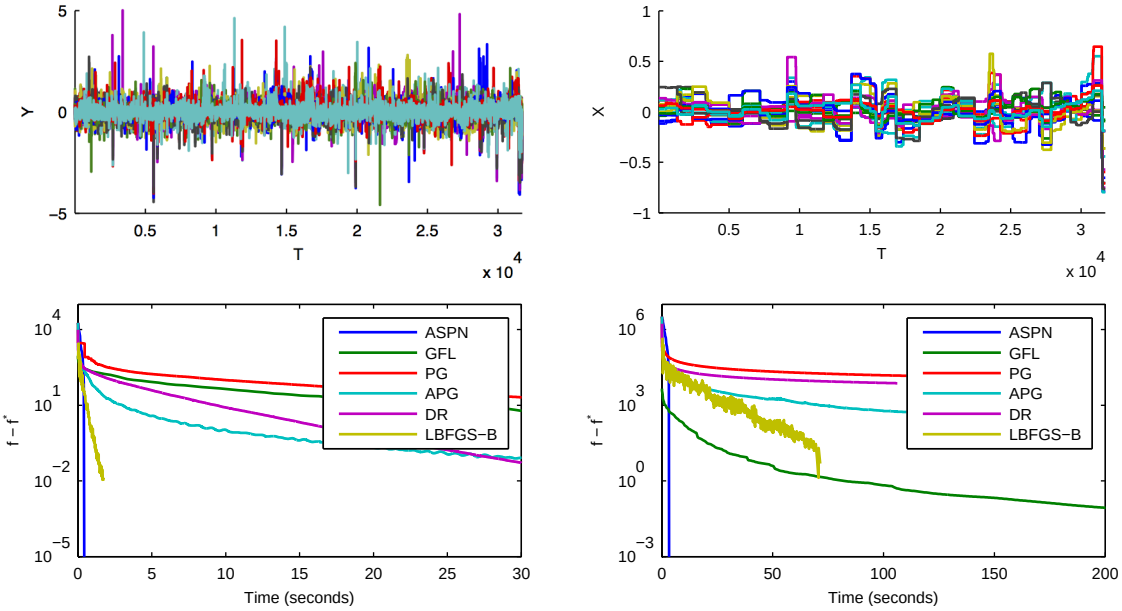

Figure 6.3: Top left: Lung data from [15]. Top right: recovered signal using group fused lasso. Bottom left: Timing results on bladder problem, $T = 2143$, $n = 57$. Bottom right: Timing results on lung problem, $T = 31708$, $n = 18$.

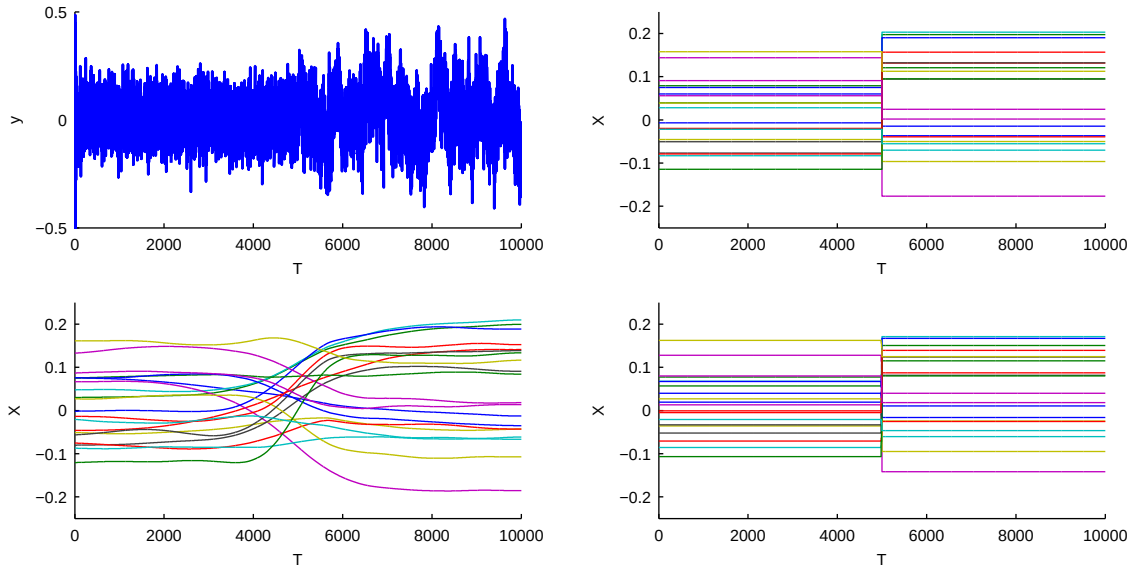

Figure 6.4: Top left: Observed autoregressive signal z_t. Top right: true autoregressive model parameters. Bottom left: Autoregressive parameters recovered with "simple" ADMM algorithm. Bottom right: parameters recovered using alternative ADMM w/ ASPN.

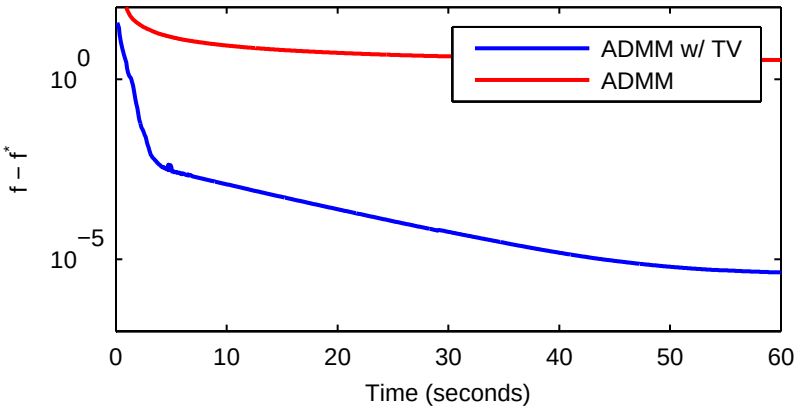

Figure 6.5: Convergence of simple ADMM versus alternative ADMM w/ ASPN

6.3.2 Linear regression segmentation

Here we apply the two different ADMM methods discussed in Section 6.2.1 to the task of segmenting auto-regressive time series models. In particular, we observe some time series $z_1, \ldots, z_T$, and we fit a linear model to this data $z_t \approx a_t^T x_t$ where $a_t = (z_{t-1}, z_{t-2}, \ldots, z_{t-n})$. Figure 6.4 (top) shows an example time series generated by this process, as well as the true underlying model that generated the data (with additional noise). This is the rough setting used in [92], which was the first example we are aware of that uses such regularization techniques within an linear regression framework. Figure 6.4 (bottom) shows the model parameters recovered using the method from Section 6.2.1, which here match the ground truth closely.

Of more importance, though, is the comparison between the two different ADMM approaches. Figure 6.5 shows convergence versus running time and here the "simple" ADMM approach, which encodes the difference operator in the constraints (and thus has simpler updates), converges significantly slower than our alternative. Importantly, the X axis in this figure is measured in time, and we emphasize that even though the "simple" ADMM updates are individually slightly faster (they do not involve GFL subproblems), their overall performance is much poorer. Further, as illustrated in Figure 6.4 (bottom), the "simple" ADMM approach never actually obtains a piecewise constant X except at the optimum, which is never reached in practice.

6.3.3 Color total variation denoising

Finally, as described in Section 6.2.2, we apply the proximal Dykstra algorithm, using ASPN as a fast subroutine, to color image denoising. Figure 6.6 shows a 256x256 image generated by combining various solid-colored shapes, corrupted with per-RGB-component noise of $\mathcal{N}(0, 0.1)$, and then recovered with total variation denoising. There has been enormous work on total variation denoising, and while a full comparison is beyond the scope of this work, ADMM or methods such as those used by the FTVd routines in [144], for instance, are considered to be some of the fastest for this problem. In Figure 6.7, we show the performance of our approach and ADMM versus iteration number, and as expected observe better convergence; for single-core systems,

Figure 6.6: Left: original image. Middle: image corrupted with Gaussian noise. Right: imaged recovered with total variation using proximal Dykstra and ASPN.

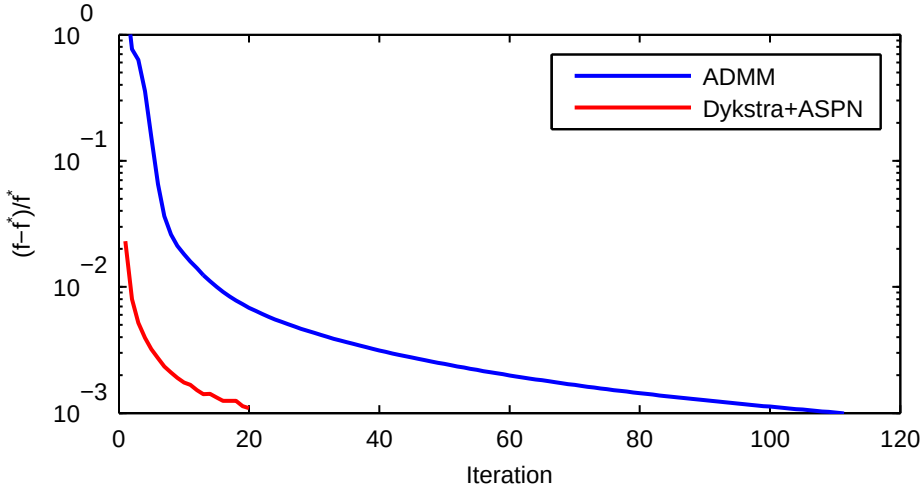

Figure 6.7: Comparison of proximal Dykstra method to ADMM for TV denoising of color image.

ADMM is ultimately a better solution for this problem, since each iteration of ADMM takes about 0.767 seconds in our implementation whereas 512 calls to ASPN take 20.4 seconds. However, the advantage to the ASPN approach is that all these calls can be trivially parallelized for a 256X speedup (the calls are independent and all code is CPU-bound), whereas parallelizing a generic sparse matrix solve, as needed for ADMM-based approaches, is much more challenging and thus per-iteration performance highlights the potential benefits of the ASPN approach.

Part III

Applications in Energy

Introduction to Applications in Energy

In Parts I and II we develop convex methods with wide applicability to many different domains, focusing on convex problem formulations proposed by researchers in statistics, machine learning with applications including computational biology, finance and many others. In many ways, the strength of the convex optimization (as well as machine learning) is that this model-based approach provides a general set of tools that can be applied in many seemingly different applications. At the same time, the goals of a particular application often drive the development of new approaches; our interest in the models and methods studied has been in large part driven by applications in energy systems and specifically the challenge of building the next-generation electrical grid, a highly complex system which is presently facing a number of distinct challenges and opportunities. Chapters 7-9, comprising Part III of this thesis, consider several such challenges, taking a data-driven approach driven by scalable convex optimization methods.

Short-term forecasting is a ubiquitous practice for electrical grid operators who must ensure that the supply and demand for electricity is balanced at all times on the grid. For example, over the past several decades experts have developed highly accurate models for aggregate load forecasting (often on the order of 1-2% relative error) which is critical input into the day-ahead planning process. However, new renewable energy sources such as wind and solar are now being integrated into the grid, requiring new forecasting approaches. In Chapter 7, we propose a probabilistic forecasting model for the joint distribution of future power production over multiple locations and multiple time points. This gives rise to a high-dimensional statistical estimation problem which we solve using the convex optimization methods developed in Chapter 4. The outputs of such a model could be integrated a scenario-based day-ahead dispatch algorithm in order to more effectively integrated renewable sources, improving on the point forecast algorithms employed now.

Beyond forecasting supply and demand at the level of the transmission network, understanding the nature of energy usage at a more fine-grained level poses a number of opportunities and challenges. Recently, utilities have invested heavily in the deployment of smart meters which now report energy usage for millions of homes in the United States. These smart meters record and report whole home energy usage in 15 minute intervals, giving much greater visibility into fine-grained consumption patterns. However, it can be difficult to pinpoint the exact energy end-uses from this data, as there are often times several significant loads behind the meter. In Chapter 8, we develop a optimization-based approach that disaggregates smart meter energy usage into categories of consumption through the use of contextual information such as outdoor temperature data. Taking an optimization approach to this problem provides a rich set of tools (e.g., different loss functions for usage categories assumed to be spiky or smooth), and our methods are able to

scale to the problem requirement of analyzing millions of homes and years worth of data.

Finally, Chapter 9 considers a different framework for developing next-generation grid solutions: microgrids, small networks of power producing and consuming devices. With the advent of cheap distributed generation, microgrids are increasingly attractive as they allow for independent operation, providing resiliency as well as a solution to situations where conventional grids are unavailable. Unlike the conventional grid, microgrid operation presents significant challenges due to low inertia as the amount of generation is often close to the amount of consumption, leading to large transient responses when loads are switched on or off. In this setting, we would like to differentiate between short transient responses and overload conditions when available generation is truly insufficient. We adopt a data-driven approach to this problem, developing a method for predicting microgrid collapse from historical observations; computationally, our approach is variation on the traditional support vector machine and thus the model fits within the general convex programming approach developed in Part I.

Chapter 7

Probabilistic Forecasting of Electricity Generation and Demand

Short-term forecasting is a ubiquitous practice in a wide range of energy systems, including forecasting demand, renewable generation, and electricity pricing. Although it is known that probabilistic forecasts (which give a distribution over possible future outcomes) can improve planning and control, many forecasting systems in practice are just used as "point forecast" tools, as it is challenging to represent high-dimensional non-Gaussian distributions over multiple spatial and temporal points. In this section, we apply the sparse Gaussian conditional random field model discussed in Chapter 4 and extend it to the non-Gaussian case using the copula transform. On a wind power forecasting task, we show that this probabilistic model greatly outperforms other methods on the task of accurately modeling potential distributions of power (as would be necessary in a stochastic dispatch problem, for example).

7.1 Introduction

Forecasting, the task of predicting future time series from past observations, is ubiquitous in energy systems. As well-known examples, electricity system operators routinely forecast upcoming electrical load and use these forecasts in market planning [119, 130]; wind farms forecast future power production when offering bids into these markets [71, 83, 115]; and there is a growing use of forecasting at the micro-scale for coordinating smart grid operations [6]. Despite their ubiquity and the complexity of many forecasting methods, most methods are ultimately employed as "point forecast" strategies; users train a system to output point predictions of upcoming values, typically to minimize a metric such as root mean squared error. However, for many complex control and planning tasks, such point forecasts are severely limited: the processes that make up electrical demand, wind power, etc, are stochastic systems and the notion of a "perfect forecast" is unattainable. Thus, *probabilistic forecasts*, which output a distribution over potential future outcomes instead of a single prediction, are of substantial practical interest. Indeed, studies have demonstrated that in the context of electrical demand and wind power, probabilistic forecasts can offer substantial benefits over point predictions [100].

The challenge of probabilistic forecasts is that it is often very hard to describe the joint dis-

tribution over all predicted values because many variables of interest are highly non-Gaussian and it can be difficult to accurately model correlations in a high-dimensional output space. For this reason, most of the literature on probabilistic forecasting has often made simplifying assumptions, for example using specific forms of Gaussian linear models, such as autoregressive moving average (ARMA) models (e.g. [82]); or only predicting non-Gaussian marginal distributions for single output variables (e.g. [81]). Indeed, past work has explicitly highlighted the challenge of developing models that can capture joint distributions over future values.

7.2 The probabilistic forecasting setting

We consider the following setting: let $z_t \in \mathbb{R}^n$ denote a *vector-valued observation* at time t; for example, the ith element of z_t, denoted $(z_t)_i$ could denote the power output by a particular wind farm at time t, and i could range over a collection of wind farms. We let $w_t \in \mathbb{R}^m$ denote a set of (known) exogenous variables that may affect the evolution of the sequence; for example, w_t may include the current time of day, day of the year, and even external variables such as wind forecasts at upcoming time points. The goal of our forecasting setting is to predict H_f future values given H_p past values and the exogenous variables:

$$\text{Given } z_{t-H_p+1:t}, w_t, \text{ predict } z_{t+1:t+H_f}. \tag{7.1}$$

This setting can be referred to as the *vector autoregressive exogenous* (VARX) setting [99]; however, this terminology is also used to describe a particular form of probabilistic model for the sequence, so we just refer to it generally as a multivariate forecasting problem.

Although predicting a single estimate of future observations from past observations can be very useful in many situations, we often want to understand more broadly the *distribution* of possible future observations given past observations and exogenous factors, denoted as

$$p(z_{t+1:t+H_f} | z_{t+1:t+H_f}, w_t). \tag{7.2}$$

We are particularly interested in the case where individual observations z_t are high-dimensional and we want to predict their evolution over a relatively long time horizon, resulting in a high-dimensional probability distribution. The task is made more challenging by the fact that the observations may not have Gaussian distributions (indeed, in the case of our wind power setting, they typically do not) and by the fact that we may have relatively few past observations upon which to build our high-dimensional model.

7.2.1 Relation to existing settings and models

A common method for handling such settings is a vector autoregressive exogenous model in which we model future observations as a linear combination of past observations, exogenous variables, and a Gaussian noise term

$$z_{t+1} = \sum_{i=0}^{H_p-1} \Theta_i z_{t-i} + \Psi w_t + \epsilon_t \tag{7.3}$$

where $\Theta_i \in \mathbb{R}^{n \times n}$ and $\Psi \in \mathbb{R}^{n \times m}$ are model parameters, and $\epsilon_t \sim \mathcal{N}(0, \Sigma)$ is a zero-mean Gaussian random variable with covariance Σ. If we want to make forecasts over a multiple future time points, we can iteratively apply this model or explicitly build an additional VARX model to predict z_{t+2} from past values.

A common extension to the autoregressive framework is to add a moving sum of noise variables, resulting in the autoregressive moving average (ARMA) family of models (see, e.g., [20]) and in this particular setting, the VARMAX model. Additionally, we may choose Θ such that the overall system has at least one unit root (ARIMA models) or consider only a certain periodic set of past observations (seasonal ARIMA models). These and other approaches have been used extensively in the literature, and while they do impose a joint probabilistic model over future observations, they are very limited in the form of this distribution (multivariate Gaussian with a particular covariance matrix).

In the sequel we will consider forecasting using a model that allows for more general dependencies between predicted variables, can capture non-Gaussian marginal distribution of the variables via a copula transform, and which can be learned from very little data by exploiting sparsity. We focus largely on extensions of the pure autoregressive setting, but the model can also be extended to the ARMA setting by introducing additional latent variables.

7.3 Forecasting with the sparse Gaussian CRF

Here we describe the application of the sparse Gaussian conditional random field (SGCRF, see Chapter 4) to probabilistic forecasting. For simplicity of notation, we refer to the set of all known variables as a single vector $x \in \mathbb{R}^n$, while the unknown variables we are attempting to predict are given by $y \in \mathbb{R}^p$

$$x = \begin{bmatrix} z_t \\ z_{t-1} \\ \vdots \\ z_{t-H_p+1} \\ w_t \end{bmatrix}, \quad y = \begin{bmatrix} z_{t+1} \\ z_{t+2} \\ \vdots \\ z_{t+H_f} \end{bmatrix}. \tag{7.4}$$

Recall, the SGCRF models the distribution $y|x$ as

$$p(y|x; \Theta, \Lambda) \propto \exp\left\{-\frac{1}{2} y^T \Lambda y - x^T \Theta y\right\} \tag{7.5}$$

where $\Lambda \in \mathbb{R}^{p \times p}$ and $\Theta \in \mathbb{R}^{n \times p}$ are the parameters of the model. Critically, we can express models with high correlation between variables even though Λ and Θ are sparse. To see this, note that the model can easily be transformed to mean/covariance form

$$p(y|x) \sim \mathcal{N}(-\Lambda^{-1}\Theta^T x, \Lambda^{-1}) \tag{7.6}$$

but $\Lambda^{-1}\Theta$ and Λ^{-1} are likely dense even when Λ and Θ are sparse. Thus, in the forecasting setting, each element of our prediction $z_{t+1:t+H_f}$ can depend on every element of $z_{t-H_p+1:t}$ and w_t. By exploiting sparsity, we learn the model efficiently from much less data than would be required to estimate the mean and covariance directly.

7.3.1 Non-Gaussian distributions via copula transforms

The above SGCRF is limited in that it can only model the distribution over y as a multivariate Gaussian. To overcome this limitation, we employ a (Gaussian) copula transform [148], a method for converting multivariate Gaussian distributions into multivariate distributions with arbitrary marginal distributions. Previous work has applied the copula transform to extend the sparse Gaussian MRF to non-Gaussian distributions [76] and here we extend this to the SGCRF, forming a model which is well-suited for probabilistic forecasting in a wide variety of energy systems.

Formally, suppose $u \in \mathbb{R}$ is a univariate random variable with cumulative distribution function (CDF) F; when we only have samples of u, we use the empirical CDF

$$\hat{F}(u) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\{u < u_i\}. \tag{7.7}$$

In the case that we expect the variables to come from known distribution (e.g. the Weibull distribution for modeling wind speeds), we could use the analytical CDF of this distribution directly. The copula transform simply converts the sample distribution to a uniform $[0, 1]$ random variable by the CDF F, then applies the inverse normal CDF Φ^{-1} to transform the $[0, 1]$ random variable into a Gaussian random variable. Our algorithm models the variables using a SGCRF in this transformed Gaussian space, and then transforms back to the original distribution by applying the inverse copula transform (the normal CDF Φ followed by the inverse CDF F^{-1}). Importantly, this process isolates the probabilistic assumptions of the SGCRF model, allowing it to better handle non-Gaussian variables such as the distribution over future wind power production.

7.3.2 Final Algorithm

As with all learning methods, using SGCRFs consists of a training stage where we learn the parameters that maximize the model's likelihood on past observations. Then, for a new scenario (denoted $x' \in \mathbb{R}^n$), we use the model to make predictions about the future observations y'. The training stage consists of the following elements:

1. Given data (x_i, y_i), for $i = 1, \ldots, m$ (recall that each x_i consists of H_p past observations and external inputs w_t, and each y_i consists of H_f future observations), first estimate the univariate marginal distributions of each $(y_i)_j$, denoted F_j.
2. Transform each the y_i variables to a variable with marginal Gaussian distributions $\tilde{y}_i$ by applying the elementwise copula transform

$$(\tilde{y}_i)_j = \Phi^{-1}(F_j((y_i)_j)) \tag{7.8}$$

3. Train a SGCRF model (i.e., estimate the Θ and Λ parameters) on $(x_i, \hat{y}_i)$, $i = 1, \ldots, m$.

With a model, we can perform any of the following tasks:

- **Compute the most likely output:** Compute the mean in the Gaussian space $\tilde{y}' = -\Lambda^{-1}\Theta^T x'$; then transform each element of $\tilde{y}'$ using the inverse copula transform

$$(\hat{y}')_j = F^{-1}(\Phi((\tilde{y}')_j)) \tag{7.9}$$

- **Compute the probability of a given output y':** Convert y' to the Gaussian space using (7.8) and compute

$$\begin{aligned} p(y'|x') =& p(\tilde{y}'|x'; \Theta, \Lambda) \\ =& \frac{1}{Z(x')} \exp\left\{-\frac{1}{2}(\tilde{y}')^T \Lambda \tilde{y}' - (x')^T \Theta \tilde{y}'\right\} \end{aligned} \tag{7.10}$$

- **Draw a random sample of future observations:** Sample

$$\tilde{y}' \sim \mathcal{N}(-\Lambda^{-1}\Theta^T x', \Lambda^{-1}) \tag{7.11}$$

and then apply the inverse copula transform (7.9) to $\tilde{y}'$.

7.4 Experimental results on wind power forecasting

In this section, we describe an application of the above probabilistic forecasting method to a real-world wind power prediction task. We use data from the GEFCom 2012 forecasting challenge, a wind power forecasting competition that was recently held on Kaggle [58], where the goal was to predict power output from 7 nearby wind farms over the next 48 hours using forecasted wind speed and direction as input variables.

In our setup, we model wind power production jointly across all wind farms as $z_t \in \mathbb{R}^7$ and include the forecasted wind at each farm as exogenous variables. We model the non-linear dependence of the wind power using radial basis functions (RBFs) with centers and variances tuned using cross-validation, resulting in 10 RBFs for each time point and location and $w_t \in \mathbb{R}^{3360}$. We also include autoregressive features for past wind power over the previous 8 hours which we found experimentally to be sufficient to capture the autoregressive behavior of wind power in this dataset. In our framework, the input and output variables (x_t, y_t) are compromised of w_t and z_t ranging over past and future time points

$$x_t = \begin{bmatrix} z_t \\ \vdots \\ z_{t-7} \\ w_t \end{bmatrix}, \quad y_t = \begin{bmatrix} z_{t+1} \\ \vdots \\ z_{t+48} \end{bmatrix} \tag{7.12}$$

resulting in $x_t \in \mathbb{R}^{3416}$ and $y_t \in \mathbb{R}^{336}$.

We fit the model using 80% of the provided data (874 training examples) and report results on the held out set. As baselines, we consider a linear regression model (LR) which predicts each output independently and an ARMAX model with AR(3) and MA(2) components, both using the same input features as the SGCRF.

7.4.1 Probabilistic predictions

Typically, forecasting systems are evaluated solely on the quality of the point forecasts produced and we see in Table 7.1 that, on this basis, the SGCRF method performs significantly better than

Algorithm	RMSE
Linear Regression	0.1560
Linear Regression + copula	0.1636
ARMAX	0.1714
SGCRF	**0.1488**
SGCRF + copula	0.1584

Table 7.1: Comparison of mean prediction error on wind power forecasting.

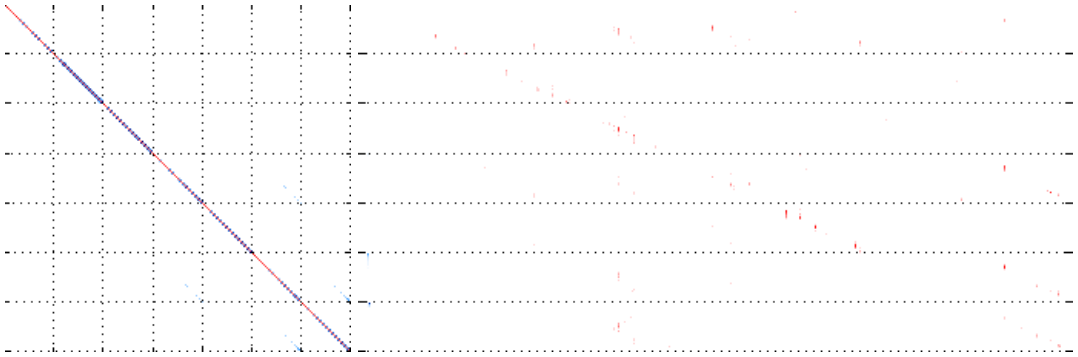

Figure 7.1: Sparsity patterns Λ and Θ from the SGCRF model. Λ is estimated to have 1412 nonzero entries (1.2% sparse) and Θ is estimated to have 7714 nonzero entries (0.67% sparse). White denotes zero values and wind farms are grouped together in blocks.

linear regression and ARMAX. Due to the high dimensionality of the feature space relative to the number of training examples, we expect the ℓ_1 penalty employed by the SGCRF to be statistically efficient in identifying the underlying structure of the correlations in wind power production and the dependence of wind power on wind forecasts; indeed in Figure 7.1, we see that the estimated parameters exhibit a high degree of sparsity.

However, we are primarily interested not in the accuracy of point forecasts, but in the ability of the models to capture the distribution of future power production. Indeed, as shown in the same Table 7.1, the inclusion of the copula transform degrades the performance of the models in terms of RMSE; this is expected since by assuming a Gaussian distribution over the noise, the untransformed models are explicitly minimizing mean squared error. RMSE alone is a poor measure of how well an algorithm can actually predict future observations: if we judge the algorithms by the ability to accurately predict the range of possibilities for future outcomes, a different picture emerges. For example, a natural task for a wind farm operation would be to generate a distribution over *total* power produced by all seven wind farms in the next 24 hours, in order to establish 95% confidence intervals about the power to be produced; this could in turn by used by stochastic optimal dispatch method, to determine how much power to generate from other sources.

Table 7.2 illustrates the *coverage* of the confidence intervals generated by different approaches, evaluated on a held-out test set of the wind power data. For each example in the test set, we used each method to generate many samples of upcoming wind power and for each of these samples, we computed the total power aggregated over all the farms, all times, or both, and used these

Method	Task	90%	95%	99%
LR	Aggregate farms	0.6943	0.7653	0.8600
	Aggregate times	0.3790	0.4451	0.5364
	Both	0.1944	0.2500	0.3333
LR + copula	Aggregate farms	0.7256	0.8051	0.9040
	Aggregate times	0.4028	0.4663	0.5728
	Both	0.2176	0.2639	0.3380
ARMAX	Aggregate farms	0.5682	0.6473	0.7570
	Aggregate times	0.6779	0.8188	0.9544
	Both	0.2454	0.3102	0.4213
SGCRF	Aggregate farms	0.8267	0.8791	0.9443
	Aggregate times	0.6104	0.6885	0.7976
	Both	0.4306	0.5370	0.6389
SGCRF + copula	Aggregate farms	**0.8981**	**0.9468**	**0.9830**
	Aggregate times	**0.8743**	**0.9266**	**0.9722**
	Both	**0.8796**	**0.9259**	**0.9676**

Table 7.2: Coverage of confidence intervals for wind power forecasting models.

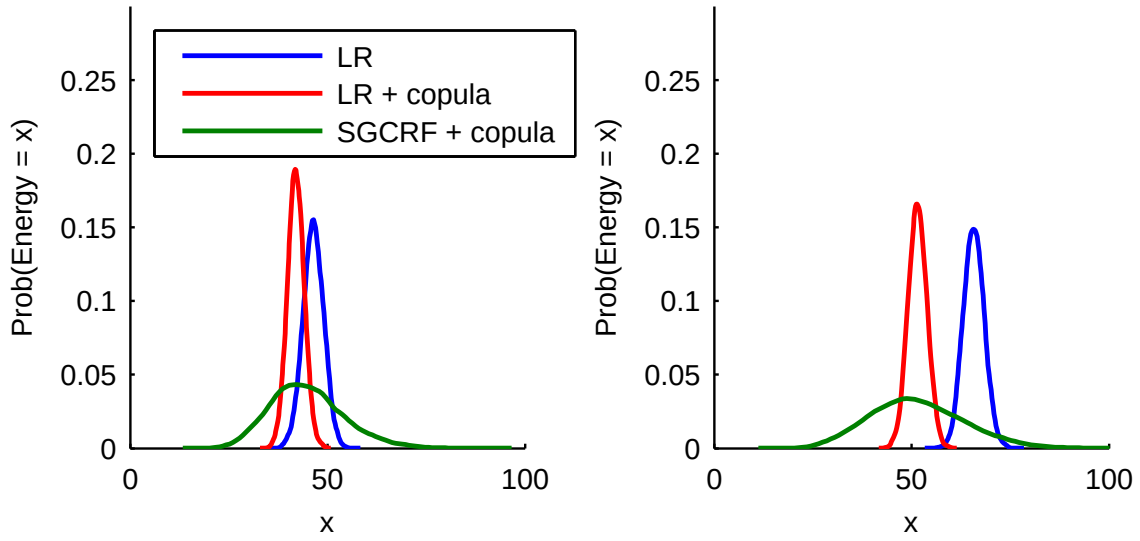

Figure 7.2: Examples of predictive distributions for total energy output from all wind farms over a single day.

to generate histograms of the aggregated power. Finally, we used these histograms to estimate 90%, 95%, and 99% confidence intervals of the aggregate power, and evaluated how often the true total wind power fell into that interval.

As seen in Table 7.2, the SGCRF + copula model produces intervals that map very closely to their desired coverage level, whereas linear Gaussian and ARMAX models perform much worse. To see why this occurs, we show in Figure 7.2 several of these estimated distributions of aggregate power, sampled from the different models. The independent Gaussian models are substantially overconfident in their predictions, as summing together multiple i.i.d. random variables will tend to tighten the variance, leading to vastly inaccurate predictions when those variables are in fact highly correlated. We note that we could also consider a joint linear model by forming the unregularized MLE for the covariance matrix, but in general this is not well-suited for high-dimensional problems and in fact is undefined for $p > m$. The ARMAX model does capture some of the correlation across time via the moving average component and we see in Table 7.2 that it performs significantly better than linear regression when aggregating predictions across multiple times. However, the SGCRF with the copula transform clearly achieves the best results implying that it is more accurately capturing the correlated nature of the actual joint distribution.

7.4.2 Ramp detection

One particularly appealing possibility for the probabilistic forecasting methods is in the area of predicting wind "ramps," times when power experiences a sudden jump from a relatively low to a relatively high value. Because wind power grows with the cube of wind speed in Region 2 of the turbine operating conditions (before the wind turbines reach rated power), a small increase in wind speed can lead to a large change in power, and predicting when these ramps occur is one of the primary open challenges in wind forecasting. Indeed, a well-known issue with many forecasting methods is that while they may accurately predict that a ramp will occur, they are significantly limited in accurately capturing the uncertainty over where the ramp will occur [45].

Although a detailed analysis of the ramp prediction capabilities of our approach is not the main focus on this paper, we briefly highlight the potential of our approach in this task. In particular, because the model accurately captures correlations in the predicted observations over time, if we draw random samples from our model, then we expect scenarios where the ramp occurs at different times; this is in contrast to most "independent" probabilistic methods, which would assume a fairly tight distribution over possible future scenarios. This situation is illustrated in Figure 7.3, where we show the mean prediction along with 10 samples drawn from each model. Again, in a stochastic optimal control task, an operator would be much better served by considering the possible scenarios generated from our model than from the independent probabilistic model, as they consist of several different timings for the upcoming power ramp.

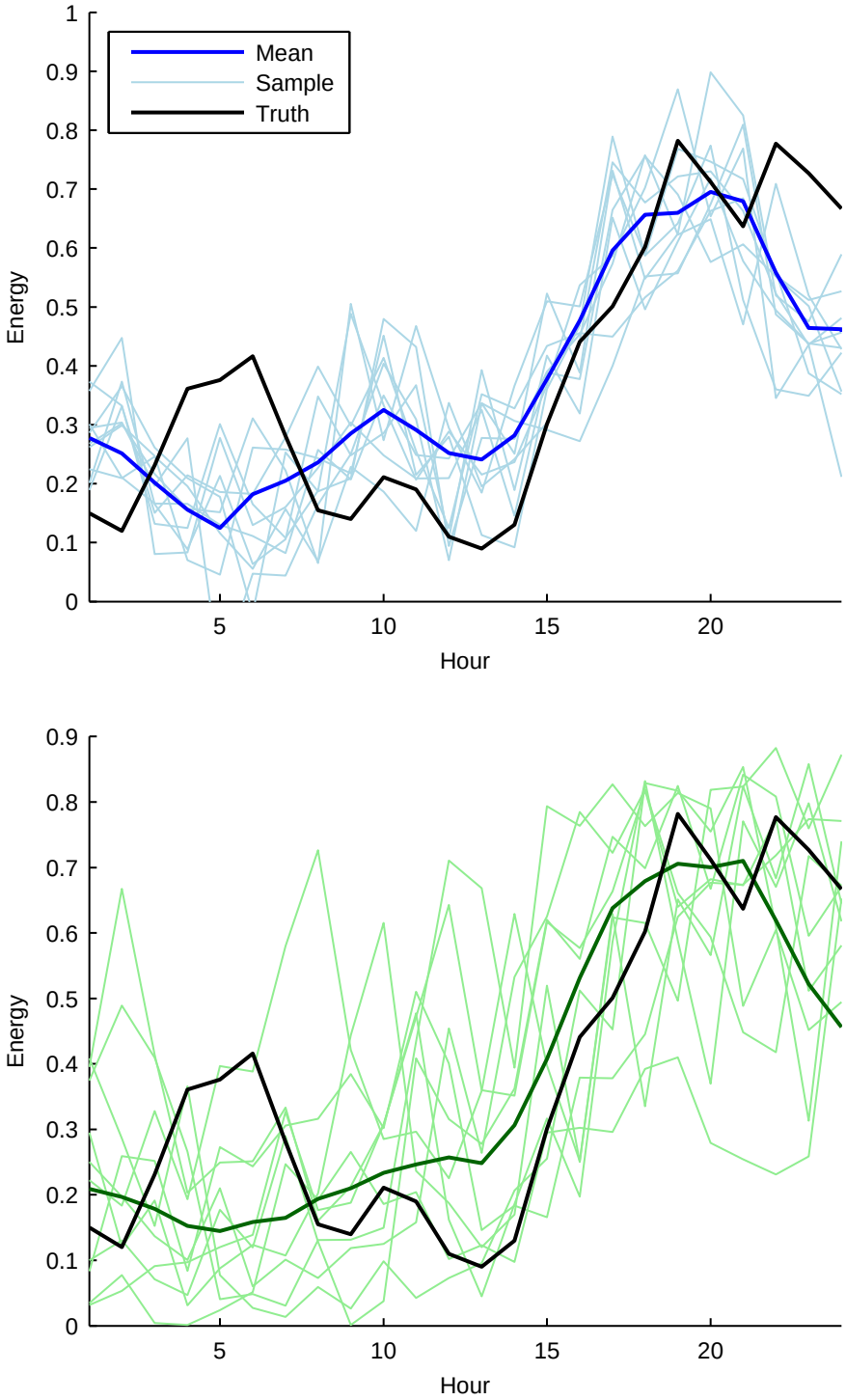

Figure 7.3: Samples drawn from the linear regression model (top), and SGCRF model (bottom)

Chapter 8

Contextually Supervised Source Separation for Energy Disaggregation

Contextually supervised source separation is a framework for single-channel source separation that lies between the fully supervised and unsupervised setting. Instead of supervision, we provide input features for each source signal and use convex methods to estimate the correlations between these features and the unobserved signal decomposition. It is a natural fit for domains with large amounts of data but no explicit supervision; for example, energy disaggregation of hourly smart meter data (the separation of whole-home power signals into different energy uses). Here contextual supervision allows us to provide itemized energy usage for thousands of homes, a task previously impossible due to the need for specialized data collection hardware. On smaller datasets which include labels, we demonstrate that contextual supervision improves significantly over a reasonable baseline and existing unsupervised methods for source separation.

8.1 Introduction

We consider the *single-channel source separation* problem, in which we wish to separate a single aggregate signal into a mixture of unobserved component signals. Traditionally, this problem has been approached in two ways: the *supervised* setting [70, 107, 111],where we have access to training data with the true signal separations and the *unsupervised* (or "blind") setting [18, 30, 74, 110], where we have only the aggregate signal. However, both settings have potential drawbacks: for many problems, including energy disaggregation—which looks to separate individual energy uses from a whole-home power signal [57]—it can be difficult to obtain training data with the true separated signals needed for the supervised setting; in contrast, the unsupervised setting is an ill-defined problem with arbitrarily many solutions, and thus algorithms are highly task-dependent.

Contextually supervised source separation provides a compelling framework for energy disaggregation from "smart meters", communication-enabled power meters that are currently installed in more than 32 million homes [63], but are limited to recording whole home energy usage at low frequencies (every 15 minutes or hour). This is an important task since many studies have shown that consumers naturally adopt energy conserving behaviors when presented

with a breakdown of their energy usage [29, 42, 86]. There are several possible ways that such a breakdown could be achieved; for example, by installing current sensors on each device we could monitor electricity use directly. But, as custom hardware installation is relatively expensive (and requires initiative from homeowners), algorithmic approaches that allow disaggregation of energy data already being collected are appealing. However, existing energy disaggregation approaches virtually all use high-frequency sampling (e.g. per second or faster) which still requires the installation of custom monitoring hardware for data collection. In contrast, by enabling disaggregation of readily available low-resolution smart meter data, we can immediately realize the benefits of observing itemized energy use without the need for additional monitoring hardware.

8.1.1 Related work

As mentioned above, work in single-channel source separation has been separated along the lines of supervised and unsupervised algorithms. A common strategy is to separate the observed aggregate signal into a linear combination of several *bases*, where different bases correspond to different components of the signal; algorithms such as Probabilistic Latent Component Analysis (PLCA) [116], sparse coding [94], and factorial hidden Markov models (FHMMs) [50] all fall within this category, with the differences concerning 1) how bases are represented and assigned to different signal components and 2) how the algorithm infers the activation of the different bases given the aggregate signal. For example, PLCA typically uses pre-defined basis functions (commonly Fourier or Wavelet bases), with a probabilistic model for how sources generate different bases; sparse coding learns bases tuned to data while encouraging sparse activations; and FHMMs use hidden Markov models to represent each source. In the supervised setting, one typically uses the individual signals to learn parameters for each set of bases (e.g., PLCA will learn which bases are typical for each signal), whereas unsupervised methods learn through an EM-like procedure or by maximizing some separation criteria for the learned bases. The method we propose here is conceptually similar, but the nature of these bases is rather different: instead of fixed bases with changing activations, we require features that effectively generate time-varying bases and learn activations that are constant over time.

Orthogonal to this research, there has also been a great deal of work in *multi-channel* blind source separation problems, where we observe multiple mixings of the same sources (typically, as many mixings as there are signals) rather than in isolation. These methods can exploit significantly more structure and algorithms like Independent Component Analysis [12, 26] can separate signals with virtually no supervised information. However, when applied to the single-channel problem (when this is even possible), they typically perform substantially worse than methods which exploit structure in the problem, such as those described above.

From the applied point of view, algorithms for energy disaggregation have received growing interest in recent years [66, 69, 70, 98, 149] but these approaches all use either high-frequency sampling of the whole-building power signal or known (supervised) breakdowns whereas the focus of this work is disaggregating low-resolution smart data without the aid of explicit supervision, as discussed in the previous section.

8.2 Contextually supervised source separation

We begin by formulating the optimization problem for contextual source separation. Formally, we assume there is some unknown matrix of k component signals

$$Y \in \mathbb{R}^{T \times k} = \begin{bmatrix} | & | & & | \\ y_1 & y_2 & \cdots & y_k \\ | & | & & | \end{bmatrix} \tag{8.1}$$

from which we observe the sum $\bar{y} = \sum_{i=1}^{k} y_i$. For example, in our disaggregation setting, $y_i \in \mathbb{R}^T$ could denote a power trace (with T total readings) for a single type of appliance, such as the air conditioning, lighting, or electronics, and $\bar{y}$ denotes the sum of all these power signals, which we observe from a home's power meter.

In our proposed model, we represent each individual component signal y_i as a linear function of some component-specific bases $X_i \in \mathbb{R}^{T \times n_i}$

$$y_i \approx X_i \theta_i \tag{8.2}$$

where $\theta_i \in \mathbb{R}^{n_i}$ are the signal's coefficients. The formal objective of our algorithm is: given the aggregate signal $\bar{y}$ and the component features X_i, $i = 1, \ldots, k$, estimate both the parameters θ_i and the unknown source components y_i. We cast this as an optimization problem

$$\begin{aligned} \underset{Y,\theta}{\text{minimize}} \quad & \sum_{i=1}^{k} \{\ell_i(y_i, X_i\theta_i) + g_i(y_i) + h_i(\theta_i)\} \\ \text{subject to} \quad & \sum_{i=1}^{k} y_i = \bar{y} \end{aligned} \tag{8.3}$$

where $\ell_i : \mathbb{R}^T \times \mathbb{R}^T \to \mathbb{R}$ is a loss function penalizing differences between the ith reconstructed signal and its linear representation; g_i is a regularization term encoding the "likely" form of the signal y_i, independent of the features; and h_i is a regularization penalty on θ_i. Choosing ℓ_i, g_i and h_i to be convex functions results in a convex optimization problem.

A natural choice of loss function ℓ_i is a norm penalizing the difference between the reconstructed signal and its features $\|y_i - X_i\theta_i\|$, but since our formulation enables loss functions that depend simultaneously on all T values of the signal, we allow for more complex choices as well. For example in the energy disaggregation problem, air conditioning is correlated with high temperature but does not respond to outside temperature changes instantaneously; thermal mass and the varying occupancy in buildings often results in air conditioning usage that correlates with high temperature over some window (for instance, if no one is in a room during a period of high temperature, we may not use electricity then, but need to "make up" for this later when someone does enter the room). In this case, the loss function

$$\ell_i(y_i, X_i\theta_i) = \|(y_i - X_i\theta_i)(I \otimes 1^T)\|_2^2 \tag{8.4}$$

which penalizes the aggregate difference of y_i and $X_i\theta_i$ over a sliding window, can be used to capture such dynamics. In many settings, it may also make sense to use ℓ_1 or ℓ_∞ rather than ℓ_2 loss, depending on the nature of the source signal.

Model	MAE
Mean prediction	0.3776
Nonnegative sparse coding	0.2843
ℓ_2 loss for y_1	0.1205
ℓ_2 loss for y_1, ℓ_1 loss for y_2	0.0994
ℓ_2 loss for y_1, ℓ_1 loss for y_2, penalty on $\|Dy_i\|$	**0.0758**

Table 8.1: Performance on disaggregation of synthetic data.

Likewise, since the objective term g_i depends on all T values of y_i, we can use it to encode the likely dynamics of the source signal independent of $X_i\theta_i$. For air conditioning and other single appliance types, we expect sharp transitions between on/off states which we can encode by penalizing the ℓ_1 norm of Dy_i where D is the linear difference operator subtracting $(y_i)_{t-1}-(y_i)_t$. For other types of energy consumption, for example groups of many electronic appliances, we expect the signal to have smoother dynamics and thus ℓ_2 loss is more appropriate. Finally, we also include h_i for statistical regularization purposes—but for problems where $T \gg n_i$, such as the ones we consider in energy disaggregation, the choice of h_i is less important.

8.3 Experimental results

In this section we evaluate contextual supervision for energy disaggregation on one synthetic dataset and two real datasets. On synthetic data we demonstrate that contextual supervision significantly outperforms existing methods (e.g. nonnegative sparse coding) and that by tailoring the loss functions to the expected form of the component signals (as is a feature of our optimization framework), we can significantly increase performance. On real data, we begin with a dataset from Pecan Street, Inc. (`http://www.pecanstreet.org/`) that is relatively small (less than 100 homes), but comes with labeled data allowing us to validate our unsupervised algorithm quantitatively. Here we show that our unsupervised model does remarkably well in disaggregating sources of energy consumption and improves significantly over a reasonable baseline. Finally, we apply the same methodology to disaggregate large-scale smart meter data from Pacific Gas and Electric (PG&E) consisting of over 4000 homes and compare the results of our contextually supervised model to aggregate statistics from independent survey data.

In all experiments, we tune the model using hyperparameters that weigh the terms in the optimization objective; in the case of energy disaggregation, the model including hyperparameters α and β is shown in Table 8.3. We set these hyperparameters using a priori knowledge about the relative frequency of each signal over the entire dataset. For energy disaggregation, it is reasonable to assume that this knowledge is available either from survey data (e.g. [128]), or from a small number of homes with more fine-grained monitoring, as is the case for the Pecan Street dataset. In both cases, we use the same hyperparameters for all homes in the dataset.

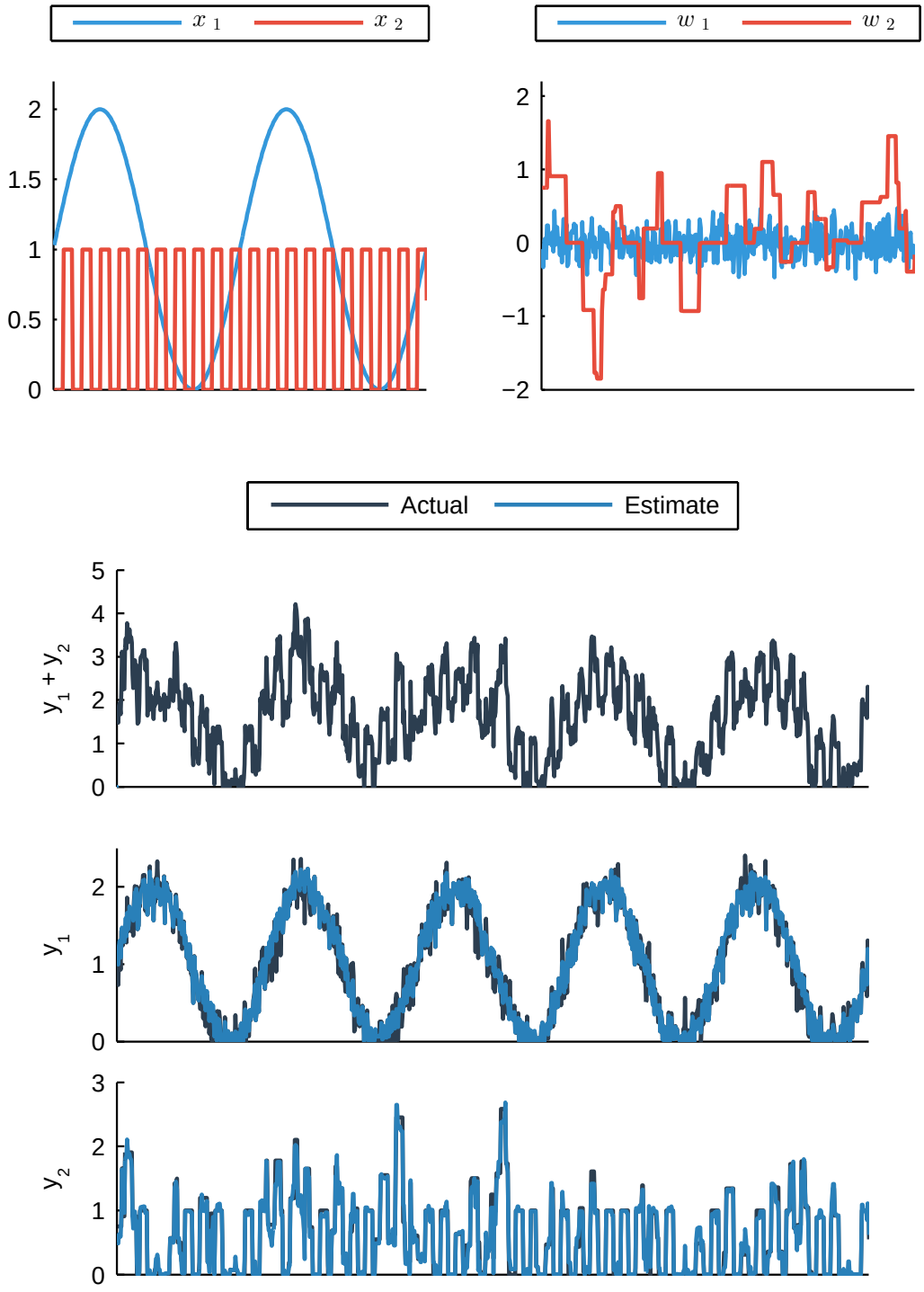

Figure 8.1: Synthetic data generation process starting with two underlying signals (top left), corrupted by different noise models (top right), summed to give the observed input (row 2) and disaggregated (rows 3 and 4).

Category	Mean	NNSC	Contextual
Base	0.2534	0.2793	0.1849
A/C	0.2849	0.2894	0.1919
Appliance	0.2262	0.2416	0.1900
Average	0.2548	0.2701	**0.1889**

Table 8.2: Comparison of performance on Pecan Street dataset, measured in mean absolute error (MAE).

Category	Features	ℓ_i	g_i
Base	Hour of day	$\alpha_1 \|y_1 - X_1\theta_1\|_1$	$\beta_1 \|Dy_1\|_2^2$
Heating	RBFs over temperatures $< 50°\text{F}$	$\alpha_2 \|S_2(y_3 - X_3\theta_3)\|_1$	$\beta_2 \|Dy_3\|_1$
A/C	RBFs over temperatures $> 70°\text{F}$	$\alpha_3 \|S_2(y_2 - X_2\theta_2)\|_1$	$\beta_3 \|Dy_2\|_1$
Appliance	None	$\alpha_4 \|y_4\|_1$	$\beta_4 \|Dy_4\|_1$

Table 8.3: Model specification for contextually supervised energy disaggregation.

8.3.1 Disaggregation of synthetic data

The first set of experiments considers a synthetic generation process that mimics signals that we encounter in energy disaggregation. The process described visually in Figure 8.1 (top) begins with two signals, the first is smoothly varying over time while the other is a repeating step function

$$X_1(t) = \sin(2\pi t/\tau_1) + 1, \quad X_2(t) = I(t \bmod \tau_2 < \tau_2/2) \tag{8.5}$$

where $I(\cdot)$ is the indicator function and τ_1, τ_2 are the period of each signal. We also use two different noise models: for the smooth signal we sample Gaussian noise from $\mathcal{N}(0, \sigma^2)$ while for the step function, we sample a distribution with a point mass at zero, uniform probability over $[-1, 0) \cup (0, 1]$ and correlate it across time by summing over a window of size β. Finally, we constrain both noisy signals to be nonnegative and sum them to generate our input.

We generate data under this model for $T = 50000$ time points and consider increasingly specialized optimization objectives while measuring the error in recovering $Y^\star = XD(\theta^\star) + W$, the underlying source signals corrupted by noise. As can be seen in Table 8.1, by using ℓ_1 loss for y_2 and adding $g_i(y_i)$ terms penalizing $\|Dy_1\|_2^2$ and $\|Dy_2\|_1$, error decreases by 37% over just ℓ_2 loss alone; in Figure 8.1, we observe that our estimation recovers the true source signals closely with the g_i terms helping to capture the dynamics of the noise model for w_2.

As a baseline for this result, we compare to the mean prediction heuristic (predicting at each time point a breakdown proportional to the overall probability of each signal) and to a state-of-the-art unsupervised method, nonnegative sparse coding [60]. We apply sparse coding by segmenting the input signal into 1000 examples of 50 time points (1/4 the period of the sine wave, $X_1(t)$) and fit a sparse model of 200 basis functions. We report the best possible source separation by assigning each basis function according to an oracle measuring correlation with the true source signal and using the best value over a grid of hyperparameters. As can be seen in Table 8.1, the mean prediction heuristic is nearly 5 times worse and sparse coding is nearly 4 times worse than our best contextually supervised model.

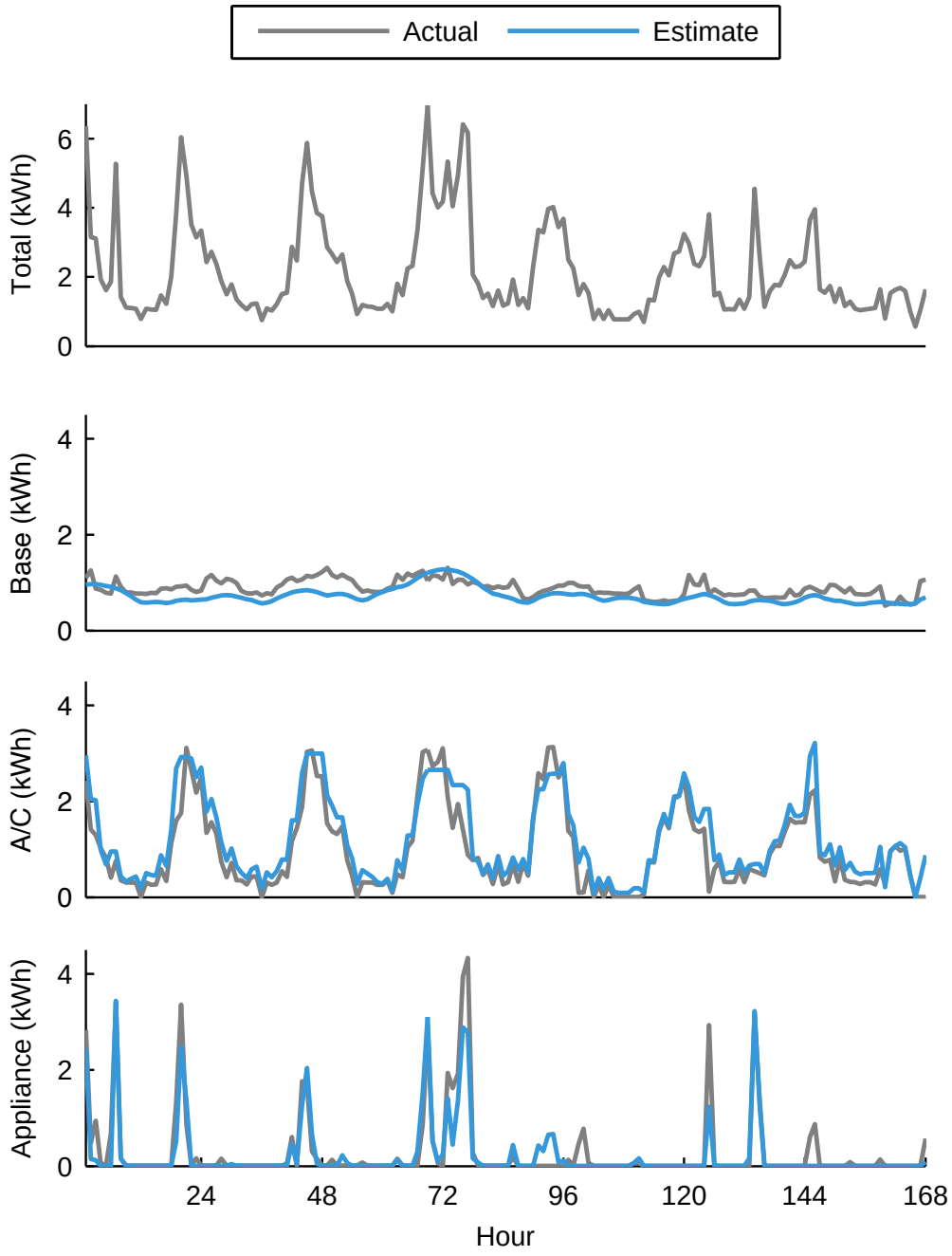

Figure 8.2: Energy disaggregation results over one week and a single home from the Pecan Street dataset.

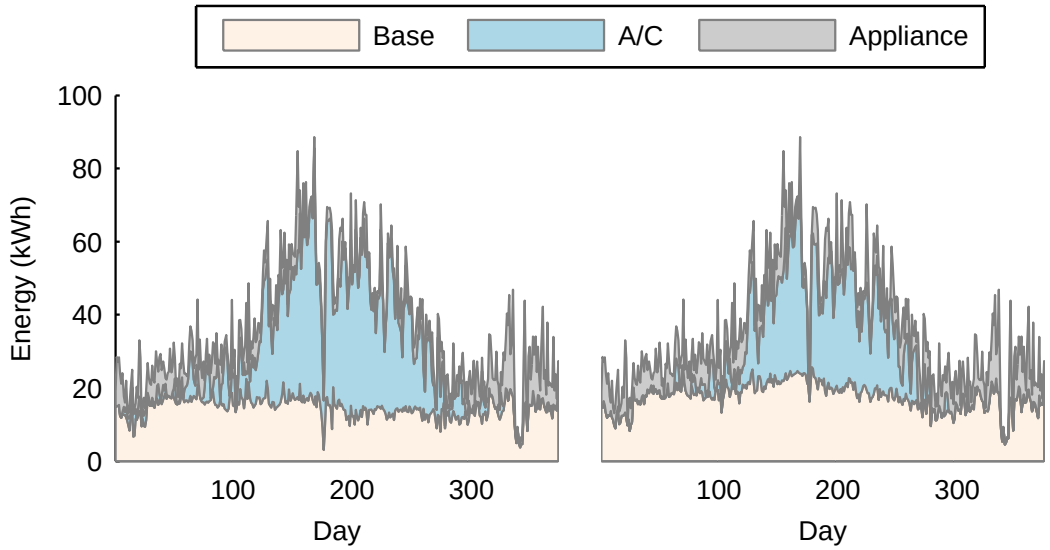

Figure 8.3: Energy disaggregation results over entire time period for a single home from the Pecan Street dataset with estimated (left) and actual (right).

8.3.2 Energy disaggregation with ground truth

Next we consider the ability of contextual supervision to recover the sources of energy consumption on a real dataset from Pecan Street consisting of 84 homes each with at least 1 year worth of energy usage data. As contextual information we construct a temperature time series using data from Weather Underground (`http://www.wunderground.com/`) measuring the temperature at the nearby airport in Austin, Texas. The Pecan Street dataset includes fine-grained energy usage information at the minute level for the entire home with an energy breakdown labeled according to each electrical circuit in the home. We group the circuits into categories representing air conditioning, large appliances and base load and aggregate the data on an hourly basis to mimic the scenario presented by smart meter data.

The specification of our energy disaggregation model is given in Table 8.3—we capture the non-linear dependence on temperature with radial-basis functions (RBFs), include a "Base" category which models energy used as a function of time of day, and featureless "Appliance" category representing large spikes of energy which do not correspond to any available context. For simplicity, we penalize each category's deviations from the model using ℓ_1 loss; but for heating and cooling we first multiply by a smoothing matrix S_n (1's on the diagonal and n super diagonals) capturing the thermal mass inherent in heating and cooling: we expect energy usage to correlate with temperature over a window of time, not immediately. We use $g_i(y_i)$ and the difference operator to encode our intuition of how energy consumption in each category evolves over time. The "Base" category represents an aggregation of many sources which we expect to evolve smoothly over time, while the on/off behavior in other categories is best represented by the ℓ_1 penalty. Finally we note that in the Pecan Street data, there is no labeled circuit corresponding exclusively to electric heating ("Heating"), and thus we exclude this category for this dataset.

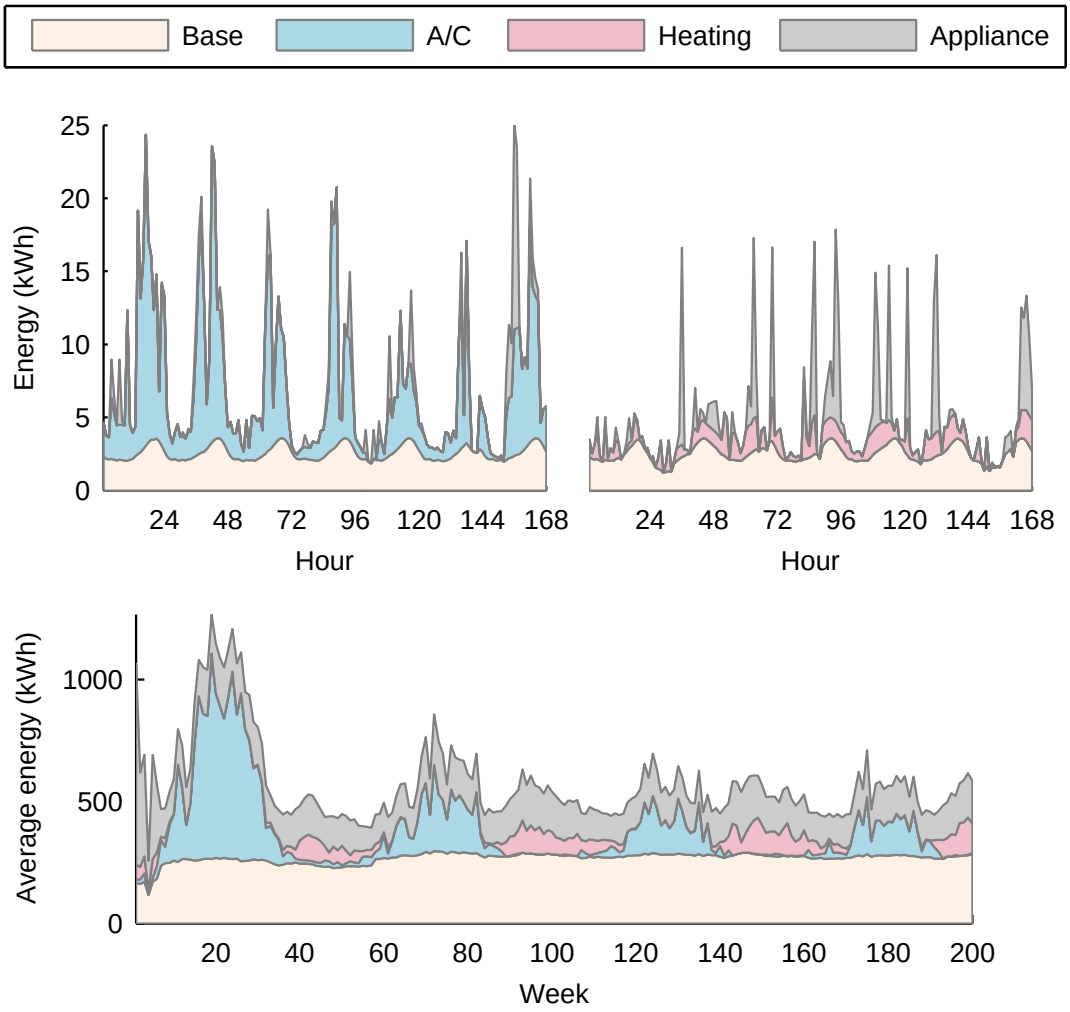

Figure 8.4: Disaggregated energy usage for a single home near Fresno, California over a summer week (top left) and a winter week (top right); aggregated over 4000+ homes over nearly four years (bottom)

In Table 8.2, we compare the results of contextual supervision with the mean prediction heuristic and see that contextual supervision improves by 26% over this baseline which is already better than nonnegative sparse coding. Qualitatively we consider the disaggregated energy results for a single home over one week in Figure 8.2 and see that contextual supervision correctly assigns energy usage to categories—a large amount of energy is assigned to A/C which cycles on and off roughly corresponding to external temperature, large spikes from appliances happen at seemingly random times and the smoothly varying base load is captured correctly. In Figure 8.3, we consider the disaggregation results for the same home across the entire time period and see that the contextually supervised estimates correspond very closely to the actual sources of energy consumption.

8.3.3 Large-scale energy disaggregation

Next, we turn to the motivating problem for our model: disaggregating large-scale low-resolution smart meter data into its component sources of consumption. Our dataset consists of over 4000 homes and was collected by PG&E from customers in Northern California who had smart meters between 1/2/2008 and 12/31/2011. According to estimations based on survey data, heating and cooling (air conditioning and refrigerators) comprise over 39% of total consumer electricity usage [128] and thus are dominant uses for consumers. Clearly, we expect temperature to have a strong correlation with these uses and thus we provide contextual supervision in the form of temperature information. The PG&E data is anonymized, but the location of individual customers is identified at the census block level and we use this information to construct a parallel temperature dataset as in the previous example.

We present the result of our model at two time scales, starting with Figure 8.4 (top), where we show disaggregated energy usage for a single home over a typical summer and winter week. Here we see that in summer, the dominant source of energy consumption is estimated to be air conditioning due to the context provided by high temperature. In winter, this disappears and is replaced to a smaller extent by heating. In Figure 8.4 (bottom), itemized energy consumption aggregated across all 4000+ homes demonstrates these basic trends in energy usage. Quantitatively, our model assigns 15.6% of energy consumption to air conditioning and 7.7% to heating, reasonably close to estimations based on survey data [128] (10.4% for air conditioning and 5.4% for space heating). We speculate that our higher estimation may be due to the model conflating other temperature-related energy usages (e.g. refrigerators and water heating) or to differences in populations between the survey and smart meter customers.

Chapter 9

Preventing Cascading Failures in Microgrids

In this chapter, we consider the challenge of building and operating microgrids, a fundamentally different setting than the conventional electrical grid that was implicitly the focus of Chapters 7 and 8. Whereas the conventional grid features top-down, central control (e.g. by a government-endorsed entity, such as a utility), microgrids are formed by small, isolated networks of individual devices and thus are naturally bottom-up. As such, they are not encumbered by the legacy requirements of the conventional grid and have many potential advantages, particularly with respect to resiliency. Furthermore, in many parts of the developing world, e.g. rural Africa and India, microgrids are the only practical option due to a lack of existing grid infrastructure. In these settings, the microgrid approach is increasingly attractive due to the falling cost of renewable generation technology (photovoltaic cells) and cheap power electronics. In the developed world, microgrids continue to find many applications including large vehicles (airplanes, boats, RVs, etc.) and sites which require redundant or isolated sources of power (e.g. military installations, hospitals, and data centers). In addition, with the falling cost of solar panels and battery storage, there is increasing interest in microgrid technology at the individual consumer and business level—for example, to operate a single home in off-grid mode in the case of grid failure.

While microgrids offer many deployment and operational advantages, they also pose a number of unique challenges. The most fundamental challenge is a lack of *inertia*: conventional grids have massive scale and thus the behavior of a single device has relatively small impact on the overall flow of current over the entire network. This allows conventional grid operators to ensure that the voltage and frequency remain tightly regulated and enables energy-consuming devices to assume that current can be sourced without affecting these characteristics. In contrast, microgrids are much smaller and individual devices can cause significant transient responses simply by switching on or off. Typically, it is the responsibility of the power electronics connected to generating devices to ensure microgrid power quality by regulating voltage and frequency within a pre-specified acceptable range, usually a much wider operating band than is accepted in conventional grids. In the common case of an AC microgrid, the power electronics are also responsible for transforming DC input (e.g. from photovoltaic cells) to AC output and are referred

to generically in the microgrid literature as *inverters*[1].

In the microgrid setting, the inverters representing each power source are responsible for ensuring reliable operation by providing the necessary current to energy-consuming devices (loads). At the same time, due to the possibility of large transient responses due to microgrid dynamics driven by low inertia, these devices must include electronic circuit breakers (ECBs, or simply, breakers) which protect their components from an overload condition. The breaker safety settings cause the inverter to disengage from the microgrid once a local current threshold is exceeded for a specified time duration protecting components that cannot withstand high currents for long periods of time. As such, there is a tradeoff between conservative breaker settings (low threshold, short time duration) which protect the individual device at the expense of the microgrid and more aggressive settings (high threshold, long time duration) which sacrifice some protection in order to allow the microgrid to be more resilient to transients. The challenge that we will consider in this chapter is to identify reasonable breaker settings under realistic load and generation profiles, enabling stable microgrid operation while protecting individual devices.

Identifying breaker settings for a realistic microgrid is difficult as the flow of current over the microgrid network is influenced by many factors. In short, a sequence of breaker trips can be caused in a multititude of ways with each trip event depending on the threshold values and durations placed on each inverter, the load profile, and the time-varying generating capacities of each power source (common scenarios involve renewable sources which depend on external weather). Moreover tripping is often caused by transient conditions in the highly interconnected microgrid system involving loads which are often time-varying and nonlinear (i.e. with impedances that change over time and depending on voltage). These factors make it difficult to analytically prescribe breaker settings even for small microgrids with known loads and power sources.

Instead, our approach, which we will explore in this chapter, is based on learning a set of breaker parameters defining a stable region of microgrid operation from data. As the actual failure of a microgrid and the corresponding power equipment can be dangerous and costly, this work focuses on learning from simulation with a realistic microgrid model which we present in detail in Section 9.1. In practice, we expect that our approach would be used in an iterative process in which a realistic set of parameters are learned via a simulation which is continuously adjusted to match real operating conditions. When the actual microgrid is deployed and operating, our learning can also be applied directly to historical data collected from current and voltage sensing devices on the microgrid.

The rest of this chapter is organized as follows. Section 9.2 presents our machine learning model which is a variant of the classic support vector machine (SVM). From the perspective of machine learning, the method is straightforward, simply learning a function that maps breaker settings for the entire microgrid (a single feature vector concatenating threshold and time duration for each inverter) to a classification of microgrid failure or stability within a time horizon. We do make one modification to the standard SVM, employing a "one-sided" variant which heavily weights stable operation under the assumption that if a breaker configuration fails even once, it may fail again in the future and thus should be deemed unstable. This can be viewed as an extreme version of class weighting and from a geometric perspective produces a decision

[1]A network of inverters is a standard microgrid topology even when including power sources which naturally produce alternating current, e.g. diesel generators.

function that captures the entire negative class on one side of the hyperplane. On simulated data, presented in Section 9.3, we demonstrate that the one-sided SVM variant outperforms the standard SVM in the low false positive regime of interest for this problem.

9.1 System and problem description

In this section, a model of a real microgrid is presented (see Figure 9.2). The data used by the machine learning algorithms proposed in this chapter are obtained using a simulation engine based on this model. This simulation engine employs high-fidelity models of inverters and loads, and droop-based control architectures [55]. The simulation models and control architecture described in this section are not the goals of this chapter; they provide the context under which the data for machine learning is obtained and emphasize the details of the microgrid that the model captures. It is to be noted that even though the study is focused on a specific microgrid the framework being developed is applicable for broader class of microgrids.

A microgrid comprises multiple power sources, multiple loads that consume power, power electronic devices (such as inverters), and controllers (see Figure 9.2). The power generated from various sources is conditioned (and shared) by inverters that provide and distribute power to the loads consistent with requirements of the load. For instance an inverter whose input is provided by a DC power source can output an AC current through appropriate control at a fast time scale of related electronics. After averaging the fast time scale dynamics, an inverter can be modeled as a controllable voltage source, v_{inv}, with an inductor and a capacitor as shown in Figure 9.1 [145]. A control system manages power sharing and voltage regulation at the outputs of the inverters as different loads come on or go off the grid. More specifically, the control system can feedback measured signals such as inductor current, inverter output voltage and current for determining the switching control (or equivalently the v_{inv} in the averaged model).

In the microgrid considered in this chapter, two power sources provide energy to a shared load where the power flow is conditioned by inverters. For each inverter the operation is realized using standard PWM based operation based on high bandwidth periodic switching which admits the average model described by Figure 9.1 [145]. It is assumed for each inverter that the inverter

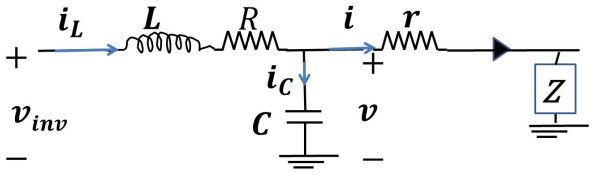

Figure 9.1: Averaged model of a single inverter with a linear load Z.

output current i, inductor current i_L and inverter output voltage v are measured variables. The control system for each inverter consists of an outer-control loop that implements the voltage-active power droop law to generate a reference $v_{lref,k}$ where

$$v_{lref,k} = [E^* - n_k(P_k - P_k^*)] \sin \omega_0 t \tag{9.1}$$

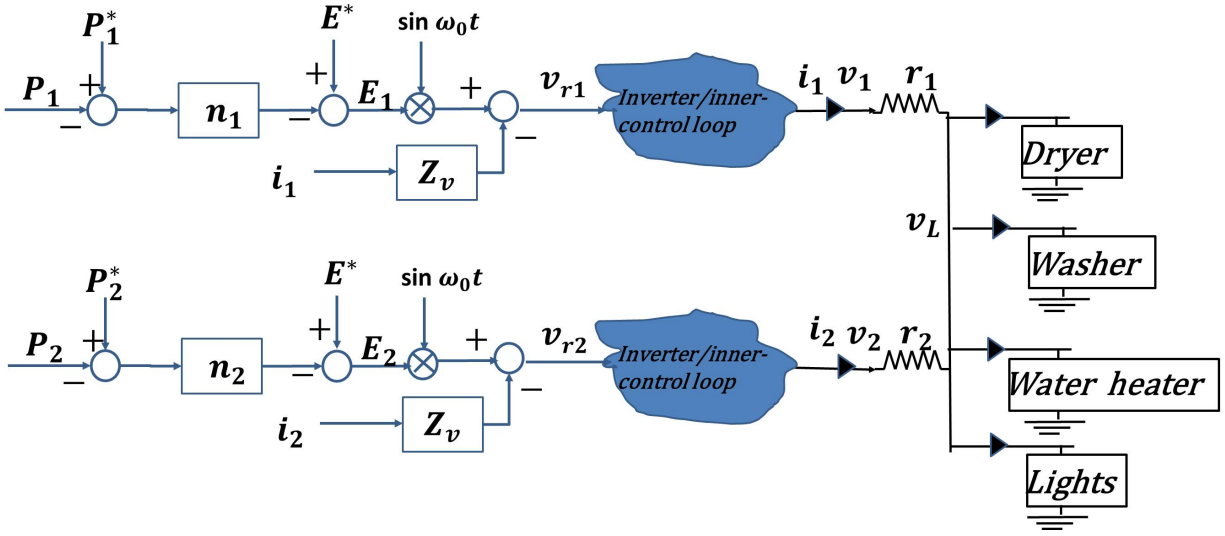

Figure 9.2: (a) Schematic describing the system shows the outer-control loop. The loads being serviced are a dryer, washer, water heater and lights.

where E^* is set to $120\sqrt{2}$, $\omega_0 = 2\pi f_0$ with $f_0 = 60Hz$, P_k^* is the setpoint active power to be sourced by the inverter, P_k is the actual power being delivered by the inverter and n_k is the droop coefficient which dictates the change in the voltage magnitude desired for a given error $P_k - P_k^*$ (see Figure 9.2). The inner control loop generates the reference voltage $v_{r,k}$ to be tracked by the k^{th} inverter given by

$$v_{r,k} = v_{lref,k} - Z_v i_k \tag{9.2}$$

where Z_v is the virtual resistance [55]. The reference voltage $v_{r,k}$ is tracked using a inner voltage and current controller (see Figure 9.3). Note that the reference voltage $v_{r,k}$ is tracked using a

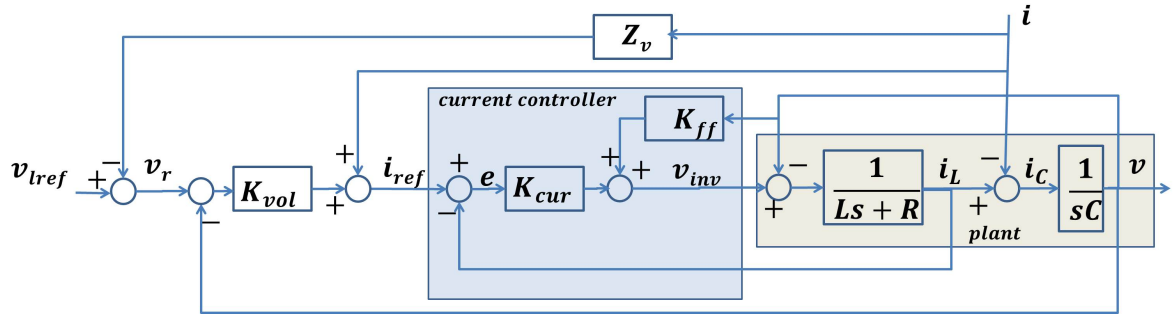

Figure 9.3: The inner-control loop for voltage regulation where a virtual resistance is incorporated.

inner-control current loop. Here, the output voltage is compared with the reference $v_{r,k}$ which is used to generate a reference current $i_{ref,k}$ to be tracked by the current controller K_{cur}. The tracking of the voltage reference via a current controller allows for placing safety measures to

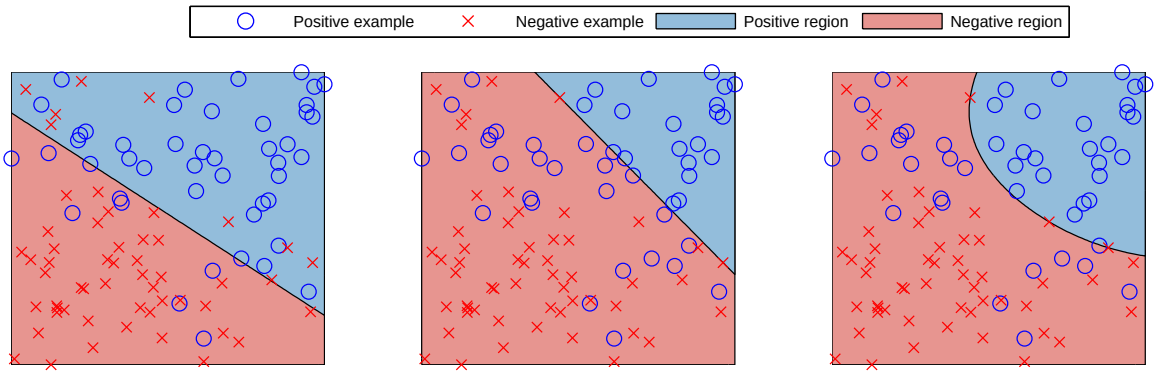

Figure 9.4: Example of method on synthetic data with linear SVM (left), one-sided SVM (center) and one-sided SVM with RBF kernel (right).

limit unsafe magnitudes of i_{ref}. Furthermore, to safeguard the inverters from damage, if the RMS of the k^{th} inverter inductor current, $i_{L,k}$ crosses a specified threshold $i_{th,k}$ and remains above the threshold for a specified guard-time $t_{g,k}$ then the k^{th} inverter is shut down.

The loads consist of a washer, dryer, water heater (rated at 1.7 KW) and lights (rated at 1.3 KW). Washer and dryer pose load profiles which vary with time. The models used for washer and dryer are representative of the behavior of generic washer and dryers and are realistic. The total commanded generation is 4 KW with one inverter sourcing 1 KW while the other sources 3 KW. In the setup above the choice of $(i_{th,1}, t_{g,1})$ and $(i_{th,2}, t_{g,2})$ has a significant influence on the overall viability of the system for a demanding load profile. Aggressive choices of $(i_{th,1}, t_{g,1})$ and $(i_{th,2}, t_{g,2})$ motivated by objectives of protecting devices can lead to unacceptable probability of blackout, whereas, on the other hand large values can damage the inverters.

We consider the following functions to provide a guidance on the choice of the breaker settings. The function $f_0 : \mathbb{R}^4 \to \{-1, 1\}$ which takes the input $(i_{th,1}, t_{g,1}, i_{th,2}, t_{g,2})$ and provides an output s where $s = 1$ implies that at least one inverter remains operational and $s = -1$ implies that both inverters have shut down. Similarly $f_1 : \mathbb{R}^4 \to \{-1, 1\}$ represents the survivability of inverter 1 where the input remains the same as for f_0 and the output is 1 if inverter 1 remains operational else the output is -1; f_2 is defined similar to f_1 to describe the survivability of inverter 2.

Remark 1 *Note that occurrence of a blackout (where all inverters shut down) in a microgrid depends on a complex interaction of many factors of which $i_{th,1}, t_{g,1}, i_{th,2}, t_{g,2}$ form only a small subset. These factors include dynamics of loads, the on-and-off time schedules of loads, the controller architecture, and power commanded from each inverter. The framework for learning functions f_k ($k = 0, 1$ and 2) has to consolidate the variability of factors not provided as inputs to f_k.*

9.2 Machine learning model

In the following a machine learning approach is taken to learn the functions f_0, f_1 and f_2. This approach is appealing as it makes few assumptions on the underlying system—in particular, we do not attempt to model the complex time-varying and nonlinear switched dynamics that describe the microgrid comprising the inverters, the loads, and the control system. We also make limited assumptions on the load schedule which in practice is controlled by the user of the microgrid who expects faultless operation. Our approach learns the function f_k ($k = 0, 1, 2$) from test data and in the large sample limit will converge to the true underlying function, subject to weak smoothness constraints on f_k [131].

We first formulate the problem in the classification framework; the *feature set* x and *predicted outcome* y respectively refer to the input and the output of the function f_k to be learned. A labeled example is a tuple consisting of a feature set x_i and the corresponding *observed* outcome y_i. In the standard setting, given a set of n labeled examples (x_i, y_i) for $i = 1, \ldots, n$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$, the classification problem seeks $f(x_i)$ which minimizes a composite objective including the empirical loss between $f(x_i)$ and y_i and structural penalties on f. For each $k \in \{0, 1, 2\}$, we aim at solving a classification problem to obtain f_k.

We first consider the standard support vector machine (SVM) [27] for solving the classification problem; SVMs are appealing because they have a natural geometric interpretation which we adapt to our problem and because they yield efficient algorithms for learning nonlinear functions through dual formulation and kernels. The result of a basic SVM is a hyperplane that separates *positive* examples (where features x_i lead to outcome $y_i = 1$) and *negative* examples (where features x_i lead to outcome $y_i = -1$). The basic SVM seeks the optimal hyperplane separating positive and negative examples by solving the convex optimization problem

$$\begin{aligned} &\underset{w,b}{\text{minimize}} \;\; \frac{1}{2}\|w\|^2 \\ &\text{subject to} \;\; y_i(w^T x_i - b) \geq 1, \;\; 1 \leq i \leq n, \end{aligned} \tag{9.3}$$

where the learned parameters $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ determine a hyperplane in p dimensions. In the above setting the constraint $y_i(w^T x_i - b) \geq 1$ characterizes two conditions

$$\begin{aligned} &(w^T x_i - b) \geq 1 \text{ for all } (x_i, y_i) \text{ with } y_i = 1 \\ &(w^T x_i - b) \leq -1 \text{ for all } (x_i, y_i) \text{ with } y_i = -1; \end{aligned}$$

thus the condition $y_i(w^T x_i - b) \geq 1$ forces the features corresponding to positive and negative examples respectively to lie on one side of the hyperplane H_a given by $w^T x - b = 1$ and the opposite side of the hyperplane H_b given by $w^T x - b = -1$. The distance between these parallel hyperplanes is given by $2/\|w\|$; this distance is maximized (or equivalently $\|w\|^2$ is minimized) to find the maximum margin classifier which is the unique solution to the SVM optimization problem. Assuming that the examples are *linearly separable*—a set of labeled examples is linearly separable if there exists (w, b) such that $y_i(w^T x_i - b) \geq 1$ for *all* examples in the set—the resulting function is given by

$$f(x) = \text{sign}((w^\star)^T x - b^\star) \tag{9.4}$$

where $(w^\star, b^\star)$ is the solution to (9.3).

In general, labeled examples are not linearly separable which renders problem (9.3) infeasible; the standard approach to overcome this issue is to introduce slack variables ξ_i which allow for the possibility of misclassification of i^{th} example. The resulting optimization problem is given by

$$\begin{aligned} \underset{w,b,\xi_i}{\text{minimize}} \;\; & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \;\; & y_i(w^T x_i - b) \geq 1 - \xi_i,\; 1 \leq i \leq n \\ & \xi_i \geq 0, \end{aligned} \tag{9.5}$$

where $\xi_i \rangle 0$ indicates that example i was misclassified which is penalized in the objective function with weight C. For our case, problem formulation (9.5) is still not adequate, which we illustrate as follows. Note that in view of Remark 1, it is possible to have labeled examples (x_i, y_i) and (x_j, y_j) such that $x_i = x_j$ and $y_i \neq y_j$. That is, for the same feature set it is possible to have a blackout outcome for one load schedule and no-blackout outcome for another load schedule. The ratio ρ_s of blackout to no-blackout outcomes for a given feature set x_s is fixed by the true distribution of the load schedule. Now suppose a function f_0 is learnt such that $f_0(x_s) = -1$ (that is predicted outcome of x_s is a blackout), then the proportion of correctly predicted outcomes by f_0 to all the outcomes for examples that have x_s as feature set is ρ_s; no extra data will improve the performance better than ρ_s. Furthermore (9.5) places equal emphasis on positive and negative examples. These issues can be addressed either by improving the classifier (e.g. by extending the feature set to include more comprehensive information, such as load schedules) or by modifying the classification approach to produce more desirable results for our specific application of preventing blackouts. In the sequel we consider the latter as (in general) producing perfect classifications for complex systems is intractable.

Specifically, since choice of current threshold and guard time parameters are critical to ensure failsafe operation of the microgrid, it is necessary that classification scheme selects parameters that have no failures (no negatives) over all observed data. Ideally the classification scheme should not characterize any parameter set as safe if it leads to a negative outcome even for one specific schedule of loads. In this aspect, the standard classification criterion described in (9.5) is lacking since it does not emphasize exclusion of negative examples. Accordingly, we adapt the SVM by adding a constraint that ensures such exclusion by posing the following optimization problem,

$$\begin{aligned} \underset{w,b,\xi_i}{\text{minimize}} \;\; & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \;\; & y_i(w^T x_i - b) \geq 1 - \xi_i,\; 1 \leq i \leq n \\ & \xi_i \geq 0 \\ & \xi_i \leq 1 \text{ for } i \in \mathcal{N}, \end{aligned} \tag{9.6}$$

where $\mathcal{N}$ denotes the set of negative examples; this new formulation, the *one-sided SVM*, ensures through the added constraint $\xi_i \leq 1$ that the resulting classifier in (9.4) will label all negative

training examples correctly. Analogous to the standard SVM, additional benefits are gained by considering the dual formulation:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \min(\alpha_i, C) - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{subject to} \quad & \alpha_i \geq 0 \\ & \alpha_i \leq C \text{ for } i \in \mathcal{P} \\ & \sum_{i=1}^{n} y_i \alpha_i = 0. \end{aligned} \tag{9.7}$$

The dual formulation above is appealing for two reasons: first, it allows us to apply computationally efficient algorithms (e.g. sequential minimal optimization [101]); second, it allows us to consider nonlinear classification functions. Indeed the classifier that results from problem (9.6) is restricted to be linear with the number of parameters determined by the dimension p of the feature set. Specifically, the separating hyperplanes defined by w and b constitute $p + 1$ (equal to five in our case) unknown parameters. Better fitting of data can be accomplished by nonlinear classifiers. Consider a positive definite kernel $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to R$ where $\mathcal{X}$ is the original feature space. Given such a kernel, it can be shown that there exists a mapping $\phi : \mathcal{X} \to Z$ where Z is an inner-product space with an inner-product $\langle \cdot, \cdot \rangle$ such that for all elements $x, \tilde{x}$ in $\mathcal{X}$, $\langle \phi(x), \phi(\tilde{x}) \rangle = k(x, \tilde{x})$. In our case the original features space is $\mathcal{X} = R^4$. The dimension of Z can be much larger than that of $\mathcal{X}$ and possibly infinite. The classification problem is now cast in the new Hilbert space where hyperplanes are sought to separate the positive examples from the negative examples. The resulting optimization problem is described by (9.6) with x_i replaced by $\phi(x_i)$ with the appropriate inner-product

$$\begin{aligned} \underset{w,b,\xi_i}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(\langle w, \phi(x_i) \rangle - b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & \xi_i \leq 1 \text{ for } i \in \mathcal{N}. \end{aligned} \tag{9.8}$$

Even though problem (9.8) involves the mapping ϕ, we can efficiently solve it by using the dual formulation

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \min(\alpha_i, C) - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & \alpha_i \geq 0 \\ & \alpha_i \leq C \text{ for } i \in \mathcal{P} \\ & \sum_{i=1}^{n} y_i \alpha_i = 0 \end{aligned} \tag{9.9}$$

which is equivalent to the previous dual (9.7) except that it depends on the features only through the kernel function $k(x_i, x_j)$. In addition, determining how the optimal hyperplane classifies x

(more precisely $\phi(x)$), does not require knowledge of the map ϕ because the signed distance from the optimal hyperplane determined by $\langle w, \phi(x)\rangle$ and b can be written in terms of $\langle \phi(x), \phi(x_i)\rangle$ which in turn is given by the known kernel $k(x, x_i)$. Indeed the following can shown

Theorem 1 *For the optimal solution $(w^\star, b^\star)$ of (9.8), the following hold true*

$$\begin{aligned} \langle w^\star, \phi(x)\rangle &= \sum_{i=1}^{m} \alpha_i y_i \langle \phi(x_i), \phi(x)\rangle \\ b^\star &= \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} [\langle w^\star, \phi(x_i)\rangle - y_i] \end{aligned} \tag{9.10}$$

where $\mathcal{A} = \{i : 0\langle \alpha_i \langle C\}$ denotes the set of support vectors which lie on the interior of the $[0, C]$ constraint. Thus, given the optimal α_i's, we can compute the distance from the hyperplane for any new example x with an expression that depends only $k(x_i, x)$. In this chapter we use the Gaussian radial basis function (RBF) kernel given by

$$k(x, \tilde{x}) = \exp(-\|x - \tilde{x}\|^2 / 2\sigma^2). \tag{9.11}$$

which is a standard choice for kernel support vector machines.

In Figure 9.4 we construct a simple example using the one-sided SVM with kernels to learn a function on $\mathbb{R}^2$. First, we classify the training examples using the standard linear SVM—note that the separating hyperplane strikes a good balance between misclassification of positive and negative examples. Next, the one-sided SVM shifts this hyperplane so that we have no misclassification of negative examples while still attempting to classify the positive examples correctly. Finally, by using the RBF kernel, we find a nonlinear separator which captures a greater number of positive examples while still keeping the negative examples on one side.

9.3 Results and Discussion

In this section we examine the ability of our approach to find safe parameter settings in the simulated microgrid system described previously. We find that given enough data, the model is able to find a set of parameters that avoid cascading failures; we also compare the classification performance of the one-sided SVM to that of the standard SVM and show that on this classification task, our method is better at minimizing the number of false positives while still capturing a large percentage of the true positives.

We generate data by simulating our model assuming a time horizon of $T = 3$ time units with varying load profiles for a total of $n = 5000$ training examples. The electrical signals reside primarily at 60Hz (time period of 16.7 ms) and for long times (which can exceed 30 minutes) of normal operation of washers and dryers not much new information is obtained. To restrict undue and unwarranted computational and simulation burden we run the simulation for scaled down time units. In addition, when evaluating our methods we use 5-fold cross validation whereby we train the functions on 4/5 of the training examples and report classification results for the held out 1/5. We repeat this procedure 5 times and report error results that are the average over these 5 experiments.

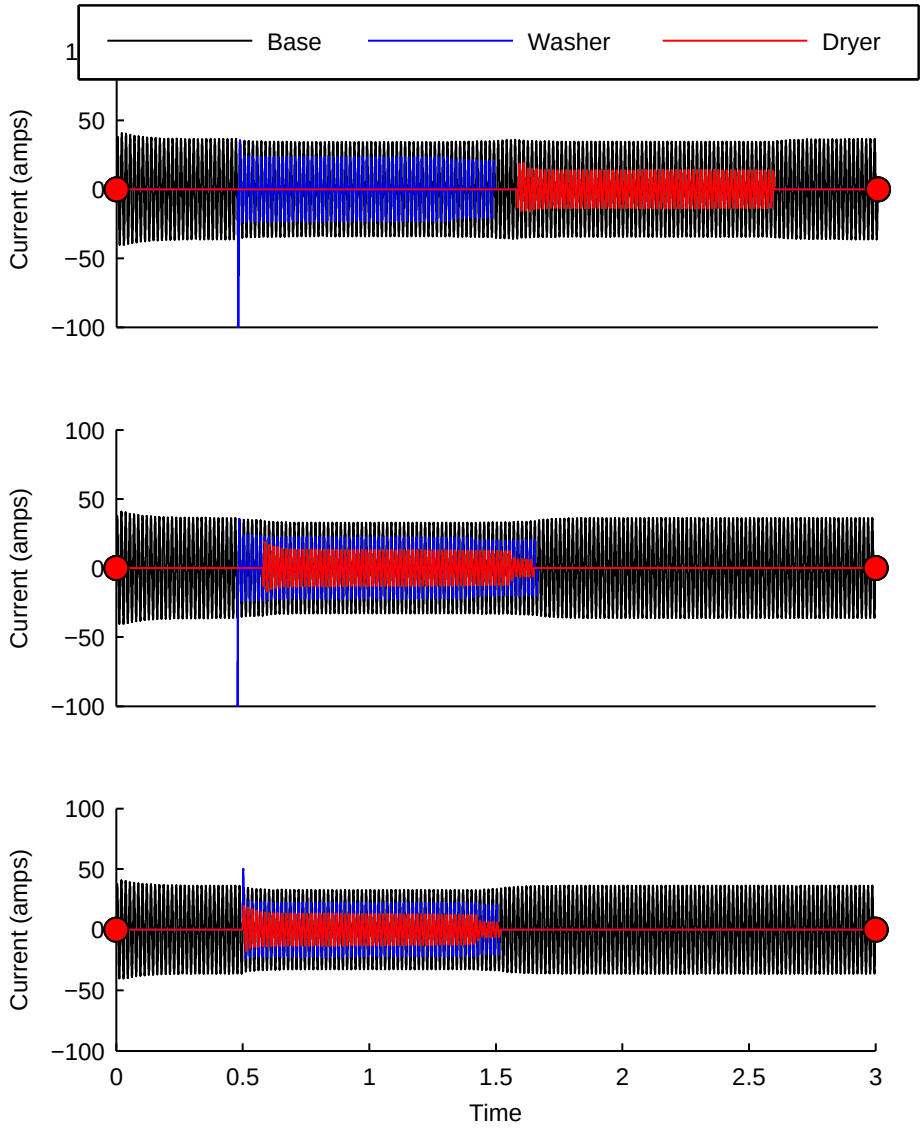

Figure 9.5: Example scenarios showing varying simulation conditions: washer and dryer do not overlap (top), washer and dryer overlap in steady state (middle) and washer and dryer overlap during the transient start up (bottom).

Table 9.1: Comparison of classification algorithms

SVM	Accuracy	TPR@95	TPR@99	TPR@99.9
Linear	90.78	86.58	72.45	56.48
Linear, OS	69.68	82.67	70.84	62.12
RBF Kernel	**93.08**	**90.42**	76.89	55.17
RBF Kernel, OS	82.28	87.82	**80.35**	**68.47**

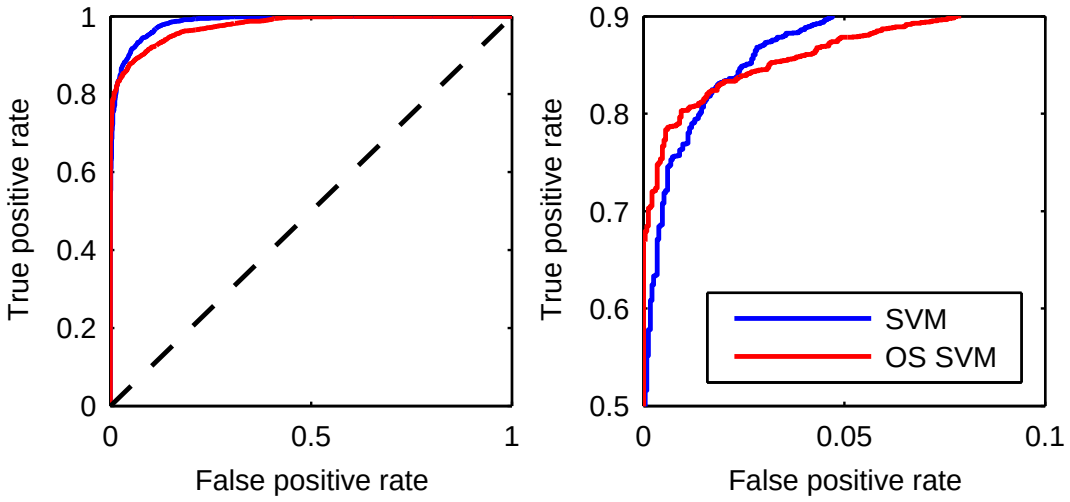

Figure 9.6: Comparison of classifiers on the entire range over the entire ROC curve (left) and focused on a low false positive rate (right).

In each simulation, we assume that the lights and the water heater remain on throughout the time horizon and vary the load by randomly choosing a start time for the washer and dryer. In choosing the washer and dryer start times there are three three possible scenarios, depicted in Figure 9.5—since we want to emphasize the worst case scenarios for stability, in 2/5 of the simulations sample the washer and dryer and start times uniformly from $[0.5, 0.55]$, in 2/5 from $[0.5, 0.7]$ and in 1/5 we choose the start times so they do not overlap. In all scenarios the dryer and washer remain on for 1 time unit. Note that in each of these scenarios not only is the steady state current demand varying depending on which devices are on at a particular time, but also the instantaneous current drawn is driven by transients from the loads and the droop characteristic[2] implemented by the inverters—all characteristics typical of a microgrid environment. Finally, in each simulated scenario we pick the parameter settings for each inverter uniformly at random with current thresholds in the range of $[10, 50]$ for inverter 1, $[18, 50]$ for inverter 2 and time limits between $[0.001, 0.04]$. We record which scenarios result in the failure of one or both of the inverters; our task is to classify these parameters in order to identify parameter settings that are stable under all load profiles.

The data generated was used to obtain solutions from four optimization problems—the stan-

[2]To stabilize the microgrid, the inverters source additional current in response to a drop in voltage, see e.g. [56] for a description of this common microgrid control scheme.

dard SVM (9.5) and one-sided SVM (9.6) are used to learn a linear and nonlinear classifier using the RBF kernel, with the kernel variants requiring solving the dual form (e.g. (9.7)). In Table 9.1 we compare SVM-based algorithms that classify the feature space in terms of no-blackout and blackout outcomes (corresponding to learning f_0). Note that the one-sided SVM with RBF kernel significantly outperforms the standard SVM approaches in maximizing the true positive rate (TPR) at a given false positive level. Here TPR refers to the fraction of no-blackout outcomes that were correctly predicted, and FPR refers to the fraction of blackout outcomes that were incorrectly predicted (as not blackout). As was described in the previous section, the one-sided SVM places larger emphasis on the negative examples which leads to lower a number of false positives at the expense of overall classification accuracy. For our application, ensuring viability of the electrical system is strongly desired as we would like to prevent cascading failures and microgrid collapse as much as possible. We see this tradeoff in Figure 9.6 which shows that the one-sided SVM achieves better performance on false positive rates less than 3% at the expense of performance on the rest of the curve.

We present results on safe parameter regions for each inverter (corresponding to learning f_1 and f_2) under different scenarios in Figure 9.7. In our setting, the classification function maps $\mathbb{R}^4 \rightarrow \{-1, 1\}$ and thus in order to visualize this function we fix two of the parameters and consider the classifier boundary as we vary the other two. The top row corresponds to fixing the parameters for inverter 2, first with aggressive settings $(0.01, 10)$ (top left) that will typically lead to that inverter failing. In this scenario, inverter 1 must source all of the current and thus the safe parameter settings are much higher that those required when inverter 2 has conservative settings $(0.05, 50)$ (top right). On the other hand, the situation depicted on the bottom row for inverter 2 is qualitatively different: due to the power sharing arrangement between the two inverters, it must be responsible for a significant portion of the load even when inverter 1 is fully operational. Thus even with low thresholds and guard times for inverter 1, the separating boundary is similar to the case where the inverter has high thresholds and guard times.

The framework presented demarcates regions of safe parameters and unsafe parameters. These regions can be utilized by a microgrid designer to arrive at breaker settings where the designer can incorporate data on the rating of the breaker while making decisions. Furthermore, the TPR and FPR curves can guide the level of robustness desired. Although the chapter has presented data from a two inverter system, the algorithms presented are scalable to more complex microgrids. Also, there are no restrictions on the topology of the network of inverters assumed. The two inverter system has the ease of presentation and can easily communicate the central ideas which is one of the reasons for the choice of this system.

The framework developed can be extended to expand the feature set. For example, microgrids are envisioned to have a communication layer, where controllers incorporating measurement devices can collect measured signals. These measurements can be added to the feature set which leads to a rich class of problems which can possibly enable of adaptation in real-time for microgrids.

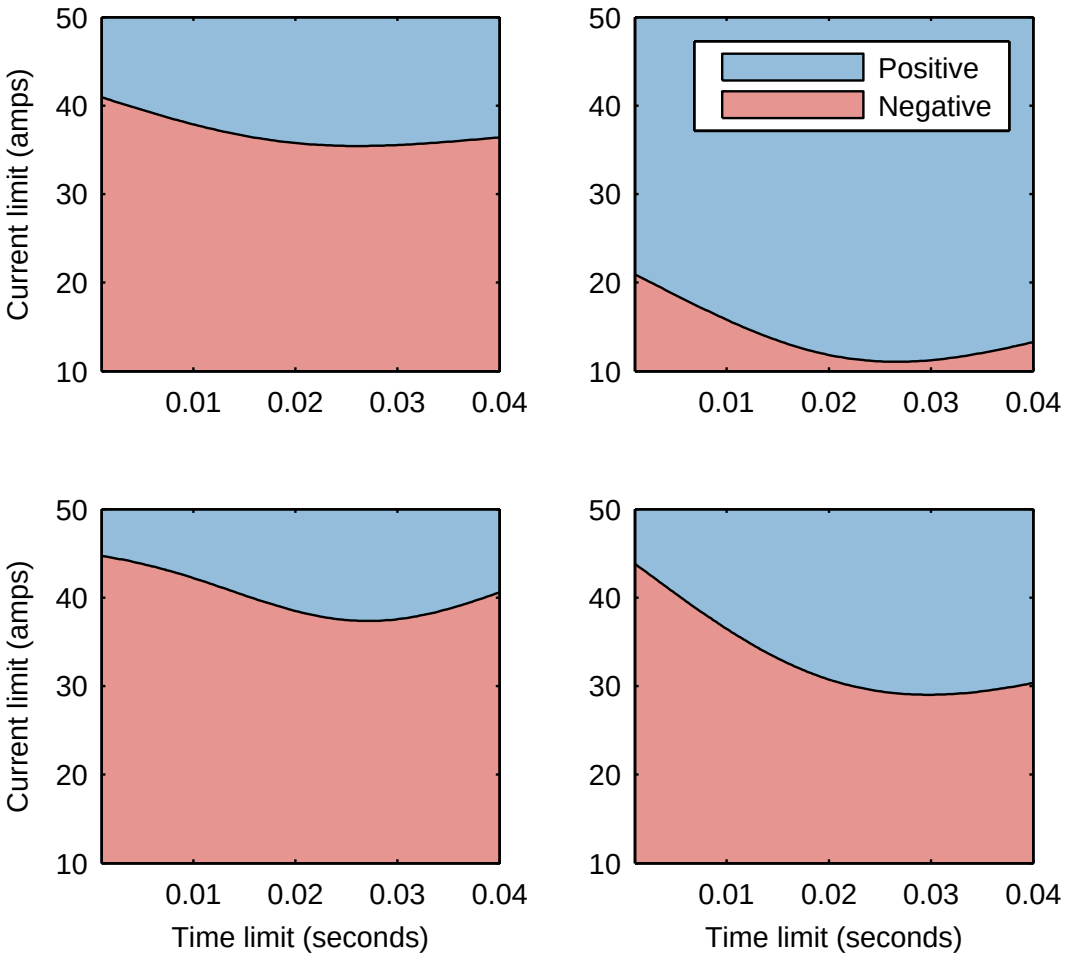

Figure 9.7: Learned safe parameter regions for each inverter under different scenarios. The top row shows safe parameters for inverter 1 when inverter 2 has thresholds $(0.01, 10)$ (top left) and $(0.05, 50)$ (top right). Bottom row, vice versa.

Chapter 10

Conclusion

This thesis examines the problem of developing scalable methods for convex optimization inspired by several applications arising from the development of the next-generation electrical grid as well as other problems in statistical machine learning. In order to meet the challenge of developing an efficient grid far more capable than the existing system, it is increasingly evident data-driven solutions will play a major role. In our applications, we use data to build better forecasting models for renewable integration, more intelligent analytics for improving energy end-uses and more robust microgrids. Although these problems are somewhat different in nature, each benefits from an abundance of data and thus are able to take advantage of statistical models learned efficiently with the tools provided by convex optimization.

As is frequently the case for real-world machine learning problems, the development and deployment of convex models in practice requires specialized algorithms. We develop three such algorithms in Part II of this thesis, with applications in our own work in energy as well as many other domains. The common thread between these three problems is a general Newton-like method which exploits sparsity for algorithmic benefit. For highly sparse problems (the common regime for high-dimensional statistical models) these methods deliver state-of-the-art performance often providing solutions for problems that would otherwise be impractical.

Although the development of specialize methods for particular problems solves immediate practical issues, there is a clear downside in terms of extensibility. This inspires the work of Part I, the development of scalable methods for *general* convex programming. By enhancing the functionality of the symbolic compiler to target new classes of algorithms, we are able to achieve orders of magnitude speed ups over existing approaches. This allows convex models to be more rapidly prototyped, developed and deployed on a much wider class of problems. Although the existing approach focuses exclusively on operator splitting technique, the same general principles could be applied to target new classes of algorithms as well as new architectures (e.g., GPU, distributed) likely enabling substantial gains in performance. It is our view that with these and many other developments, this approach of formulating convex problems in a high-level language will come to be the dominant paradigm for convex optimization.

Bibliography

[1] Torch 5. URL `http://torch5.sourceforge.net/`. 1

[2] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*. 1

[3] Samrachana Adhikari, Fabrizio Lecci, James T Becker, Brian W Junker, Lewis H Kuller, Oscar L Lopez, and Ryan J Tibshirani. High-dimensional longitudinal classification with the multinomial fused lasso. *arXiv preprint arXiv:1501.07518*, 2015. 3.3.3

[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015. 1

[5] Carlos M. Alaíz, Álvaro Barbero Jiménez, and José R. Dorronsoro. Group fused lasso. In *International Conference on Artificial Neural Networks and Machine Learning (ICANN)*, pages 66–73, 2013. 6, 6.1.1, 6.3.1

[6] N. Amjady, F. Keynia, and H. Zareipour. Short-term load forecast of microgrids by a new bilevel prediction strategy. *Smart Grid, IEEE Transactions on*, 1(3):286–294, 2010. 7.1

[7] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*, volume 1. Prentice Hall Englewood Cliffs, NJ, 1990. 5, 5.1

[8] Galen Andrew and Jianfeng Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007. 4.3

[9] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *Journal of Machine Learning Research*, 9:485–516, 2008. 4

[10] Álvaro Barbero and Suvrit Sra. Fast newton-type methods for total variation regularization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 313–320, 2011. 6, 6.2.2

[11] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009. 4.3, 6.3.1

[12] Anthony J Bell and Terrence J Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 7(6):1129–1159, 1995. 8.1.1

[13] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, 2011. 1

[14] Dimitri P Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM Journal on control and Optimization*, 20(2):221–246, 1982. 6.1.1

[15] Kevin Bleakley and Jean-Philippe Vert. The group fused lasso for multiple change-point detection. *arXiv preprint arXiv:1106.4199*, 2011. (document), 6, 6.3.1, 6.3.1, 6.3

[16] Peter Blomgren and Tony F Chan. Color TV: total variation methods for restoration of vector-valued images. *Image Processing, IEEE Transactions on*, 7(3):304–309, 1998. 6.2.2

[17] Vincent D Blondel and John N Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000. 5

[18] T Blumensath and M Davies. Shift-invariant sparse coding for single channel blind source separation. *SPARS*, 5:75–78, 2005. 8.1

[19] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011. 1, 2.2.3, 3.3.4, 5

[20] P.J. Brockwell. *Time Series Analysis*. Wiley Online Library, 2005. 7.2.1

[21] R. H. Byrd, J. Nocedal, and F. Oztoprak. An Inexact Successive Quadratic Approximation Method for Convex L-1 Regularized Optimization. *ArXiv e-prints*, September 2013. 4.2, 5.2.2

[22] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5): 1190–1208, 1995. 6.3.1

[23] Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006. 5

[24] Patrick L Combettes and Jean-Christophe Pesquet. A Douglas-Rachford splitting approach to nonsmooth convex variational signal recovery. *Selected Topics in Signal Processing, IEEE Journal of*, 1(4):564–574, 2007. 6.3.1

[25] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011. 6.2.2

[26] Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36 (3):287–314, 1994. 8.1.1

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):

273–297, 1995. 9.2

[28] Joachin Dahl and Lieven Vandenberghe. Cvxopt: A python package for convex optimization. In *Proc. eur. conf. op. res*, 2006. 2.1.3

[29] S. Darby. The effectiveness of feedback on energy consumption. Technical report, Environmental Change Institute, University of Oxford, 2006. 8.1

[30] ME Davies and CJ James. Source separation using single channel ica. *Signal Processing*, 87(8):1819–1832, 2007. 8.1

[31] Damek Davis and Wotao Yin. Convergence rate analysis of several splitting schemes. *arXiv preprint arXiv:1406.4834*, 2014. 3.1

[32] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 1

[33] Steven Diamond and Stephen Boyd. Convex optimization with abstract linear operators. 2015. 3.2.1

[34] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. 2015. 2.1, 3.3

[35] Alexander Domahidi, Eric Chu, and Stephen Boyd. Ecos: An socp solver for embedded systems. In *Control Conference (ECC), 2013 European*, pages 3071–3076. IEEE, 2013. 2.1.3, 3.3

[36] David L Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52 (4):1289–1306, 2006. 5

[37] F. Dörfler, M. R. Jovanovic, M. Chertkov, and F. Bullo. Sparsity-Promoting Optimal Wide-Area Control of Power Networks. *ArXiv e-prints*, July 2013. 5.3.2, 5.3.2

[38] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008. 3.2.2

[39] Krishnamurthy Dvijotham, Evangelos Theodorou, Emanuel Todorov, and Maryam Fazel. Convexity of optimal linear controller design. In *Proceedings of the Control and Decision Conference*, 2013. 5

[40] Krishnamurthy Dvijotham, Emanuel Todorov, and Maryam Fazel. Convex structured controller design. *CoRR*, abs/1309.7731, 2013. 5

[41] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004. 5

[42] Karen Ehrhardt-Martinez, Kat A Donnelly, and Skip Laitner. Advanced metering initiatives and residential feedback programs: a meta-review for household electricity-saving opportunities. In *American Council for an Energy-Efficient Economy*, 2010. 8.1

[43] Chih-Hai Fan, Jason L Speyer, and Christian R Jaensch. Centralized and decentralized solutions of the linear-exponential-gaussian problem. *Automatic Control, IEEE Transactions on*, 39(10):1986–2003, 1994. 5

[44] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Lib-

linear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008. 3.3.5

[45] C. Ferreira, J. Gama, L. Matias, A. Botterud, and J. Wang. A survey on wind power ramp forecasting. Technical report, Argonne National Laboratory (ANL), 2011. 7.4.2

[46] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. Feedback control of dynamic systems. *Addison-Wesley, Reading, MA*, 1994. 2.2.3

[47] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. 4

[48] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010. 3.3.5, 4.2

[49] Apostolos Gerasoulis. A fast algorithm for the multiplication of generalized hilbert matrices with vectors. *Mathematics of Computation*, 50(181):179–188, 1988. 1

[50] Zoubin Ghahramani and Michael I Jordan. Factorial hidden markov models. *Machine learning*, 29(2-3):245–273, 1997. 8.1.1

[51] Pontus Giselsson. Tight global linear convergence rate bounds for douglas-rachford splitting. *arXiv preprint arXiv:1506.01556*, 2015. 3.1

[52] Michael Grant, Stephen Boyd, and Yinyu Ye. *Disciplined convex programming*. Springer, 2006. 2.1

[53] Michael Grant, Stephen Boyd, and Yinyu Ye. CVX: Matlab software for disciplined convex programming, 2008. 1, 2.1

[54] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987. 5.2.1

[55] J. M. Guerrero, J. C. Vasquez, J. Matas, L. G. de Vicuña, and M. Castilla. Hierarchical control of droop-controlled ac and dc microgrids–A general approach toward standardization. *IEEE Trans. Ind. Electron.*, 58(1):158–172, 2011. 9.1, 9.1

[56] J.M. Guerrero, L.G. de Vicuna, J. Matas, M. Castilla, and J. Miret. A wireless controller to enhance dynamic performance of parallel inverters in distributed generation systems. *IEEE Trans. Power Electron.*, 19(5):1205–1213, Sept. 2004. ISSN 0885-8993. doi: 10.1109/TPEL.2004.833451. 2

[57] George William Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, 1992. 8.1

[58] T. Hong. Global energy forecasting competition, 2012. URL `http://www.gefcom.org`. 7.4

[59] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012. 5.2.1

[60] Patrik O Hoyer. Non-negative sparse coding. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 557–565. IEEE, 2002. 8.3.1

[61] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. Ravikumar. Sparse inverse covariace matrix

estimation using quadratic approximation. In *Neural Information Processing Systems*, 2011. 4.3, 4.3.2, 4.4.2

[62] Cho-Jui Hsieh, Mátyás A. Sustik, Inderjit S. Dhillon, and Pradeep D. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. *CoRR*, abs/1306.3212, 2013. 3.3.5, 4.2

[63] Institute for Electric Efficiency. Utility-scale smart meter deployments, plans, and proposals. Technical report, Institute for Electric Efficiency, 2012. 8.1

[64] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014. 1

[65] Nicholas A Johnson. A dynamic programming algorithm for the fused lasso and l 0-segmentation. *Journal of Computational and Graphical Statistics*, 22(2):246–260, 2013. 3.2.2, 3.3.3

[66] Hyungsul Kim, Manish Marwah, Martin F Arlitt, Geoff Lyon, and Jiawei Han. Unsupervised disaggregation of low frequency power measurements. In *SDM*, volume 11, pages 747–758. SIAM, 2011. 8.1.1

[67] Jingu Kim and Haesun Park. Fast active-set-type algorithms for L1-regularized linear regression. In *International Conference on Artificial Intelligence and Statistics*, pages 397–404, 2010. 6.1.3

[68] Kwangmoo Koh, Seung-Jean Kim, and Stephen P Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research*, 8(8): 1519–1555, 2007. 5

[69] J Zico Kolter and Tommi Jaakkola. Approximate inference in additive factorial hmms with application to energy disaggregation. In *International Conference on Artificial Intelligence and Statistics*, pages 1472–1482, 2012. 8.1.1

[70] J Zico Kolter, Siddarth Batra, and Andrew Y Ng. Energy disaggregation via discriminative sparse coding. In *Neural Information Processing Systems*, pages 1153–1161, 2010. 8.1, 8.1.1

[71] B. Lange, K. Rohrig, F. Schlögl, Ü. Cali, and R. Jursa. Wind power forecasting. *Renewable electricity and the grid*, pages 95–120, 2008. 7.1

[72] R Lau, R Persiano, and P Varaiya. Decentralized information and control: A network flow example. *Automatic Control, IEEE Transactions on*, 17(4):466–473, 1972. 5

[73] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Neural Information Processing Systems*, 2007. 6.1.3

[74] Michael S Lewicki and Terrence J Sejnowski. Learning overcomplete representations. *Neural computation*, 12(2):337–365, 2000. 8.1

[75] Fu Lin, M. Fardad, and M.R. Jovanovic. Design of optimal sparse feedback gains via the alternating direction method of multipliers. *Automatic Control, IEEE Transactions on*, 58 (9):2426–2431, 2013. 5, 5.1, 5.1, 5.3, 5.3.1, 5.3.1, 5.3.1

[76] Han Liu, John Lafferty, and Larry Wasserman. The nonparanormal: Semiparametric estimation of high dimensional undirected graphs. *The Journal of Machine Learning Research*, 10:2295–2328, 2009. 7.3.1

[77] Johan Löfberg. YALMIP: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004. 2.1

[78] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012. 1

[79] Andrew Makhorin. Glpk (gnu linear programming kit), 2008. 2.1.3

[80] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010. 1

[81] J Mendes, R.J. Bessa, H. Keko, J. Sumaili, V. Miranda, C. Ferreira, J. Gama, A. Botterud, Z. Zhou, and J. Wang. Development and testing of improved statistical wind power forecasting methods. Technical report, Argonne National Laboratory (ANL), 2011. 7.1

[82] M. Milligan, M. Schwartz, and Y. Wan. Statistical wind power forecasting models: results for us wind farms. Technical report, National Renewable Energy Laboratory, Golden, CO, 2003. 7.1

[83] C. Monteiro, R. Bessa, V. Miranda, A. Botterud, J. Wang, G. Conzelmann, et al. Wind power forecasting: state-of-the-art 2009. Technical report, Argonne National Laboratory (ANL), 2009. 7.1

[84] Jean-Jacques Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *CR Acad. Sci. Paris Sér. A Math*, 255:2897–2899, 1962. 3.2.2

[85] APS Mosek. The mosek optimization software. *Online at http://www. mosek. com*, 54, 2010. 2.1.3

[86] B. Neenan and J. Robinson. Residential electricity use feedback: A research synthesis and economic framework. Technical report, Electric Power Research Institute, 2009. 8.1

[87] Yu Nesterov and A Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992. 1

[88] Yurii Nesterov, Arkadii Nemirovskii, and Yinyu Ye. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM, 1994. 2.1.3

[89] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. *Advances in Neural Information Processing Systems*, 2002. 4

[90] Robert Nishihara, Laurent Lessard, Benjamin Recht, Andrew Packard, and Michael I Jordan. A general analysis of the convergence of admm. *arXiv preprint arXiv:1502.02009*, 2015. 3.1

[91] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Operator splitting for conic optimization via homogeneous self-dual embedding. *arXiv preprint arXiv:1312.3039*, 2013. 1, 2.1.3, 3.3, 3.3.4

[92] Henrik Ohlsson, Lennart Ljung, and Stephen Boyd. Segmentation of ARX-models using sum-of-norms regularization. *Automatica*, 46(6):1107–1111, 2010. 6.2.1, 6.3.2

[93] Peder Olsen, Figen Oztoprak, Jorge Nocedal, and Steven Rennie. Newton-like methods for sparse inverse covariance estimation. In *Advances in Neural Information Processing Systems 25*, pages 764–772, 2012. 4.3, 5.3.1

[94] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision research*, 37(23):3311–3326, 1997. 8.1.1

[95] Gurobi Optimization et al. Gurobi optimizer reference manual. *URL: http://www. gurobi. com*, 2012. 2.1.3, 3.3.5

[96] VY Pan, M Abu Tabanjeh, ZQ Chen, EI Landowne, and A Sadikou. New transformations of cauchy matrices and trummer's problem. *Computers & Mathematics with Applications*, 35(12):1–5, 1998. 1

[97] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):123–231, 2013. 2.2.3, 2, 3.2.2

[98] Oliver Parson, Siddhartha Ghosh, Mark Weal, and Alex Rogers. Non-intrusive load monitoring using prior models of general appliance types. In *26th AAAI Conference on Artificial Intelligence*, 2012. 8.1.1

[99] J.H.W. Penm, J.H. Penm, and RD Terrell. The recursive fitting of subset varx models. *Journal of Time Series Analysis*, 14(6):603–619, 2008. 7.2

[100] P. Pinson. Estimation of the uncertainty in wind power forecasting. *Centre Energétique et Procédés–Ecole des Mines de Paris Rue*, 23, 2006. 7.1

[101] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998. 9.2

[102] Luis F Portugal, Joaquim J Judice, and Luis N Vicente. A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables. *Mathematics of Computation*, 63(208):625–643, 1994. 6.1.3

[103] Xin Qi, Murti V Salapaka, Petros G Voulgaris, and Mustafa Khammash. Structured optimal and robust control with multiple criteria: A convex solution. *Automatic Control, IEEE Transactions on*, 49(10):1623–1640, 2004. 5

[104] Aaditya Ramdas and Ryan J Tibshirani. Fast and flexible admm algorithms for trend filtering. *arXiv preprint arXiv:1406.2082*, 2014. 3.3.3

[105] Tankred Rautert and Ekkehard W Sachs. Computational design of optimal output feedback controllers. *SIAM Journal on Optimization*, 7(3):837–852, 1997. 5.1, 5.1, 5.1, 5.1

[106] Michael Rotkowitz and Sanjay Lall. A characterization of convex problems in decentralized control. *Automatic Control, IEEE Transactions on*, 51(2):274–286, 2006. 5

[107] Sam T Roweis. One microphone source separation. *Advances in neural information*

processing systems, pages 793–799, 2001. 8.1

[108] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992. 3.3.3, 6.2.2

[109] Nils Sandell Jr, Pravin Varaiya, Michael Athans, and Michael Safonov. Survey of decentralized control methods for large scale systems. *Automatic Control, IEEE Transactions on*, 23(2):108–128, 1978. 5

[110] Mikkel N Schmidt and Morten Mørup. Nonnegative matrix factor 2-d deconvolution for blind single channel source separation. In *Independent Component Analysis and Blind Signal Separation*, pages 700–707. Springer, 2006. 8.1

[111] Mikkel N Schmidt and Rasmus Kongsgaard Olsson. Single-channel speech separation using sparse non-negative matrix factorization. In *Spoken Language Proceesing, ISCA International Conference on (INTERSPEECH)*, 2006. 8.1

[112] Simone Schuler, Ping Li, James Lam, and Frank Allgöwer. Design of structured dynamic output-feedback controllers for interconnected systems. *International Journal of Control*, 84(12):2081–2091, 2011. 5

[113] Simone Schuler, Ulrich Münz, and Frank Allgöwer. Decentralized state feedback control for interconnected systems with application to power systems. *Journal of Process Control*, 2013. 5

[114] P. Shah and P.A. Parrilo. H2-optimal decentralized control over posets: A state space solution for state-feedback. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 6722–6727, 2010. 5

[115] G. Sideratos and N.D. Hatziargyriou. An advanced statistical method for wind power forecasting. *Power Systems, IEEE Transactions on*, 22(1):258 –265, feb. 2007. ISSN 0885-8950. doi: 10.1109/TPWRS.2006.889078. 7.1

[116] Paris Smaragdis, Bhiksha Raj, and Madhusudana Shashanka. A probabilistic latent variable model for acoustic modeling. *Advances in models for acoustic processing, NIPS*, 148, 2006. 8.1.1

[117] Kyung-Ah Sohn and Seyoung Kim. Joint estimation of structured sparsity and output structure in multiple-output regression via inverse-covariance regularization. In *International Conference on Artificial Intelligence and Statistics*, pages 1081–1089, 2012. 4

[118] Kyung-Ah Sohn and Seyoung Kim. Joint estimation of structured sparsity and output structure in multiple-output regression via inverse-covariance regularization. In *Proceedings of the Conference on Artificial Intelligence and Statistics*, 2012. 4.3, 4.4.1

[119] S. A. Soliman and A. M. Al-Kandari. *Electrical Load Forecasting: Modeling and Model Construction*. Elsevier, 2010. 7.1

[120] Jos F Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999. 2.1.3

[121] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Machine Learning*, 4(4):267–373, 2011. 4

[122] J. Swigart and S. Lall. Decentralized control. *Networked Control Systems. Lecture Notes in Control and Information Sciences*, 406:179–201, 2011. 5

[123] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 3.3.1, 5

[124] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005. 3.3.3, 6

[125] Kim-Chuan Toh, Michael J Todd, and Reha H Tütüncü. Sdpt3a matlab software package for semidefinite programming, version 1.3. *Optimization methods and software*, 11(1-4): 545–581, 1999. 2.1.3

[126] Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1):387–423, 2009. 4.2, 4.2, 4.3, 5.2.2, 5.2.2

[127] Madeleine Udell, Karanveer Mohan, David Zeng, Jenny Hong, Steven Diamond, and Stephen Boyd. Convex optimization in Julia. In *High Performance Technical Computing in Dynamic Languages (HPTCDL), 2014 First Workshop for*, pages 18–28. IEEE, 2014. 2.1

[128] U.S. Energy Information Administration. 2009 RECS survey data, 2009. Available at `http://www.eia.gov/consumption/residential/data/2009/`. 8.3, 8.3.3

[129] Riaz A Usmani. Inversion of a tridiagonal jacobi matrix. *Linear Algebra and Its Applications*, 212:413–414, 1994. 6.1.2

[130] Various. *PJM Manual 19: Load Forecasting and Analysis*. PJM, 2012. Available at: `http://www.pjm.com/planning/resource-adequacy-planning/~/media/documents/manuals/m19.ashx`. 7.1

[131] Régis Vert and Jean-Philippe Vert. Consistency and convergence rates of one-class svms and related algorithms. *The Journal of Machine Learning Research*, 7:817–854, 2006. 9.2

[132] Bo Wahlberg, Stephen Boyd, Mariette Annergren, and Yang Wang. An ADMM algorithm for a class of total variation regularized estimation problems. *arXiv preprint arXiv:1203.1828*, 2012. 6.2.1

[133] Po-Wei Wang, Matt Wytock, and J Zico Kolter. Epigraph projections for fast general convex programming. *In Submission*, 2016. 1

[134] Shih-Ho Wang and E Davison. On the stabilization of decentralized control systems. *Automatic Control, IEEE Transactions on*, 18(5):473–478, 1973. 5

[135] Hans S Witsenhausen. A counterexample in stochastic optimum control. *SIAM Journal on Control*, 6(1):131–147, 1968. 5

[136] Matt Wytock and J Zico Kolter. A fast algorithm for sparse controller design. *arXiv preprint arXiv:1312.4892*, 2013. 1, 2

[137] Matt Wytock and J. Zico Kolter. Large-scale probabilistic forecasting in energy systems using sparse Gaussian conditional random fields. In *Decision and Control (CDC), 2013*

IEEE 52nd Annual Conference on, pages 1019–1024. IEEE, 2013. 3

[138] Matt Wytock and J. Zico Kolter. Sparse Gaussian conditional random fields: Algorithms, theory, and application to energy forecasting. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1265–1273, 2013. 1, 2, 4, 4.2

[139] Matt Wytock and J. Zico Kolter. Contextually supervised source separation with application to energy disaggregation. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. 3

[140] Matt Wytock, Srinivasa Salapaka, and Murti Salapaka. Preventing cascading failures in microgrids with one-sided support vector machines. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 3252–3258. IEEE, 2014. 3

[141] Matt Wytock, Suvrit Sra, and J. Zico Kolter. Fast Newton methods for the group fused lasso. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, 2014. 1, 2

[142] Matt Wytock, Po-Wei Wang, and J Zico Kolter. Convex programming with fast proximal and linear operators. *arXiv preprint arXiv:1511.04815*, 2015. 1

[143] Bo Xin, Yoshinobu Kawahara, Yizhou Wang, and Wen Gao. Efficient generalized fused lasso and its application to the diagnosis of alzheimers disease. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2163–2169, 2014. 3.3.3

[144] Junfeng Yang, Wotao Yin, Yin Zhang, and Yilun Wang. A fast algorithm for edge-preserving variational multichannel image restoration. *SIAM Journal on Imaging Sciences*, 2(2):569–592, 2009. 6.3.3

[145] A. Yazdani and R. Iravani. *Voltage-Sourced Converters in Power Systems*. Wiley, 2010. ISBN 9780470551561. 9.1

[146] Xiao-Tong Yuan and Tong Zhang. Partial gaussian graphical model estimation. *CoRR*, abs/1209.6419, 2012. 4, 4.3, 4.4, 4.4.1

[147] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010. 1

[148] I. Žežula. On multivariate gaussian copulas. *Journal of Statistical Planning and Inference*, 139(11):3942–3946, 2009. 7.3.1

[149] M. Ziefman and K. Roth. Nonintrusive appliance load monitoring: Review and outlook. *IEEE Transactions on Consumer Electronics*, 57(1):76–84, 2011. 8.1.1