INFERENCE AND LEARNING IN HIGH-DIMENSIONAL SPACES

by

Alejandro J. Weinstein

A thesis submitted to the Faculty and Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Electrical Engineering).

Golden, Colorado

Date: _____

Signed: _____

Alejandro J. Weinstein

Signed: _____

Dr. Michael B. Wakin

Thesis Advisor

Golden, Colorado

Date: _____

Signed: _____

Dr. Randy Haupt

Professor and Department Head

Department of Electrical Engineering and Computer Science

ABSTRACT

High-dimensional problems have received a considerable amount of attention in the last decade by numerous scientific communities. This thesis considers three research thrusts that fall under the umbrella of inference and learning in high-dimensional spaces. Each of these trusts aim to tackle the so called "curse of dimensionality" in a particular way.

The first research thrust focuses on recovering a signal whose amplitudes have been clipped. We present two new algorithms for recovering a clipped signal by leveraging the model assumption that the underlying signal is sparse in the frequency domain. Both algorithms employ ideas commonly used in the field of Compressive Sensing (CS); the first one is a modified version of Reweighted ℓ_1 minimization, and the second one is a modification of a simple greedy algorithm known as Trivial Pursuit. An empirical investigation shows that both approaches can recover signals with significant levels of clipping.

The second research thrust focuses on denoising a signal ensemble by exploiting sparsity both at the inter- and intra-signal level. The problem of signal denoising using thresholding estimators has received a significant amount of attention in the literature, starting in the 1990s when Donoho and Johnstone introduced the concept of wavelet shrinkage. In this approach, the signal is represented in a basis where it is sparse, and each noisy coefficient is thresholded by a parameter that depends on the noise level. We are extending this concept to the case where one has a set of signals, and the location of the nonzero coefficients for all these signals is the same. Our approach is based on a vetoing mechanism, where in addition to thresholding, the inter-signal information is used to "save" a coefficient that otherwise would be "killed". Our method achieves a better performance than independent denoising, and we quantify the expected value of this improvement. The results show a consistent improvement over the independent denoising, achieving results close to the ones produced by an oracle. We validate the technique using both synthetic and real world signals.

The third research thrust focuses on using sparse models in Reinforcement Learning (RL). In RL one is interested in designing an agent able to interact with a given environment. The agent observes its current state, and based on this observation takes an action. As a consequence, it gets a reward and transitions to a new state. The design objective is to conceive a policy, or control rule, that maximizes the aggregated rewards. When the number of states is large, the design of such policies requires the use of function approximations; it also requires the design of feature vectors, i.e., the design of a mapping between a state and a vector that summarizes the state. In this work we propose new algorithms that, by exploiting the structure of the functions to be approximated, simplify the design of the feature vectors. These methods are also more efficient than the existing ones in terms of computational complexity and the required number of samples. We evaluate the performance of the proposed methods empirically in a variety of environments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."
The Long Dark Tea-Time of the Soul, D. Adams.

This has been been a long journey, a little longer than what I initially expected, but nonetheless an exciting one. No doubt this has been possible thanks to all the people I've met along the way.

I had the fortune and pleasure to work under the supervision of Dr. Wakin during these last three years. I'm grateful for his guidance and unconditional support. Although it was not unusual for me to leave his office with more questions than answers, his insight always pushed me into the right direction. Thank you Dr. Wakin!

I began this journey a little over five years ago, when I met Dr. Moore. At the moment I was looking for a job, but he convinced me to join the graduate program (he didn't have to push too hard). One of the first things that Dr. Moore told me was that the "P" in PhD is for philosophy, not for practicality. These words helped me to keep the right perspective during all these years.

Although to be precise, this journey started earlier, when Karem was accepted as a graduate student at Mines, and we moved from Chile to Colorado. Karem has been my partner and best friend. Thank you Karem for your constant support and for putting up with me (I know it's not always easy). I'm looking forward to start a new stage of our life back in Chile. I love you Karem!

One doesn't join a PhD program to make friends. However, I suspect that in the future when I look back to this period of my life, closest to my heart will be all the friends I made. I met Borhan right at the beginning, and it didn't take us long to become good friends (I don't make friends easily Borhan, so don't take this lightly!). We shared countless conversations,

discussions, and backgammon games. As our group grew, I also shared many moments with Armin, Andrew, Farshad, and lately, with Dehui. Thank you guys!

I would also like to thank the members of my thesis committee, Dr. Hale, Dr. Moore, Dr. Tenorio, and Dr. Vincent. I interacted with each of you in different ways during my PhD, and you have been all very helpful.

And last but not least, I would like to say thanks to my family, specially to my mom. I know it has been hard for them to be apart from me. This, however, never has been an impediment to offer me their full support. I'll see you soon!

To Karem and

my beloved family

CHAPTER 1

INTRODUCTION

> "Remember kids, the only difference between screwing around and science is writing it down."
>
> Adam Savage

On August 8, 2000, exactly 100 years after David Hilbert presented 10 of the 23 famous *Hilbert's problems*[1] at the Paris conference of the International Congress of Mathematicians, David Donoho offered his take on some mathematical challenges for the 21$^{\text{st}}$ century. In his lecture, entitled "High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality" [34], he stated:

> "We are now in a setting where many very important data analysis problems are high-dimensional. Many of these high-dimensional data analysis problems require new or different mathematics. A central issue is the curse of dimensionality, which has ubiquitous effects throughout the sciences. This is countervailed by three blessings of dimensionality. Coping with the curse and exploiting the blessings are centrally mathematical issues, and only can be attacked by mathemetical means."

The "Curse of dimensionality" is a term, apparently coined by Richard Bellman [6], used to describe the kind of problems that arise when the number of dimensions involved in a problem is high. For instance, if we wish to approximate an s-times continuously differentiable function of D variables with a reconstruction error below ϵ, we need on the order of $\epsilon^{-D/s}$ function samples [25], i.e., the number of samples is exponential in D. These

[1] Hilbert's problems are a list of 23 problems in mathematics. These problems where all unsolved when Hilbert stated them in 1900, and they received a lot of attention by the mathematical community. Three problems remain unsolved.

type of phenomena are usually surprising because our intuition about the geometry of two and three dimensional spaces does not carry on to higher dimensions.

To illustrate how our intuition fails [63, Sec. 1.2.2] in high-dimensional spaces, consider the volume of a D-dimensional sphere of radius r

$$V_s(r) = \frac{\pi^{\frac{D}{2}} r^D}{\Gamma\left(1 + \frac{D}{2}\right)},$$

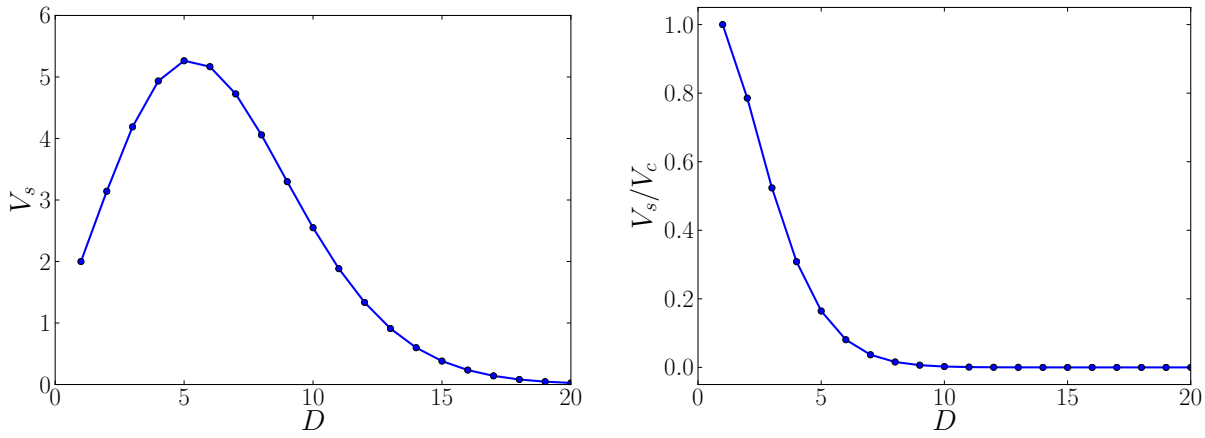
where Γ denotes the gamma function, and the volume of a D-dimensional circumscribed cube with volume $V_c(r) = (2r)^D$. The first surprise is that as the number of dimensions increases, the volume of the sphere goes to zero (see Fig. 1.1-a). The second surprise is how volumes distribute in high-dimensional spaces. In three dimensions and for a radius equal to 1, $V_s(r)/V_c(r) = \pi/6$, i.e., around half of the volume of the cube is contained in its corners. However,

$$\lim_{D\to\infty} \frac{V_s(r)}{V_c(r)} = 0.$$

This means that in high dimensions, most of the volume of a cube concentrates around its corners (see Fig. 1.1-b). This is commonly called the *empty space phenomenon*.

But in his lecture Donoho also stated that there are three blessings bestowed upon high-dimensional spaces. The first blessing is the "concentration of measure phenomenon," which encompasses the fact that a random variable that is a Lipschitz function of many independent variables is almost constant. The second blessing is the existence of asymptotic results, i.e., the kind of results obtained by letting the number of dimensions go to infinity. The third blessing is the approach to continuum, which is the fact that many times high-dimensional data is the discretization of an underlying continuous variable.

This thesis considers three research thrusts. Common to these thrusts is the focus on objects that exist in high-dimensional spaces. Two of the research thrusts are instances of inference problems and the last one is an instance of a learning problem.



(a) (b)

Figure 1.1: Empty space phenomenon. (a) Volume V_s of a D-dimensional sphere of radius one as a function of the number of dimensions D. As the number of dimensions increases the volume goes to zero. (b) Ratio between the volume of a D-dimensional sphere of radius one V_s and the volume of a circumscribed D-dimensional cube V_c as a function of the number of dimensions D. In high-dimensional spaces the ratio is close to zero, i.e., most of the volume of the cube is contained in its corners.

The first research thrust focuses on declipping a signal. We consider the problem of recovering a discrete-time signal for which a fraction of the samples are clipped, i.e., for the samples whose amplitudes are beyond the measurement range, we observe a saturation value instead of the actual value. We think of a discrete-time signal as a point in $\mathbb{R}^N$, where N is the number of samples. As customary in Compressive Sensing (CS), we cope with the curse of dimensionality by assuming a sparse signal model —in this case we assume that the signal is sparse in the frequency domain. As pointed out by Donoho [34, Sec. 9.2], it is now well understood that there are many functional classes, and sparsity is one of them, that allow to "crack" the curse of dimensionality.

The second research thrust focuses on denoising a signal ensemble. We consider the problem of estimating a set of signals from noisy observations. More precisely, we observe J discrete-time signals of length N. We think about these signals as J points in $\mathbb{R}^N$. In addition to using a sparse signal model, we cope with the curse of dimensionality by assuming an inter-signal model known as a Joint Sparsity Model (JSM) [5], where all the signals in

the ensemble share the same support.[2] This research thrust also exploits the "asymptotic blessing of dimensionality," since as the number of signals increases, we can approach the optimal behavior attained by an *oracle estimator*.

The third research thrust focuses on using sparse models in Reinforcement Learning (RL). In RL one is interested in designing an agent able to interact with a given environment. The agent observes its current state, and based on this observation takes an action. As a consequence, it gets a reward and transitions to a new state. The design objective is to conceive a policy, or control rule, that maximizes the aggregated rewards. When the number of states is large, the design of such policies requires the use of function approximations; it also requires the design of feature vectors, i.e., the design of a mapping between a state and a vector that summarizes the state. In this work we propose new algorithms that, by exploiting the structure of the functions to be approximated, simplify the design of the feature vectors. This methods are also more efficient than the existing ones in terms of computational complexity and the required number of samples. We evaluate the performance of the proposed methods empirically in a variety of environments.

1.1 Joint Denoising

A classic problem in signal processing is to remove the noise of an observed signal. This task, often call *signal denoising*, can be approached using different techniques. The classic approach is based on the theory of linear filters [85]. A more modern approach is based on the theory developed by Johnstone and Donoho in the 1990s, known as *thresholding estimators* [37, 67]. This theory considers transforming the signal of interest into a domain where the signal is sparse, typically the wavelet domain, processing each coefficient individually by applying a simple thresholding function to it, and transforming the signal back to its original domain. For signals that are sparse in the wavelet domain, this approach can be considered

[2]The support of a signal in a given domain is defined as the location of its nonzero coefficients.

as a form of adaptive smoothing, and has the advantage of preserving signal features, such as discontinuities, that linear filters typically destroy.

Although thresholding estimators have been improved in many ways—for instance, by adapting the threshold to the data, by developing translation-invariant thresholding estimators, by developing nondiagonal estimators, etc. [67, Secs. 11.2.3 and 11.4]—one aspect of this field that so far has been ignored is denoising a signal ensemble. This is a situation commonly observed when working with sensor arrays or sensor networks. If one observes a set of signals it is possible to naively denoise the signals independently; in this thesis, though, we exploit the structure that exists among the signals to obtain better results. Several authors [5, 27, 94, 100] have proposed signal models for a set of signals based in the sparsity patterns of the ensemble. Baron et al. [5] called this model a Joint Sparsity Model (JSM). From the three proposed JSM model variants, we assume that our signals satisfy the JSM-2 model, where all the signals share the same support.

Similarly to the independent denoising of a signal, our proposed method, called *joint denoising*, starts by transforming all the signals into a domain where the signals are sparse. Then it processes all the signal coefficients at a given location at once. If all the coefficients at that location are smaller than a threshold, they are all set to zero. If at least one coefficient at that location is larger than the threshold, *all* the coefficients at that location are kept. In the final step the signals are transformed back to the original domain. Since a "large" coefficient is able to "save" all the coefficient at a given location, we call this a vetoing scheme.

1.2 Declipping a Signal in Sparseland

In many practical situations, either because a sensor has the wrong dynamic range or because signals arrive that are larger than anticipated, it is common to record signals whose amplitudes have been clipped. Any method for restoring the values of the clipped samples

must—implicitly or explicitly—assume some model for the structure of the underlying signal. For example, one of the first attempts to "de-clip" a signal was the work of Abel and Smith [1], who assumed that the underlying signal had limited bandwidth relative to the sampling rate (i.e., that it was oversampled) and recovered the original signal by solving a convex feasibility problem. Godsill et al. [51] later tackled the de-clipping problem using a parametric model and a Bayesian inference approach.Along the same lines, Olofsson [76] proposed a maximum *a posteriori* estimation technique for restoring clipped ultrasonic signals based on a signal generation model and a bandlimited assumption.

Meanwhile, recent research in fields such as CS [19] has shown the incredible power of sparse models for recovering certain signal information. Many signals can be naturally assumed to be sparse in that they have few nonzero coefficients when expanded in a suitable basis; the name "Sparseland" has been used to describe the broad universe of such signals [43]. Although a typical CS problem involves an incomplete set of random measurements (as opposed to a complete—but clipped—set of deterministic samples), sparse models have made a limited appearance in the de-clipping literature. In particular, Gemmeke [49] et al. imputed noisy speech features by considering the spectrogram of the signal as an image with missing samples, represented the spectrogram in terms of an overcomplete dictionary, and used sparse recovery techniques to recover the missing samples. Using the model assumption that the underlying signal is sparse in an overcomplete harmonic dictionary, Adler et al. [2] later adapted the Orthogonal Matching Pursuit (OMP) [43] recovery algorithm from CS into a de-clipping algorithm that they call *constraint*-OMP. Studer and Baraniuk [91] considered a general model to restore an approximately sparse signal with sparse corruptions; one can formulate the declipping problem under their setting, though the theory applies only to small levels of clipping. Finally, Stoica et al. [90, and references therein], also used a sparse model for spectral estimation using irregularly sampled data. Their work, however, considered random samples, which are again fundamentally different from deterministic clipped samples.

In this thesis, we present two methods for de-clipping a signal under the assumption that the original signal is sparse in the frequency domain, i.e., that it can be represented as a concise sum of harmonic sinusoids. This model is general enough to embrace a wide set of signals that could be recorded from certain communication systems, resonant physical systems, etc. This model is also commonplace in the CS literature, particularly in settings involving random time-domain measurements. Although the measurements we consider are not random,[3] we do find that certain ideas from CS can be leveraged. In particular, we have modified several CS algorithms in an attempt to account for the clipping constraints. Among the methods that we have tried, the two with the best performance are a modified version of Reweighted ℓ_1 minimization [21] and a modified version of the Thresholding algorithm [43], also known as Trivial Pursuit (TP) [5]. This is surprising since TP, a very simple greedy algorithm, is one of the poorest performing algorithms in conventional CS problems [43]. We also show that, when tested on frequency sparse signals, these two methods outperform *constraint*-OMP.

1.3 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning [8, 92, 93]. It considers an *agent* that interacts with a given *environment*. The agent is able to take actions and to observe its current state. After taking an action, the agent observes its new state together with an immediate reward (this reward does not need to be positive). The goal is to design a policy, or control law, such that the sum of all the observed rewards is maximized. This problem is challenging because typically to maximize the total reward the agent needs to take actions that do not always look promising. This is why it is common to say that in RL "things need to get worse before they get better".

RL has been applied successfully in different domains. An early success case was *TD-Gammon* [96], a program that learned to play Backgammon. It is also common to use RL

[3] In fact they are "adversarial," in that clipping eliminates the samples with the highest energy content.

to solve classic control problems (e.g., controlling an inverted pendulum [92]), in robotics (autonomous helicopter [74], obstacle avoidance [70]), and in Operations Research (e.g., maintenance with limited resources and channel allocation in cellular systems [8]).

Finding or estimating the so-called value function is at the core of solving an RL problem. In its simplest form this is done by computing the value function for each state. In other words, the value function is stored as a look-up table. However, when the number of states is too large, or the state space is continuous, this approach is infeasible. As discussed in Sec. 5.2, this is overcome by using a function approximation scheme. Among the several function approximation architectures typically used in RL, the linear approximation approach is one of the most common ones. An important step in any linear approximation solution is the design of the feature vectors (or alternatively, the design of an approximation basis). Typically, this step involves designing these features or basis functions by hand, and it can become quite involved as the problem at hand becomes more complex. For this reason, researchers have focused on simplifying this step.

In this work we show how the use of sparse approximations [43] helps to alleviate the difficulties practitioners encounter when designing an approximation architecture. In particular, we propose to exploit the additional structure present in the action-value function to improve the generalization capabilities of the function approximation architecture. Our results shows that the proposed methods are able to approximate an optimal policy more efficiently, both in terms of the required number of samples an the required execution time.

1.4 Online Search Orthogonal Matching Pursuit

In the last chapter of this thesis we visit a classic algorithm, known as OMP, used to recover sparse signals. We show how this algorithm can be enhanced by incorporating ideas inspired by RL and some related concepts in the field of Artificial Intelligence.

Many areas of signal processing, including Compressive Sensing, image inpainting and

others, involve solving a sparse approximation problem. This corresponds to solving a system of equations $y = \Phi x$ where the matrix Φ has more columns than rows and x is a sparse vector. An important class of methods for solving this problem are the so called greedy algorithms, for which OMP is one of the classic representatives [98].

It is possible to think of greedy algorithms as instances of *search problems*. Karahanoğlu and Erdoğan [57] used the A* search method, a well known heuristic search algorithm for finding the shortest path between two nodes in a graph, to design a new greedy solver called A*OMP. This method stores the solution as a tree, where each node represents an index of the estimated support. At each iteration it selects, using a heuristic based on the evolution of the norm of the residue, which leaf node to expand. To avoid an exponential growing of the candidate solutions, this tree is pruned by keeping a relatively small number of leaves.

In this work we present a new greedy algorithm for solving sparse approximation problems. Like A*OMP, it frames the recovery of a sparse signal as a search instance. However, instead of using A* search which involves a monolithic planning stage, we formulate the problem as an *online* search, where the planning and execution stages are interleaved. This allows us to achieve a performance significantly better than OMP and similar to A*OMP while maintaining a reasonable computational load. Our simulations confirm this recovery performance with a computational speed $20\times$ faster than A*OMP and less than $2\times$ slower than OMP.

CHAPTER 2

SPARSE MODELS

"Who strive—you don't know how the others strive
To paint a little thing like that you smeared
Carelessly passing with your robes afloat,—
Yet do much less, so much less, Someone says,
(I know his name, no matter)—so much less!
Well, *less is more*, Lucrezia."

Robert Browning

In this chapter we introduce the concept of *sparse models* and show how these models have been used in signal processing and machine learning. We start with a few definitions that will be used through this thesis.

Definition 2.1 (ℓ_0 norm)
Given a vector[1] $x \in \mathbb{R}^N$, the ℓ_0 norm of x, denoted by $||x||_0$, is equal to the number of nonzero entries of x.

Definition 2.2 (Sparse signal)
The vector $x \in \mathbb{R}^N$ is K-sparse ($K \leq N$) if at most K entries of x are nonzero, i.e., if $||x||_0 \leq K$.

Note that calling the operator that returns the number of nonzero elements a "norm" is a misnomer, since this operator does not satisfy the *positive scalability* property.[2] Also, although the ℓ_0 norm is commonly stated as a definition, it can be derived by taking the

[1] Although we consider vectors with real entries, most definitions and results carry on to vectors with complex entries.

[2] The positive scalability property requires that $||\alpha x|| = |\alpha|\,||x||$, however, $||\alpha x||_0 = ||x||_0$.

limit of the ℓ_p norm when p goes to 0 from the right:

$$\begin{aligned}\lim_{p\to 0^+} ||x||_p^p &= \lim_{p\to 0^+} \sum_{i=1}^{N} |x_i|^p \\ &= \sum_{i=1}^{N} \lim_{p\to 0^+} |x_i|^p \\ &= \sum_{i=1}^{N} \begin{cases} 1 & |x_i| \neq 0, \\ 0 & |x_i| = 0. \end{cases}\end{aligned}$$

In practice, signals are rarely exactly sparse, but more commonly they can be well approximated by a sparse signal. For this reason, we introduce the concept of *approximately sparse signals*.

Definition 2.3 (Approximately sparse signal)
A signal is approximately K-sparse with precision ϵ if, given $\epsilon > 0$, $||x - x_K||_2 \leq \epsilon$, where x_K is equal to x at the K largest (in magnitude) entries and zero at the remaining locations. We call x_K the best K-approximation of x.

2.1 The Case for Sparse Models

We motivate the use of sparse models with the following example. Consider the 640×960 pixel image shown in Fig. 2.1-(a). This image can be represented by $640 \times 960 = 614400$ wavelet coefficients. These coefficients, sorted by magnitude, are shown in Fig. 2.2. We observe that only a small fraction of the wavelet coefficients have a relatively large magnitude (note that the ordinate-axis is in a logarithmic scale). If we approximate the original image using only the 10% largest wavelet coefficients—i.e., if we take the inverse wavelet transform of a set of coefficients equal to the 10% largest coefficients of the original image and equal to zero in the remaining locations—we get an image almost identical to the original (see Fig. 2.1-(b)). In other words, this image admits a very good sparse approximation.

(a) 640 × 960 original image

(b) Approximation using 10% of the largest (in magnituted) coefficients

Figure 2.1: Approximating an image sparse in the wavelet domain. The original image is approximately sparse in the wavelet domain (see Fig. 2.2), and it can be well approximated using only 10 % of the largest wavelet coefficients. (Source for the original image: http://bit.ly/10nOkHH)

The behavior exhibited by the image in the previous example is by no means an exception, but a common property of many natural signals [20]. This is in fact old news, and this property of natural signals has been successfully exploited in the design of compression standards such JPEG and MPEG [52]. What is new, however, is the use of sparse models—i.e., the assumption that the signals of interest can be represented efficiently in some basis—to design novel acquisition systems. In particular, this fact is the cornerstone of the new sensing paradigm known as Compressive Sensing (CS). In the remaining of the chapter we will summarize the main aspects of CS.

2.2 Compressive Sensing

The idea behind CS is that it is possible to design acquisition systems that measure a number of samples way lower than the ambient dimension of the signal of interest. In particular, it allows acquiring a signal using a sample rate significantly lower than the one

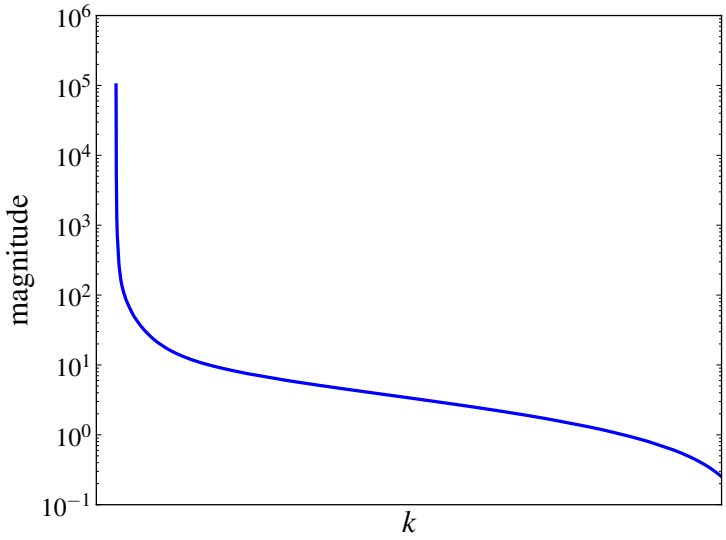


Figure 2.2: Daubechies wavelet coefficients of the image shown in Fig. 2.1-(a) sorted by magnitude. Note that the ordinate axis is in a logarithmic scale.

dictated by the Nyquist criterion.[3] Since CS allows reducing the number of samples that need to be acquired, it is particularly useful when measuring each sample is slow or expensive. For instance, CS has been successfully used in Magnetic Resonance Imaging, where it allows reducing the scanning time by a factor of five [65].

Let $x \in \mathbb{R}^N$ be a signal that can be represented in some domain (Fourier, wavelet, etc) as $x = \Psi\alpha$, where Ψ is an orthonormal $N \times N$ matrix, and $\alpha \in \mathbb{R}^N$ or $\alpha \in \mathbb{C}^N$ is a vector representing the coefficients of x in this domain. In CS we are interested in cases where α is a K-sparse or approximately K-sparse with $K \ll N$. The measurement process is modeled by a linear operator Φ, i.e.,

$$\begin{aligned} y &= \Phi x \\ &= \Phi\Psi\alpha, \end{aligned}$$

[3]This statement should not be read as if there is something wrong with the Nyquist criterion. The reason why CS allows reducing the sampling rate is that CS works under a different signal model than a classic acquisition system: CS assumes that signals are sparse in some domain, while the Nyquist criterion works under the assumption that signals are band-limited.

with Φ an $M \times N$ matrix, $M < N$. Letting $A = \Phi\Psi$, we can write

$$y = A\alpha.$$

To get back the signal of interest x given the observations y we need to solve a linear system of equations. Since the matrix A has more columns than rows ($M < N$), there are an infinite number of solutions. The only way to recover x among all these solutions is to exploit the fact that α is a sparse vector. We do this by selecting the sparsest vector α that satisfy the equation $y = A\alpha$, or equivalently, by solving the optimization problem

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \|\alpha\|_0 \\ \text{subject to} \quad & A\alpha = y. \end{aligned} \tag{P_0}$$

Unfortunately, solving the optimization problem P_0 is computationally unfeasible—it is in fact an NP-hard problem [15]. However, under appropriate conditions, it is still possible to recover α, and consequently x, by relaxing the ℓ_0 norm and replacing it by the ℓ_1 norm:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \|\alpha\|_1 \\ \text{subject to} \quad & A\alpha = y. \end{aligned} \tag{P_1}$$

Since this is a convex optimization problem [14], it is possible to solve it using a variety of numerical techniques.

Before reviewing the recovery conditions for x and some of the algorithms used to solve the optimization problem P_1, we will give some geometric intuition to justify the replacement of the ℓ_0 norm by the ℓ_1 norm.

Let us consider a general optimization problem

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \|\alpha\|_p \\ \text{subject to} \quad & A\alpha = y, \end{aligned} \tag{P_p}$$

where the p-norm of α is defined as $||\alpha||_p = \left(\sum_{i=1}^{N} |\alpha_i|^p\right)^{\frac{1}{p}}$, and let us define the p-ball of radius $r > 0$ as $B_p(r) = \{\alpha \in \mathbb{R}^N : ||\alpha||_p < r\}$.

The feasibility set $A\alpha = y$ corresponds to an $(N - M)$-hyperplane (assuming A is full-column rank). One can imagine the process of solving the optimization problem P_p as doing the following. Start with a p-ball $B_p(r)$ of very small radius and keep increasing its radius slowly—very much like inflating a balloon shaped as the p-ball. The point, or points, where the p-ball touches the hyperplane corresponding to the feasibility condition $A\alpha = y$ for the first time is the solution to the optimization problem.

Figure 2.3 shows p-balls for p set to 0, 1 and 2. The difference in shape of the balls implies that using different norms in the optimization problem P_p produces different solutions. The ℓ_0 norm, being the most "spiky" among all the norms, will find the sparsest solution; however, since it induces a non-convex ball, minimizing it is a hard problem to solve. Being isotropic the ℓ_2 norm will in general produce a dense solution. The ℓ_1 norm is the best convex approximation of the ℓ_0 norm. Since its shape is still "spiky," minimizing it produces, under appropriate conditions, the sparsest solution. Figure 2.4 shows an example that illustrates geometrically the difference between minimizing the ℓ_2 and the ℓ_1 norm. Minimizing the ℓ_2 norm corresponds to finding the intersection between a ball of spherical shape and a hyperplane. Minimizing the ℓ_1 norm corresponds to finding the intersection between a ball with the shape of a diamond and a hyperplane. Since the sphere is isotropic, it is unlikely that it will touch the hyperplane at a point where it is sparse. On the other hand, the "spiky" shape of the ℓ_1 ball promotes finding sparser solutions.

So far we have considered the case where the observations are noiseless. CS can also

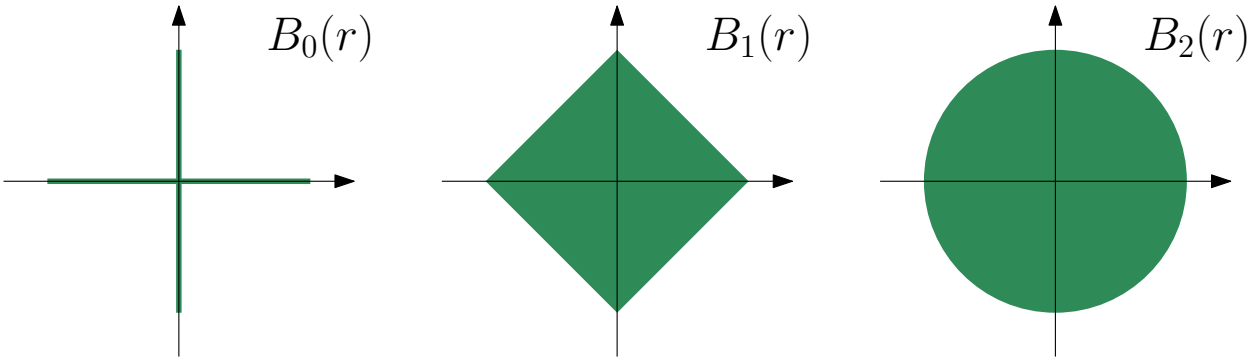


Figure 2.3: Three p-balls for p set to 0, 1 and 2. The difference in shape of the balls implies that using different norms in the optimization problem P_p produces different solutions. The ℓ_0 norm, being the most "spiky" among all the norms, will find the sparsest solution; however, since it induces a non-convex ball, minimizing it is a hard problem to solve. Being isotropic the ℓ_2 norm will in general produce a dense solution. The ℓ_1 norm is the best convex approximation of the ℓ_0 norm. Since its shape is still "spiky," minimizing it still produces, under appropriate conditions, the sparsest solution.

handle the more realistic case where one observes noisy measurements

$$y = \Phi x + \eta,$$

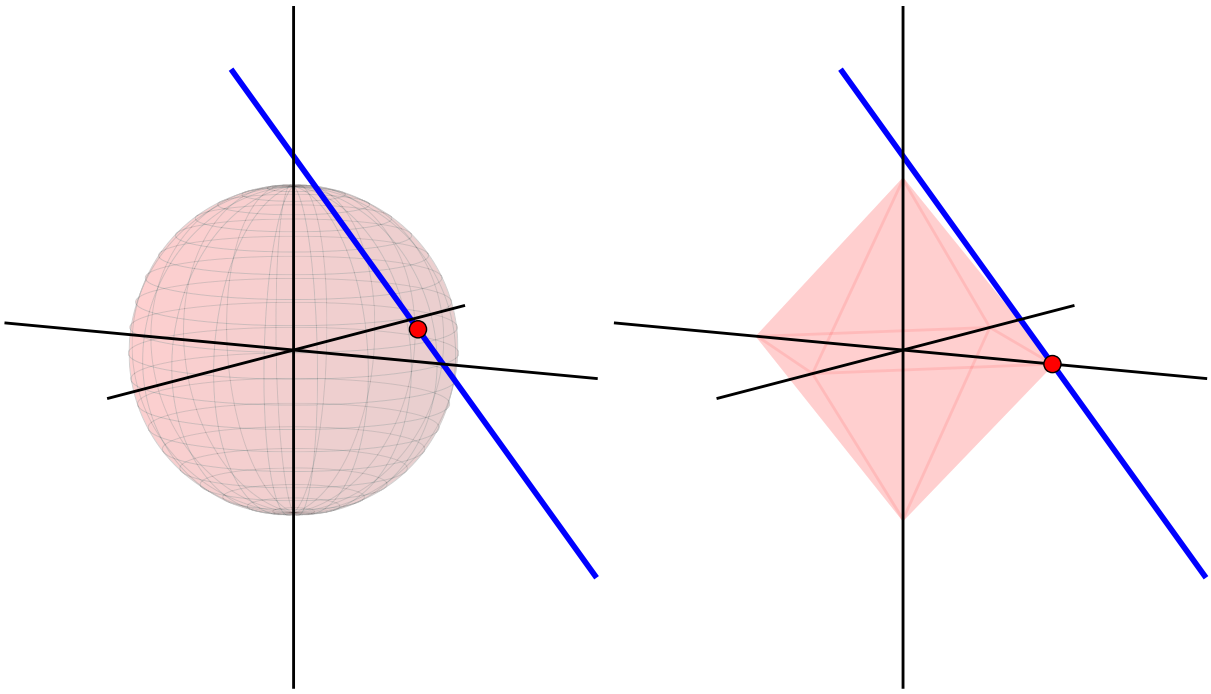
where η represents a bounded noise term[4] with $||\eta||_2 \leq \epsilon$. To recover x from noisy observations we only need to modify the convex optimization problem P_1 slightly[5]:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \|\alpha\|_1 \\ \text{subject to} \quad & ||y - A\alpha|| \leq \epsilon. \end{aligned} \tag{$P_{1\epsilon}$}$$

We can now review some conditions for which solving the optimization problem P_1 for the noiseless case, or $P_{1\epsilon}$ for the noisy case, allows the recovery of x. There are several approaches for stating these recovery conditions. Among these, we show the one based on the *Restricted Isometry Property* (RIP) [18]. Other commonly used recovery conditions include the *Null Space Property* [24, 28, 36] and the *Spark* [15, 35, 43].

[4]Although we consider the bounded noise case here, it is also possible to analyze the case where η is i.i.d. Gaussian. See [87] for an example of an analysis under such assumptions.

[5]Here and in the sequel, we use $||\cdot||$ to denote $||\cdot||_2$ when there is no risk of confusion.



(a) Solving P_p for $p = 2$ (b) Solving P_p for $p = 1$

Figure 2.4: Solving the optimization problem P_p for two different values of p. Minimizing the ℓ_2 norm corresponds to finding the intersection between a ball of spherical shape and a hyperplane. Minimizing the ℓ_1 norm corresponds to finding the intersection between a ball with the shape of a diamond and a hyperplane. Since the sphere is isotropic, it is unlikely that it will touch the hyperplane at point where the point is sparse. On the other hand, the "spiky" shape of the ℓ_1 ball promotes finding sparser solutions.

Definition 2.4 (Isometry constant [20])

For each integer $K = 1, 2, \ldots$, define the isometry costant δ_K of a matrix A as the smallest number such that

$$(1 - \delta_K) \, ||\alpha||_2^2 \leq ||A\alpha||_2^2 \leq (1 + \delta_K) \, ||\alpha||_2^2$$

holds for all K-sparse vectors α.

Loosely speaking, it is said that a matrix A satisfies the RIP of order K if δ_K is not too close to one.

Theorem 2.5 (Noiseless recovery [17, 20, 24])

Let $\alpha^\star$ be the solution to P_1, and let α_K denote the K largest in magnitude entries of α.

Assume that $\delta_{2K} < \sqrt{2} - 1$. Then $\alpha^\star$ satisfies

$$||\alpha^\star - \alpha||_2 \leq C_0 \frac{||\alpha - \alpha_K||_1}{\sqrt{K}} \text{ and}$$
$$||\alpha^\star - \alpha||_1 \leq C_0 \, ||\alpha - \alpha_K||_1$$

for some constant C_0.

Theorem 2.6 (Noisy recovery [17, 20])
Let $\alpha^\star$ be the solution to $P_{1\epsilon}$, and let α_K denote the K largest in magnitude entries of α. Assume that $\delta_{2K} < \sqrt{2} - 1$. Then $\alpha^\star$ satisfies

$$||\alpha^\star - \alpha||_2 \leq C_0 \frac{||\alpha - \alpha_K||_1}{\sqrt{K}} + C_1 \epsilon$$

for some constants C_0 and C_1.

The two previous theorems tell us under which conditions it is possible to recover a signal based on the isometry constant of the matrix A. Unfortunately, computing this constant for a given A is computationally unfeasible [4]. However, if we are able to design a CS system such that there is an element of randomness in the acquisition of the samples, it is then possible to guarantee that the RIP holds. In particular, if the $N \times N$ Ψ matrix represents an arbitrary orthobasis, if Φ is an $M \times N$ random matrix with i.i.d. entries drawn from a zero-mean Gaussian distribution, and if M is selected such that

$$M \geq CK \log(N/K)$$

for some constant C, then with overwhelming probability the $M \times N$ matrix $A = \Phi\Psi$ satisfy the RIP property of order $2K$ [20].

2.3 Solving P_1 and $P_{1\epsilon}$

Since both P_1 and $P_{1\epsilon}$ are convex optimization problems, it is possible to solve these problems using standard techniques such the interior point method [14]. It is also possible to design more ad-hoc algorithms that solve these problems by exploiting the particular structure they have. In this section we will review some of these algorithms.

To avoid notation clutter, and without loss of generality, for the remaining of this chapter we assume that signals are sparse in the ambient dimension, i.e., that Ψ is the identity matrix and that $x = \alpha$ is a sparse vector.

2.3.1 Subgradient

A common step to find a solution to an optimization problem is to compute the gradient of the cost function. To be able to deal with non-smooth cost functions, such the ℓ_1 norm, the concept of gradient can be extended to the subgradient. The subgradient of a convex function $f(x) : x \in \mathbb{R}^n \mapsto \mathbb{R}$, also known as subdifferential, is defined as the set [7, 11, 47]

$$\partial f(x) = \{z | f(\bar{x}) \geq f(x) + z^T(\bar{x} - x),\ \forall \bar{x} \in \mathbb{R}^n\}.$$

The subgradient can be used to find a minimizer of a convex function, as indicated by the following proposition.

Proposition 2.7 ([11, Sec. 3.1])
For any convex function $f(x) : x \in \mathbb{R}^n \to \mathbb{R}$, x^* is a minimizer of f if and only if $\mathbf{0} \in \partial f(x^*)$.

Example 2.8

Let $f(x) = |x|$ for $x \in \mathbb{R}$. The subgradient of f is

$$(\partial f(x))_i = \begin{cases} -1 & \text{if } x_i < 0, \\ 1 & \text{if } x_i > 0, \\ [-1, 1] & \text{if } x_i = 0. \end{cases}$$

Example 2.9

Let $f(x) = ||x||_2$ for $x \in \mathbb{R}^n$. The subgradient of f is

$$\partial f(x) = \begin{cases} \frac{x}{||x||} & \text{if } x \neq 0, \\ \{z : ||z|| \leq 1\} & \text{if } x = 0. \end{cases}$$

For $x \neq 0$, the expression follows from $||x|| = \sqrt{x^T x}$, $\partial x^T x = 2x$, and the chain rule [29, Sec. D.2.1]. For $x = 0$, we need to find the set $\{z : ||\bar{x}|| \geq z^T \bar{x} \ \forall \bar{x}\}$. By the Cauchy-Schwarz inequality, we have

$$z^T \bar{x} \leq |z^T \bar{x}| \leq ||\bar{x}|| \, ||z|| \, .$$

It follows that $||z|| \leq 1 \Rightarrow z^T \bar{x} \leq ||\bar{x}||$. On the other hand, if $||z|| > 1$ we can pick $\bar{x} = z$, in which case

$$z^T \bar{x} = ||z||^2 = ||\bar{x}||^2 > ||\bar{x}|| \, .$$

2.3.2 Least Absolute Shrinkage and Selection Operator

In statistics and machine learning it is common to work with an optimization problem closely related to the convex problem $P_{1\epsilon}$, known as *Least Absolute Shrinkage and Selection Operator* (LASSO) [97].

Consider, once again, the vector $y = Ax$, with $y \in \mathbb{R}^M$, $A \in \mathbb{R}^{M \times N}$, $x \in \mathbb{R}^N$, and

$M < N$. Given y, A, and λ, the LASSO estimates x as

$$\hat{x}^l = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \left(\frac{1}{2} ||Ax - y||_2^2 + \lambda ||x||_1 \right). \tag{2.1}$$

Using proposition 2.7, example 2.8, and the fact that $\partial ||Ax - y||_2^2 = A^T(Ax - y)$ [29, Sec. D.2.1], we can derive the necessary and sufficient conditions for the LASSO estimate:

$$\begin{aligned} \left(A^T(Ax - y)\right)_i - \lambda = 0 \text{ if } x_i < 0, \\ \left(A^T(Ax - y)\right)_i + \lambda = 0 \text{ if } x_i > 0, \\ -\lambda \leq \left(A^T(Ax - y)\right)_i \leq \lambda \text{ if } x_i = 0. \end{aligned}$$

This expression can be simplified to

$$\left(A^T(Ax - y)\right)_i = -\lambda \operatorname{sign}(x_i) \quad \text{if } x_i \neq 0, \tag{2.2}$$

$$\left|\left(A^T(Ax - y)\right)_i\right| \leq \lambda \text{ if } x_i = 0. \tag{2.3}$$

By denoting as a_i the i^{th} column of A, we can also write this expression as

$$a_i^T(Ax - y) = -\lambda \operatorname{sign}(x_i) \quad \text{if } x_i \neq 0, \tag{2.4}$$

$$\left|a_i^T(Ax - y)\right| \leq \lambda \text{ if } x_i = 0. \tag{2.5}$$

These expressions will be used later to derive the homotopy algorithm (see Sec. 2.3.5).

2.3.3 The LASSO and Soft-Thresholding Connection

There is a connection between the LASSO estimator and a soft-thresholding operation [43, Sec. 5.4]. This connection is often used to solve Eq. (2.1). To understand the origin of this relationship, consider the particular case where A is a unitary matrix.[6] Using the norm-preserving property of unitary matrices, we can write Eq. (2.1) as

$$\hat{x}^l = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \left(\frac{1}{2} ||x - b||_2^2 + \lambda ||x||_1 \right),$$

with $b = A^T y$. By decoupling the entries of x, this transformation allows to find the entries of $\hat{x}^l$ independently by solving

$$\hat{x}_i^l = \underset{x_i \in \mathbb{R}}{\operatorname{argmin}} \left(\frac{1}{2}(x_i - b_i)^2 + \lambda |x_i| \right). \tag{2.6}$$

We can find a minimizer of Eq. (2.6) by setting its subgradient equal to 0:

$$0 \in x_i - b_i + \lambda \begin{cases} -1 & x_i < 0, \\ 1 & x_i > 0, \\ [-1, 1] & x_i = 0. \end{cases}$$

This condition is satisfied by setting $x_i = b_i - \lambda$ if $x_i < 0$, $x_i = b_i + \lambda$ if $x_i > 0$, and $-\lambda \leq b_i \leq \lambda$ if $x_i = 0$. We can write

$$x_i = \operatorname{sign}(b)(|b| - \lambda)_+,$$

where $(\cdot)_+$ denotes $\max(0, \cdot)$.

In general the matrix A is not unitary. However, it is possible to design iterative

[6] A unitary matrix is a square matrix U such that $U^T U = I$. In particular, this implies that $||Ux|| = ||x||$.

algorithms that use the principle described in this section to find the LASSO estimator. See Chap. 6 of [43] for more details.

2.3.4 LARS

Least Angle Regression (LARS) is a model selection algorithm proposed by Efron et al. [42]. With a small modification, it can be used to efficiently find the whole regularization path of the LASSO estimator, that is, the LASSO estimation for all the values of λ.

As before, LARS considers the linear model

$$y = Ax,$$

where $y \in \mathbb{R}^N$, $x \in \mathbb{R}^M$, and $A \in \mathbb{R}^{M \times N}$.

Let $\mathcal{I} \subset \{1, 2, \ldots, N\}$ be the index set of active variables, and let

$$A_{\mathcal{I}} = [\cdots s_j a_j \cdots] \qquad j \in \mathcal{I}$$

be the M-by-$|\mathcal{I}|$ matrix, where a_j is the j^{th} column of A and $s_j = \pm 1$ is a sign variable, to be defined later.

Definition 2.10
Let u be a vector. We say u is *equiangular* with respect to the vectors v_j if all the angles between u and each v_j are equal and smaller than 90°.

Given $\mathcal{I}$, we can compute the unitary *equiangular* vector $u_{\mathcal{I}}$ with respect to the vectors a_j, $j \in \mathcal{I}$, as

$$u_{\mathcal{I}} = A_{\mathcal{I}} B_{\mathcal{I}} G_{\mathcal{I}}^{-1} \mathbf{1},$$

with

$$G_{\mathcal{I}} = A_{\mathcal{I}}^T A_{\mathcal{I}} \quad \text{and} \quad B_{\mathcal{I}} = \left(\mathbf{1}^T G_{\mathcal{I}}^{-1} \mathbf{1}\right)^{-\frac{1}{2}}.$$

To derive these expressions let us denote by $\tilde{u} = A_{\mathcal{I}}\alpha$ the equiangular vector with respect to x_j. For $\tilde{u}$ to be equiangular we need all the inner products between $\tilde{u}$ and a_j to be equal to the same constant, i.e., we need that[7] $A_{\mathcal{I}}^T \tilde{u} = \mathbf{1}$. It follows that

$$\begin{aligned} A_{\mathcal{I}}^T \tilde{u} &= A_{\mathcal{I}}^T A_{\mathcal{I}} \alpha = \mathbf{1} \\ \Rightarrow \alpha &= \left(A_{\mathcal{I}}^T A_{\mathcal{I}}\right)^{-1} \mathbf{1} = G_{\mathcal{I}}^{-1} \mathbf{1} \\ \Rightarrow \tilde{u} &= A_{\mathcal{I}}^T G_{\mathcal{I}}^{-1} \mathbf{1}. \end{aligned}$$

To normalize $\tilde{u}$ we compute its norm:[8]

$$\begin{aligned} ||\tilde{u}||^2 &= \left(A_{\mathcal{I}} G_{\mathcal{I}}^{-1} \mathbf{1}\right)^T \left(A_{\mathcal{I}} G_{\mathcal{I}}^{-1} \mathbf{1}\right) \\ &= \mathbf{1}^T \left(G_{\mathcal{I}}^{-1}\right)^T A_{\mathcal{I}}^T A_{\mathcal{I}} G_{\mathcal{I}}^{-1} \mathbf{1} \\ &= \mathbf{1}^T \left(G_{\mathcal{I}}^T\right)^{-1} G_{\mathcal{I}} G_{\mathcal{I}}^{-1} \mathbf{1} \\ &= \mathbf{1}^T G_{\mathcal{I}}^{-1} \mathbf{1} \\ \Rightarrow ||\tilde{u}|| &= \left(\mathbf{1}^T G_{\mathcal{I}}^{-1} \mathbf{1}\right)^{\frac{1}{2}} = B_{\mathcal{I}}^{-1}. \end{aligned}$$

The expression for $u_{\mathcal{I}}$ follows.

During each iteration, LARS proceeds as follows. Denote by $\hat{\mu}_{\mathcal{I}}$ the current LARS estimate. The correlation of the current residual is

$$\hat{c} = A^T \left(y - \hat{\mu}_{\mathcal{I}}\right).$$

[7] All we need is the inner products to be equal to a constant. For convenience we peak this constant to be 1.

[8] Here we use the fact that $G_{\mathcal{I}}$ is a symmetric matrix.

Define the maximum absolute correlation as

$$\hat{C} = \max_j \{|\hat{c}_j|\},$$

and the active set as

$$\mathcal{I} = \{j : |\hat{c}_j| = \hat{C}\}.$$

Note that, as explained below, by construction the condition $|\hat{c}_j| = \hat{C}$ is met by all the previously selected vectors. The correlation between the current equiangular vector $u_{\mathcal{I}}$ and *all* the vectors a_j is given by

$$b = A^T u_{\mathcal{I}},$$

with the sign variable s_j defined as $s_j = \text{sign}(\hat{c}_j)$.

The update to the current estimate $\hat{\mu}_{\mathcal{I}}$ is given by

$$\hat{\mu}_{\mathcal{I}+} = \hat{\mu}_{\mathcal{I}} + \hat{\gamma} u_{\mathcal{I}},$$

with

$$\hat{\gamma} = \min_{j \in \mathcal{I}^c}{}^{+} \left\{ \frac{\hat{C} - \hat{c}_j}{A_{\mathcal{I}} - b_j}, \frac{\hat{C} + \hat{c}_j}{A_{\mathcal{I}} + b_j} \right\},$$

where $\min^+\{\}$ denotes the minimum taken only over positive elements. The rationale for the update rule is as follows. For any $\gamma > 0$ let

$$\mu(\gamma) = \hat{\mu}_{\mathcal{I}} + \gamma u_{\mathcal{I}}.$$

The correlation for this vector is given by

$$c_j(\gamma) = x_j^T(y - \mu(\gamma)) = x_j^T(y - \hat{\mu}_{\mathcal{I}} - \gamma u_{\mathcal{I}}) = \hat{c}_j - \gamma b_j.$$

Note that for all $j \in \mathcal{I}$ (recall that by construction $a_j^T u_{\mathcal{I}} = A_{\mathcal{I}}$)

$$|c_j(\gamma)| = \hat{C} - \gamma A_{\mathcal{I}}.$$

In words, for all the vectors in the active set the correlation, as a function of γ, decreases by the same amount.

Now, we want to add to the active set $\mathcal{I}$ an index $j \in \mathcal{I}^c$ such that the correlation for the new index set $\mathcal{I} \bigcup \{j\}$ is the same, and at the same time, $\gamma > 0$ is the smallest possible value. We obtain the values of γ that translate into an equal correlation by solving

$$\begin{aligned} |c_j \gamma| = |\hat{c}_j - \gamma b_j| &= \hat{C} - \gamma A_{\mathcal{I}}, \\ \text{If } \hat{c}_j - \gamma b_j > 0 \Rightarrow \gamma &= \frac{\hat{C} - c_j}{A_{\mathcal{I}} - b_j}, \\ \text{If } \hat{c}_j - \gamma b_j < 0 \Rightarrow \gamma &= \frac{\hat{C} + c_j}{A_{\mathcal{I}} + b_j}. \end{aligned}$$

Thus, among all these values of γ that produce an equal reduction in the correlation, we choose the smallest positive one.

LARS is specified entirely in Algorithm 1.

2.3.5 The Homotopy Method

The homotopy method is a formulation similar to LARS that allows to efficiently compute the complete LASSO regularization path [48, Ch. 6.4.1], [38, Sec. 2]. Let x_λ be the minimizer of LASSO for a given λ, and let $I = \{i : x_\lambda(i) \neq 0\}$ be the support or *active set* of x_λ. The vector of *residual correlations* is given by

$$c = A^T(y - Ax_\lambda).$$

Algorithm 1 Least Angle Regression (LARS)

input: A, y, stopping criterion
  initialize: $\hat{\mu}_{\mathcal{I}} = 0$
  while not converged **do**
    $\hat{c} = A^T (y - \hat{\mu}_{\mathcal{I}})$
    $\hat{C} = \max_j \{|\hat{c}_j|\}$
    $\mathcal{I} = \{j : |\hat{c}_j| = \hat{C}\}$
    $A_{\mathcal{I}} = A[:, \mathcal{I}] \operatorname{diag}(\operatorname{sign}(\hat{c}(\mathcal{I})))$
    $G_{\mathcal{I}} = A_{\mathcal{I}}^T A_{\mathcal{I}}$
    $B_{\mathcal{I}} = \left(\mathbf{1}^T G_{\mathcal{I}}^{-1} \mathbf{1}\right)^{-\frac{1}{2}}$
    $u_{\mathcal{I}} = A_{\mathcal{I}} A_{\mathcal{I}} G_{\mathcal{I}}^{-1} \mathbf{1}$
    $b = A^T u_{\mathcal{I}}$
    $\hat{\gamma} = \min^+_{j \in \mathcal{I}^c} \left\{ \frac{\hat{C} - \hat{c}_j}{A_{\mathcal{I}} - b_j}, \frac{\hat{C} + \hat{c}_j}{A_{\mathcal{I}} + b_j} \right\}$
    $\hat{\mu}_{\mathcal{I}} = \hat{\mu}_{\mathcal{I}} + \hat{\gamma} u_{\mathcal{I}}$
  end while
output: $\hat{\mu}_{\mathcal{I}}$

Using these definitions we can write the optimality conditions for LASSO (see Eqs. (2.2) and (2.3)) as

$$c(I) = \lambda \operatorname{sign}(x_\lambda(I)), \tag{2.7}$$

$$|c(I^c)| \leq \lambda. \tag{2.8}$$

The homotopy method is an iterative algorithm that starts with an initial solution $x_0 = 0$. Plugging $x_0 = 0$ in Eq. (2.8) we get that the null vector is a solution as far as

$$|(A^T(y - Ax_0))_i| = |(A^T y)_i| \leq \lambda \Leftrightarrow \lambda \geq \|A^T y\|_\infty.$$

It is known that the LASSO solution is piecewise linear [3]. During iteration k, let the linear segment of the solution be[9]

$$x_{k+1}(\gamma_k) = x_k + \gamma_k d_k,$$

[9]Note that we work with x here, while the original LARS works with y.

where d_k is an $|I|$-sparse vector such that all the inner products between the active columns of A and the vector $A_{I_k} d_k$ have the same magnitude and sign equal to the current residual correlations. We compute the active entries of d_k by solving

$$A_{I_k}^T A_{I_k} d_k(I_k) = \text{sign}(c_k(I_k)), \tag{2.9}$$

and entries on I^c are set to 0. This is not an arbitrary selection. The vector d_k is used to construct the linear segment of the LASSO solution. For the linear segment to be a solution, we need to satisfy the optimality conditions. Assume that the value of γ_k is correct (we will derive conditions on γ_k later) and that x_{k-1} is a solution. For the entries on the support, we will see that this choice of d_k implies that the correlation $c(I)$ decreases by the same amount for all $i \in I$, meaning that the equality of condition (2.7) is satisfied along the solution path. We will see later that the conditions on γ_k also guarantee that condition (2.8) is satisfied for the inactive entries.

The solution path is valid as far as the optimality conditions are met. We find the breaking points of the linear segments by finding when either (2.7) or (2.8), as a function of γ, are violated. We need to consider two cases, one for each condition:

(i) The magnitude of the correlation for the inactive entries exceeds the value λ.

The residual correlation as a function of γ is

$$\begin{aligned} c_{k+1}(\gamma) &= A^T(y - Ax_{k+1}(\gamma)) \\ &= A^T(y - A(x_k + \gamma d_k)) \\ &= A^T(y - Ax_k) - \gamma A^T A d_k \\ &= c_k - \gamma A^T A d_k. \end{aligned}$$

Note that since d_k is the solution of Eq. (2.9), the entries of c_k supported on I_k satisfy

$$c_k(\gamma) = c_k - \gamma \operatorname{sign}(c_k).$$

This means that to satisfy the optimality conditions (2.7) and (2.8) as x_k changes linearly, λ must change according to

$$\lambda_{k+1}(\gamma) = \lambda_k - \gamma_k.$$

For a given entry $i \in I^c$, condition (2.8) will fail when

$$|c_k(\gamma_k, i)| = |c_k(i) - \gamma_k A^T A d_k(i)| = \lambda_k - \gamma_k,$$

which happens when

$$c_k(i) - \gamma_k A^T A d_k(i) = \lambda_k - \gamma_k \Rightarrow \gamma_k(i) = \frac{\lambda_k - c_k(i)}{1 - a_i^T A_I d_k(I)},$$

or when

$$c_k(i) + \gamma_k A^T A d_k(i) = \lambda_k + \gamma_k \Rightarrow \gamma_k(i) = \frac{\lambda_k + c_k(i)}{1 + a_i^T A_I d_k(I)}.$$

Among all the positive $\gamma_k(i)$, $i \in I^c$, we pick the one that makes the optimality condition to fail first, i.e., we pick the smallest one:

$$\gamma_k^+ = \min_{i \in I^c}{}^+ \left\{ \frac{\lambda_k - c_k(i)}{1 - a_i^T A_I d_k(I)}, \frac{\lambda_k + c_k(i)}{1 + a_i^T A_I d_k(I)} \right\}.$$

(ii) The sign of the correlation for an active variable changes sign.

As the solution changes linearly, it is also possible that an active variable fails to satisfy

condition (2.7). This happens when $x_k(\gamma, i)$, $i \in I$, change signs, i.e., when

$$x_k(\gamma, i) = x_k(i) + \gamma_k d_k(i) = 0 \Rightarrow \gamma_k(i) = -\frac{x_k(i)}{d_k(i)}.$$

Again, over all these positive γs, we choose the smallest one:

$$\gamma_k^- = \min_{i \in I}{}^+ \left\{ -\frac{x_k(i)}{d_k(i)} \right\}.$$

Finally, for γ_k we choose the smallest γ between the two obtained in steps (i) and (ii):

$$\gamma_k = \min\{\gamma_k^+, \gamma_k^-\},$$

and if $\gamma_k^+ < \gamma_k^-$, $I = I \cup \{i^+\}$, and if $\gamma_k^- \leq \gamma_k^+$, $I = I \backslash \{i^-\}$.

Note that omitting step (ii) makes the solution equal to LARS.

Why do we always choose the smallest γ? Recall that the solution is piecewise linear. The breakpoints of this piecewise linear path happen when an index enters or leaves the active set. This event occurs when the linear path reaches a point where the optimality conditions stop being satisfied. We pick the smallest γ because we need to find the very first time that any of the optimality condition starts being violated.

The homotopy method is specified entirely in Algorithm 2.

2.3.6 Group LARS/LASSO

As before, consider the vector $y = Ax$, with $y \in \mathbb{R}^M$, $A \in \mathbb{R}^{M \times N}$, $x \in \mathbb{R}^N$, and $N > M$. If in addition to being sparse it is known that the nonzero elements of x appear in groups, it is possible to improve the performance of an estimator by leveraging this extra knowledge. The following method, known as Group LARS/LASSO, is designed with this goal in mind [104].

Algorithm 2 Homotopy

input: A, y, stopping criterion

$x_0 = 0$

$c_0 = A^T y$

$\lambda_0, I_0 = \max_i(|c_0(i)|)$

$k = 0$

while not converged **do**

$c_k = c_0 - A^T A x$

$d_k = \mathbf{0}$

$d_k(I_k) = \left(A_{I_k}^T A_{I_k}\right)^{-1} \operatorname{sign}(c_k(I_k))$

$\gamma_k^+, i_k^+ = \min_{i \in I^c}^+ \left\{ \frac{\lambda_k - c_k(i)}{1 - a_i^T A_I d_k(I)}, \frac{\lambda_k + c_k(i)}{1 + a_i^T A_I d_k(I)} \right\}$

$\gamma_k^-, i_k^- = \min_{i \in I}^+ \{-x_k(i)/d_k(i)\}$

if $\gamma_k^+ < \gamma_k^-$ **then**

$\gamma_k = \gamma_k^+$

$I_{k+1} = I_k \cup \{i_k^+\}$

else

$\gamma_k = \gamma_k^-$

$I_{k+1} = I_k \backslash \{i_k^-\}$

end if

$x_{k+1} = x_k + \gamma_k d_k$

$\lambda_{k+1} = \lambda_k - \gamma_k$

$k = k + 1$

end while

output: x_j, λ_j for $j = 1, \ldots, k$

Partition x and A in J components[10] $x = [x_1^T \; x_2^T \; \cdots \; x_J^T]^T$ and $A = [A_1 \cdots A_J]$, where $x_j \in \mathbb{R}^{n_J}$, $A_j \in \mathbb{R}^{m \times n_J}$, $n_J = n/J$, $j = 1, \ldots, J$. Given y and A, group LASSO estimates x as [104] [45, Sec. 2.3.1]

$$\hat{x}^{gl} = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \left(\frac{1}{2} ||Ax - y||_2^2 + \lambda \sum_{j=1}^{J} ||x_j||_2 \right).$$

Using proposition 2.7 and example 2.9 we derive the necessary and sufficient conditions

[10] We assume that all the components have equal size and that $n_J = n/J$ is an integer.

for the group LASSO estimate as:

$$\left(A^T(Ax-y)\right)_j + \lambda \frac{x_j}{||x_j||} = 0 \text{ if } x_j \neq \mathbf{0},$$
$$-\lambda \leq \left|\left|\left(A^T(Ax-y)\right)_j\right|\right| \leq \lambda \text{ if } x_j = \mathbf{0}.$$

Note that more generalized versions of group LASSO consider groups of vectors of different lengths. For instance, Yuan and Lin [104] consider groups of vectors of length n_j, together with a weighted ℓ_2-norm $||x_j||_{K_j}$ in the cost function. They set the weight matrix to $K_j = n_J I_{n_j}$, which implies that the term $\sqrt{n_j}$ appears in the optimality conditions[11] [104, Eq. (2.2)].

The first step for dealing with the idea of group sparsity is to extend the concept of inner product between two vectors to the case of a group of vectors. We define the angle between the residue r and a group of vectors A_j as the angle between r and the projection of r onto the column-span of A_j. In general, this projection is given by [72]

$$r_{A_j} = A_j \left(A_j^T A\right)^{-1} A_j^T r,$$

where A_j is a matrix with the columns of A corresponding to group j. If we assume that these columns are orthonormal, i.e., that $A_j^T A_j = I$, the expression above simplifies to

$$r_{A_j} = A_j A_j^T r.$$

[11] $\partial ||x||_K = \partial\sqrt{x^T K x} = Kx/||x||_K$. If $K = pI$, $Kx/||x||_K = px/\sqrt{p}\,||x|| = \sqrt{p}x/||x||$.

The norm of this vector is given by

$$\begin{aligned}
\left|\left|r_{A_j}\right|\right|^2 &= \left\langle r_{A_j}, r_{A_j} \right\rangle \\
&= \left\langle A_j A_j^T r, A_j A_j^T r \right\rangle \\
&= r^T A_j \underbrace{A_j^T A_j}_{I} A_j^T r \\
&= r^T A_j A_j^T r \\
&= \left\langle r, r_{A_j} \right\rangle .
\end{aligned}$$

Thus, the angle between r and its projection onto the colum-span of A_j is given by

$$\begin{aligned}
\cos^2\theta &= \frac{\left\langle r, r_A \right\rangle^2}{\left|\left|r\right|\right|^2 \left|\left|r_A\right|\right|^2} \\
&= \frac{\left\langle r, r_A \right\rangle^2}{\left|\left|r\right|\right|^2 \left\langle r, r_{A_j} \right\rangle} \\
&= \frac{r^T A_j A_j^T r}{\left|\left|r\right|\right|^2} \\
&= \frac{\left|\left|A_j^T r\right|\right|^2}{\left|\left|r\right|\right|^2}.
\end{aligned}$$

Since the solution of group LASSO is not piecewise-linear [104], it is useful to compute the group LARS solution instead. Group LARS is an iterative algorithm that starts by setting $x^0 = 0$, $r_0 = y$ and $k = 1$ [104]. The active set is initialized with the index of the group most correlated to the current residue:

$$I_0 = \left\{ \underset{j=1,\ldots,J}{\operatorname{argmax}} \left|\left|A_j^T r_0\right|\right| \right\}.$$

At each iteration, the new coefficient is computed as

$$x_{k+1} = x_k + \gamma d_k,$$

where d_k is a vector such that the magnitude of all the active entries of the residual correlations decline equally as γ increases. The vector of residual correlations at a given iteration is defined as

$$\begin{aligned} c_k &= A^T(y - Ax_k) \\ &= A^T r_k, \end{aligned}$$

with $r_k = y - Ax_k$. The value of this correlation changes as a function of γ as

$$\begin{aligned} c_{k+1} &= A^T(y - Ax_{k+1}) \\ &= A^T(y - A(x_k + \gamma d_k)) \\ &= A^T(y - Ax_k) - \gamma A^T A d_k \\ &= c_k - \gamma A^T A d_k. \end{aligned}$$

By setting d_k as a sparse vector that on its support I_k satisfies

$$A_{I_k}^T A_{I_k} d(I_k) = A_{I_k}^T r_k,$$

we can write the entries of c_{k+1} supported on I_k as

$$c_{k+1}(i) = (1 - \gamma)c_k(i), \text{ for } i \in I_k.$$

Thus, in effect, all the active entries of the correlation vector decline by the same amount.

To find the next index to add to the active set we compute, for each index not in the active set, the value of γ such that the value of the group angle between the element not in the support and the value of the group angle for items in the support are equal. In general, the group angle between a group A_j and the residue as a function of γ is

$$\left|\left|A_j^T r_k(\gamma)\right|\right|^2 = \left|\left|A_j^T(r_k - \gamma A d_k)\right|\right|^2.$$

Then, for each $j \in I_k^c$, we find the value of γ such that the group angle is the same as the angles in the active set by solving:

$$\left|\left|A_j^T(r_k - \gamma_j A d_k)\right|\right|^2 = \left|\left|A_{j'}^T(r_k - \gamma_j A d_k)\right|\right|^2, \tag{2.10}$$

for an arbitrarily[12] chosen $j' \in I_k$. Expanding Eq. (2.10) we get[13]

$$\begin{gathered}
\left|\left|A_j^T r_k - \gamma_j A_j A d_k\right|\right|^2 = \left|\left|A_{j'}^T r_k - \gamma_j A_{j'} A d_k\right|\right|^2 \\
\left|\left|A_j^T r_k\right|\right|^2 + \gamma_j^2 \left|\left|A_j^T A d_k\right|\right|^2 - 2 r_k^T A_j \gamma_j A_j^T A d_k = \left|\left|A_{j'}^T r_k\right|\right|^2 + \gamma_j^2 \left|\left|A_{j'}^T A d_k\right|\right|^2 - 2 r_k^T A_{j'} \gamma_j A_{j'}^T A d_k \\
\left(\left|\left|A_j^T A d_k\right|\right|^2 - \left|\left|A_{j'}^T A d_k\right|\right|^2\right) \gamma_j^2 - 2 r_k^T \left(A_j A_j^T - A_{j'} A_{j'}^T\right) A d_k \gamma_j + \left|\left|A_j^T r_k\right|\right|^2 - \left|\left|A_{j'}^T r_k\right|\right|^2 = 0.
\end{gathered}$$

Thus, finding γ_j corresponds to solving the quadratic equation

$$a\gamma_j^2 - b\gamma_j + c = 0$$

with

$$\begin{aligned}
a &= \left|\left|A_j^T A d_k\right|\right|^2 - \left|\left|A_{j'}^T A d_k\right|\right|^2 \\
b &= 2 r_k^T \left(A_j A_j^T - A_{j'} A_{j'}^T\right) A d_k \\
c &= \left|\left|A_j^T r_k\right|\right|^2 - \left|\left|A_{j'}^T r_k\right|\right|^2.
\end{aligned}$$

Among all the $j \in I_k^c$ we choose the one with the smallest γ_j, and add the corresponding index to the active set. The algorithm stops when γ is equal to 1, or when $|I_k| = J$. The Group LARS method is specified entirely in Algorithm 3.

[12]Since by construction all the active entries of the residual correlations decline equally, $\left|\left|A_{j'}^T(r_k - \gamma A d_k)\right|\right|$ has the same value for all $j' \in I_k$.

[13]Here we use the fact that $||v_1 - v_2||^2 = ||v_1||^2 + ||v_2||^2 - 2v_1^T v_2$.

Algorithm 3 Group LARS

input: A, y, J, stopping criterion
  $x_0 = 0$
  $r_0 = y$
  $I_1 = \{\text{argmax}_j(\left|\left|A_j^T r_0\right|\right|)\}$
  $k = 1$
  while not converged **do**
    $d_k = \mathbf{0}$
    $d_k(I_k) = \left(A_{I_k}^T A_{I_k}\right)^{-1} A_{I_k}^T r_k$
    $j' =$ choose any from I_k
    for $j \in I_k^c$ **do**
      $a = \left|\left|A_j^T A d_k\right|\right|^2 - \left|\left|A_{j'}^T A d_k\right|\right|^2$
      $b = 2 r_k^T \left(A_j A_j^T - A_{j'} A_{j'}^T\right) A d_k$
      $c = \left|\left|A_j^T r_k\right|\right|^2 - \left|\left|A_{j'}^T r_k\right|\right|^2$
      $\gamma_j = \text{solve}(a\gamma^2 - b\gamma + c)$
    end for
    $\gamma_{j^*}, i_k = \min_j^+ \{\gamma_j\}$
    $I_{k+1} = I_k \cup \{i_k\}$
    $x_{k+1} = x_k + \gamma_{j^*} d_k$
    $r_{k+1} = y - A x_{k+1}$
    $k = k + 1$
  end while
output: x_j

2.4 Greedy Methods

Greedy methods [98] are an alternative to the recovery algorithms based on convex optimization shown so far. Orthogonal Matching Pursuit (OMP) [30, 80, 95, 99], described in Algorithm 4, is one the most popular methods in this category. OMP is an iterative algorithm that builds an estimate of the support of x by adding one index to this set at a time. The algorithm starts with an empty estimate $\Gamma^{(0)}$. It keeps a residue vector r, initially equal to y, which corresponds to the component of y perpendicular to the column span of A_Γ. At each iteration, OMP computes the correlation between the current residue and the columns of A. The index of the column with the highest correlation is added to the current estimate of the support. Using this new support estimate a new residue is computed. The loop exits when the stopping criterion is met, typically when the norm of the residue is small enough.

Algorithm 4 Orthogonal Matching Pursuit

input: A, y, stopping criterion
  initialize: $r^{(0)} = y$, $\Gamma^{(0)} = \emptyset$, $\ell = 0$
  while not converged **do**
    match: $h = |A^T r|$
    identify: $\Gamma^{(\ell+1)} = \Gamma^{(\ell)} \cup \text{argmax}_j \, |h(j)|$
    update: $x^{\ell+1} = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Gamma^{(\ell+1)}} \|y - Az\|_2$
    $r^{\ell+1} = y - Ax^{\ell+1}$
    $\ell = \ell + 1$
  end while
output: $\widehat{x} = x^\ell = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Gamma^{(\ell)}} \|y - Az\|_2$

Algorithm 5 BOMP

input: A, y, J, stopping criterion
  initialize: $r^{(0)} = y$, $\Gamma^{(0)} = \emptyset$, $\ell = 0$
  while not converged **do**
    match: $h(j) = \left|\left|A_j^T r\right|\right|$, $j = 1, \ldots, J$
    identify: $\Gamma^{(\ell+1)} = \Gamma^{(\ell)} \cup \text{argmax}_j \, h(j)$
    update: $x^{\ell+1} = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Gamma^{(\ell+1)}} \|y - Az\|_2$
    $r^{\ell+1} = y - Ax^{\ell+1}$
    $\ell = \ell + 1$
  end while
output: $\widehat{x} = x^\ell = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Gamma^{(\ell)}} \|y - Az\|_2$

If, in addition to being K-sparse, x has some extra structure, it is possible to design recovery algorithms that outperform OMP. As before, we consider the case where signals are *group sparse* [104], i.e., the vector x can be partitioned in J components[14] $x = [x_1^T \ x_2^T \ \cdots \ x_J^T]^T$ with at most $K_G \ll J$ nonzero components, where $x_j \in \mathbb{R}^{n_J}$, $n_J = n/J$, and $j = 1, \ldots, J$. The matrix A is also partitioned in the corresponding groups as $A = [A_1 \cdots A_J]$, $A_j \in \mathbb{R}^{m \times n_J}$.

An algorithm that allows one to recover group-sparse signals is Block Orthogonal Matching Pursuit (BOMP) [44, 45], described in Algorithm 5. The key difference with respect to OMP is that instead of computing the correlation between the current residue and the columns of A, each entry of the correlation vector h is computed as $h(j) = \left|\left|A_j^T r\right|\right|$.

[14] As in the group LASSO case, we assume that all the components have equal size and that $N_J = N/J$ is an integer.

CHAPTER 3

JOINT DENOISING

"Less is only more where more is no good."

Frank Lloyd Wright

Faced with the task of estimating a signal from noisy observations, there is hope of success only if some type of prior information about the signal of interest is available. In other words, the cornerstone of any signal estimation scheme is the assumption that the signals of interest can be described by a *signal model*. A simple example is the use of filters in the frequency domain—low-pass, high-pass, or band-pass filters. In this case, the signal model consists of assuming that the signals of interest are contained in a predefined range of frequencies [89, Ch. 14]. A more sophisticated approach can be used if one knows the joint probability distribution of the signal and the noise. If that is the case, it is possible to use Bayesian techniques [50, Ch. 4]. Unfortunately, many times such probability distributions are hard to find. Another signal model that has gained popularity is the *sparse signal* model. Under this model one assumes that it is possible to represent the signals of interest in a basis where only a small fraction of the coefficients are significant. For instance, if signals are smooth, then they will admit a sparse representation in the frequency domain; and if they are piecewise smooth, they will admit a sparse representation in the wavelet domain [67, Sec. 2.3.1 and 6.1.3].

Estimators based on the sparse signal model are known as *thresholding estimators* [31–33, 37]. Although thresholding estimators have been improved in many ways—for instance, by adapting the threshold to the data, by developing translation-invariant thresholding estimators, by developing nondiagonal estimators, etc. [67, Secs. 11.2.3 and 11.4]—one aspect of this field that so far has been ignored is denoising a signal ensemble. This is a situation commonly observed when working with sensor arrays or sensor networks. If one observes a

set of signals it is possible to naively denoise the signals independently; in this thesis, though, we exploit the structure that exists among the signals to obtain better results. Several authors [5, 27, 94, 100] have proposed signal models for a set of signals based in the sparsity patterns of the ensemble. Baron et al. [5] called this model the Joint Sparsity Model (JSM). From the three proposed JSM model variants, we assume that our signals satisfy the JSM-2 model, where all the signals share the same support.

Similarly to the independent denoising of a signal, our proposed method, called *joint denoising*, starts by transforming all the signals into a domain where the signals are sparse. Then it processes all the signal coefficients at a given location at once. If all the coefficients at that location are smaller than a threshold, they are all set to zero. If at least one coefficient at that location is larger than the threshold, *all* the coefficients at that location are kept. In the final step the signals are transformed back to the original domain. Since a "large" coefficient is able to "save" all the coefficients at a given location, we call this a vetoing scheme.

The chapter continues with a review of thresholding estimators. It follows with the definition of the joint denoising problem and a description and analysis of the proposed joint estimators. It finalizes with some empirical results.

3.1 Thresholding Estimators

Let $x \in \mathbb{R}^N$ be a discrete signal. This signal can be represented in an orthonormal basis as $x = \Psi\theta$, where Ψ is an $N \times N$ orthonormal matrix[1] representing the basis, and $\theta \in \mathbb{R}^N$ is a vector of coefficients. We consider the case where a sensor observes a noisy version of x given by

$$\xi = x + z,$$

[1]Note that if Ψ is an orthonormal matrix, then $\Psi\Psi^T = \Psi^T\Psi = I$.

where $z \in \mathbb{R}^N$ represents a zero-mean white noise term of variance σ^2, i.e.,

$$\mathbf{E}\left[zz^T\right] = \sigma^2 I_N,$$

where I_N is the $N \times N$ identity matrix.

Let $w = \Psi^T z$ be the noise in the coefficient domain. An important characteristic of zero-mean white noise is that it remains white noise under an orthonormal transformation:

$$\begin{aligned}\mathbf{E}\left[ww^T\right] &= \mathbf{E}\left[\Psi^T zz^T \Psi\right] \\ &= \Psi^T \mathbf{E}\left[zz^T\right] \Psi \\ &= \sigma^2 I_N.\end{aligned}$$

Thus, we can represent the noisy observations as

$$\begin{aligned}y &= \Psi^T \xi \\ &= \Psi^T (x + z) \\ &= \theta + w,\end{aligned}$$

where w is zero-mean white noise of variance σ^2.

A *diagonal estimator* [67] is an estimator that operates on each observed coefficient individually by multiplying the coefficient by a factor. We can write such estimators as

$$\begin{aligned}\widehat{\theta} &= Dy \\ &= \sum_{k=1}^{N} a_k(y(k))y(k),\end{aligned}$$

where $a_k(\cdot)$ is the function that operates in the k^{th} entry of y. If $a_k(y(k)) = a_k$ is a constant independent of the value of $y(k)$, then D is a linear operator. To evaluate the performance

of a given operator, we use the concept of *risk*, defined as

$$\begin{aligned} r(D,\theta) &= \mathbf{E}\left[\left|\left|\theta - \widehat{\theta}\right|\right|^2\right] \\ &= \sum_{k=1}^{N} \mathbf{E}\left[(\theta(k) - a_k(y(k))y(k))^2\right]. \end{aligned}$$

If D is a linear operator, then

$$\begin{aligned} r(D,\theta) &= \sum_{k=1}^{N} \mathbf{E}\left[(\theta(k) - a_k y(k))^2\right] \\ &= \sum_{k=1}^{N} \theta^2(k)(1-a_k)^2 + \sigma^2 a_k^2. \end{aligned}$$

This expression is minimized by

$$a_k = \frac{\theta^2(k)}{\theta^2(k) + \sigma^2}.$$

Note that to compute this we need to know θ, precisely the signal we are trying to estimate. Since to use this estimator we need access to information that is unknown, we call this an *oracle estimator*. The risk of this estimator is given by

$$r_{inf}(\theta) = \sum_{k=1}^{N} \frac{\theta^2(k)\sigma^2}{\theta^2(k) + \sigma^2},$$

and provides a lower bound for other estimators.

It is also useful to consider the case where $a_k \in \{0,1\}$, i.e., a "keep it or kill it" estimator. Such estimator is an orthogonal projector onto a subset of the basis defined by Ψ. The risk for this estimator is

$$\begin{aligned} r(D,\theta) &= \sum_{k=1}^{N} \mathbf{E}\left[(\theta(k) - a_k y(k))^2\right] \\ &= \sum_{k=1}^{N} \begin{cases} \theta^2(k) & a_k = 0, \\ \sigma^2 & a_k = 1. \end{cases} \end{aligned}$$

The risk is minimized by setting

$$a_k = \begin{cases} 1 & \text{if } \theta(k) \geq \sigma, \\ 0 & \text{if } \theta(k) < \sigma. \end{cases}$$

Note that this is still an oracle estimator, since it requires the knowledge of the signal coefficients. Its risk is given by

$$r_{pr}(\theta) = \sum_{k=1}^{N} \min\left(\theta(k)^2, \sigma^2\right),$$

and is related to r_{inf} by[2]

$$\frac{1}{2} r_{pr}(\theta) \leq r_{inf}(\theta) \leq r_{pr}(\theta).$$

Naturally, for an estimator to be of practical use it must operate on the noisy observations. A class of estimators commonly use in practice are the so called *thresholding estimators*. These estimators are based on the hard-thresholding function defined as

$$\rho_{HT}(x) = \begin{cases} x & \text{if } |x| \geq T, \\ 0 & \text{if } |x| < T, \end{cases}$$

and on the soft-thresholding function defined as

$$\rho_{ST}(x) = \begin{cases} x - T & \text{if } x \geq T, \\ x + T & \text{if } x \leq T, \\ 0 & \text{if } |x| < T, \end{cases}$$

where $T > 0$ is a given thresholding level. Note that the soft-thresholding function can also be written as $\rho_{ST}(x) = \max\{|x| - T, 0\} \operatorname{sign}(x)$. Figure 3.1 shows a plot of both functions. The

[2]This follows from the inequality $\frac{1}{2}\min(x, y) \leq \frac{xy}{x+y} \leq \min(x, y)$ for all $x, y \in \mathbb{R}$.

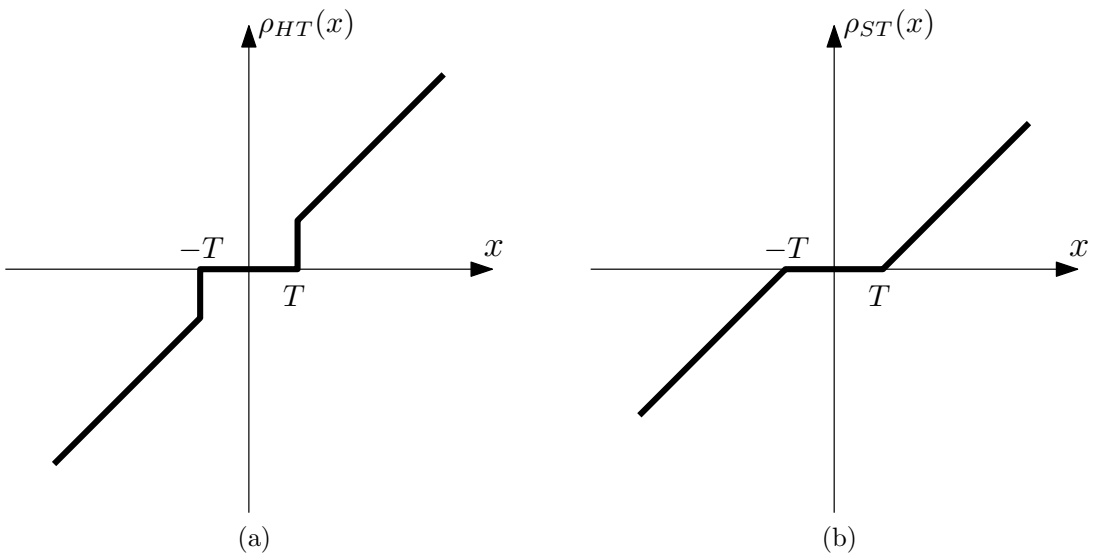


Figure 3.1: Thresholding functions. (a) Hard-thresholding function $\rho_{HT}(x)$ (b) Soft-thresholding function $\rho_{ST}(x)$. Unlike hard-thresholding, soft-thresholding is a continuous function.

main difference between the functions is that the soft-thresholding function is continuous, while the hard-thresholding function is discontinuous.

The hard-thresholding estimator is given by

$$\widehat{\theta}(k) = \rho_{HT}(y(k)), \tag{3.1}$$

and the soft-thresholding estimator is given by

$$\widehat{\theta}(k) = \rho_{ST}(y(k)). \tag{3.2}$$

For reasons that will become clearer in the next section, we call the estimator given by 3.3 the *independent estimator* and define

$$\widehat{\theta^I}(k) = \rho_{HT}(y(k)). \tag{3.3}$$

As the next theorem shows, the theoretical guarantees for both estimators are the same. In

practice however, the soft-thresholding estimator is sometimes preferred because it causes less artifacts.

Theorem 3.1 (Donoho, Johnstone [67, Sec. 11.2.2])
Let $T = \sigma\sqrt{2\log N}$. The risk $r_{th}(\theta) = \sum_k \mathbf{E}\left[\left|\theta(k) - \widehat{\theta}(k)\right|^2\right]$, where $\widehat{\theta}$ is either a hard- or soft thresholding estimator, satisfies for all $N \geq 4$,

$$r_{th}(\theta) \leq (2\log N + 1)\left(\sigma_w^2 + r_{pr}(\theta)\right).$$

3.2 Joint Denoising

In this chapter the problem we wish to solve is the following. Given a signal ensemble, and assuming that the signals in the ensemble possess some inter-signal structure, can we design an estimator with better performance than the independent estimator?

We first extend the notation as follows. Let $x_j \in \mathbb{R}^N$, for $j \in \Lambda = \{1, \ldots, J\}$, be a signal belonging to an ensemble of J signals. Let $\theta_j = \Psi x_j$ be the transform coefficients of x_j in the domain defined by Ψ, where Ψ is an $N \times N$ orthogonal matrix. We observe a noisy version of the signals given by $x_j + z_j$. We assume that z_j is Gaussian i.i.d. zero-mean white noise of variance σ_z^2. The corresponding noisy coefficients are given by

$$y_j = \Psi(x_j + z_j) = \theta_j + w_j. \tag{3.4}$$

As before, since Ψ is orthonormal, it follows that w_j is also Gaussian i.i.d. zero-mean white noise with variance $\sigma_w^2 = \sigma_z^2$. To stress the fact that classical thresholding estimators ignore any inter-signal structures, we call them *independent estimators*.

In this work we assume that the signals satisfy the JSM-2 model [5,40]. Let the support of θ_j be the index set $\Omega_j = \{k|\theta_j(k) \neq 0\}$. A set of signals satisfies the JSM-2 model if $\Omega_j = \Omega_i := \Omega$ for all $i, j \in \Lambda$. In words, this means that under this signal model, the location

of the nonzero coefficients is the same for all the signals. A situation like this one may arise, for instance, when a single signal source is captured by a sensor array. Since each sensor is at a different location, they observe coefficients with different amplitudes and phases, but with the same positions.

3.2.1 The Joint Estimator

We propose to denoise the signal ensemble using the joint estimator given by

$$\widehat{\theta}_j^V(k) = \begin{cases} 0, & |y_j(k)| < T \text{ and } |y_i(k)| < T, \text{ for all } i \in \Lambda \backslash \{j\}, \\ y_j(k), & |y_j(k)| < T \text{ and } |y_i(k)| \geq T, \text{ for some } i \in \Lambda \backslash \{j\}, \\ y_j(k), & |y_j(k)| \geq T, \end{cases}$$

$$= \begin{cases} 0, & |y_i(k)| < T, \text{ for all } i \in \Lambda, \\ y_j(k), & |y_i(k)| \geq T, \text{ for some } i \in \Lambda. \end{cases}$$

Figure 3.2 illustrates how joint denoising works. Observations larger than the threshold are kept. When a measurement is smaller than the threshold, but there is a measurement from another signal larger than the threshold at the same location, that coefficient is saved. When all the coefficients at a given location are smaller than the threshold, these coefficients are set to 0. In other words, a "large coefficient" has the power to "veto" the killing of small coefficients located at the same position.

We wish to quantify the difference between the expected square errors of both estimators (also known as *risk*). To this end, we first analyze the structure of the square error for the independent estimator. We can write

$$\left\| \theta_j - \widehat{\theta}_j^I \right\|^2 = \sum_k \left(\theta_j(k) - \widehat{\theta}_j^I(k) \right)^2. \tag{3.5}$$

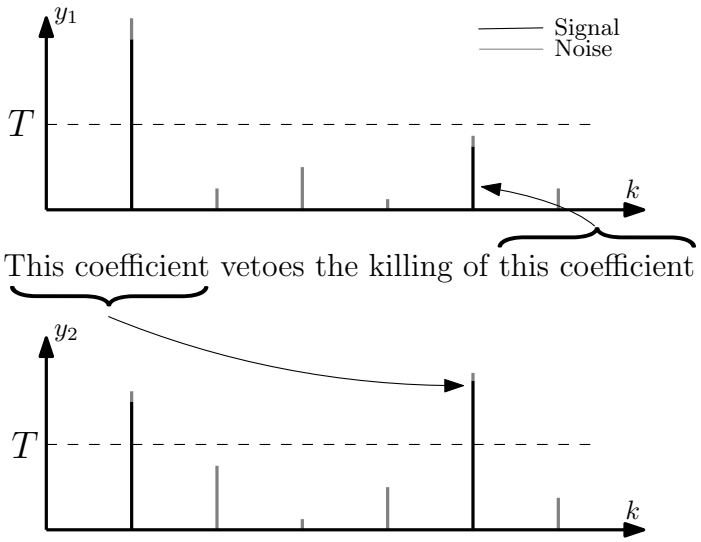


Figure 3.2: Joint denoising. If at a given location all the coefficients are smaller than T, these coefficients are set to 0. On the other hand, if at least one coefficient is larger than T, all the coefficients at this location are kept. In this example, one of the entries of y_2 "vetoes" the "killing" of a y_1 coefficient that is smaller than T.

We can now write each element of the summation in equation (3.5) as

$$\left(\theta_j(k) - \widehat{\theta}_j^I(k)\right)^2 = \begin{cases} (\theta_j(k) - y_j(k))^2 = w_j^2(k), & k \in \Omega \text{ and } |y_j(k)| \geq T, \quad (3.6a) \\ (0 - y_j(k))^2 = w_j^2(k), & k \notin \Omega \text{ and } |y_j(k)| \geq T, \quad (3.6b) \\ (\theta_j(k) - 0)^2 = \theta_j^2(k), & k \in \Omega \text{ and } |y_j(k)| < T, \quad (3.6c) \\ (0 - 0)^2 = 0, & k \notin \Omega \text{ and } |y_j(k)| < T. \quad (3.6d) \end{cases}$$

Equation (3.6) sheds some light on the nature of the estimator error, and on how the two estimators are different. Case (3.6d) occurs almost as many times as there are coefficients with zero value, i.e., the more sparse the signal is, the more often this situation happens; since in this case the contribution to the error is zero, it follows that, all other things being equal, the more sparse the signals are, the smaller the error. Finally, we notice that cases (3.6b) and (3.6c) are where the two methods to be analyzed differ.

We are now in a position to quantify the difference between the independent and the joint denoising technique. At a given location, the outcome of the two methods is different

whenever one of the following conditions happens:

Condition 1

(i) a coefficient $\theta_j(k)$ is non-zero,

(ii) an observation $y_j(k)$ is smaller than the threshold T, and

(iii) at least one observation $y_i(k)$ for $i \neq j$ is greater than T;

Condition 2

(i) a coefficient $\theta_j(k)$ is zero,

(ii) an observation $y_j(k)$ is smaller than the threshold T, and

(iii) at least one observation $y_i(k)$ for $i \neq j$ is greater than T.

Figure 3.3 illustrates these two conditions. Notice that since both signals share the same support (due to the JSM-2 assumption), when Condition 2 happens, all the measurements at that position are only noise. We can now conclude that occurrences of Condition 1 result in an improvement of Joint Denoising over Independent Denoising, while occurrences of Condition 2 result in a loss of improvement.

In order to quantify the improvement of one method over the other for a given signal, we define the quantity

$$I_j = \left\|\theta_j - \widehat{\theta}_j^I\right\|^2 - \left\|\theta_j - \widehat{\theta}_j^V\right\|^2 \quad j \in \Lambda. \tag{3.7}$$

Before presenting the theorem that quantifies I_j, we need the following lemmas and definitions.

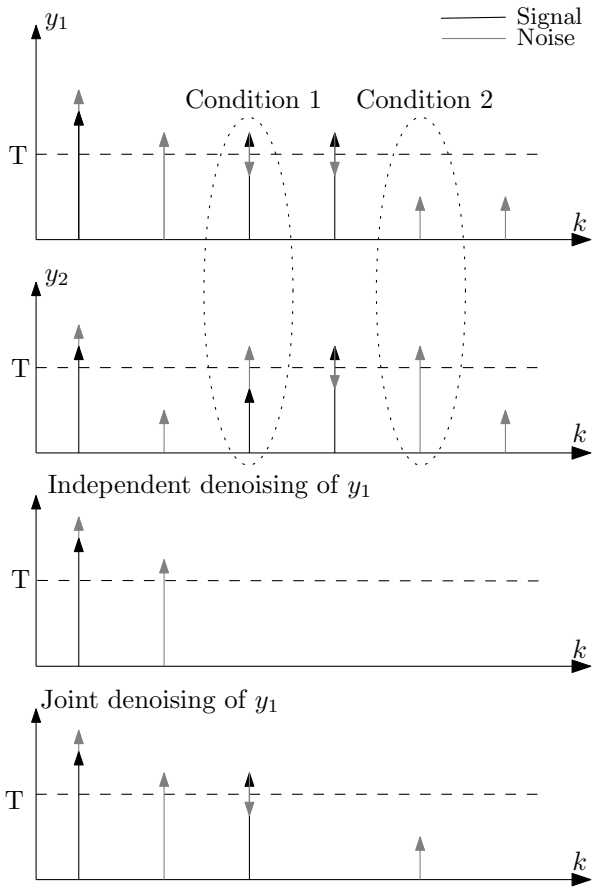


Figure 3.3: Conditions under which the two methods produce a different result. Under Condition 1, an observation equal to a nonzero coefficient plus noise saves an observation smaller than T. Under Condition 2, an observation that is only noise, since at that location all the coefficients are zero, saves an observation, that is also only noise, that is smaller than T.

Lemma 3.2 (Gaussian distribution [53, Sec. 5.2])

Let $X \sim \mathcal{N}(0, \sigma)$ be a random variable. Then

$$P\{X \leq x\} = \Phi\left(\frac{x}{\sigma}\right),$$

with $\Phi(x) = \int_{\infty}^{x} \phi(t)dt$, $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$.

Definition 3.3 (Covariance and correlation [22, Sec. 4.5])

Let X and Y be two random variables with means μ_X and μ_Y and variances σ_X^2 and σ_Y^2,

respectively. The *covariance* of X and Y is defined by

$$\mathrm{Cov}(X, Y) = \mathbf{E}\left[(X - \mu_X)(Y - \mu_Y)\right].$$

The correlation of X and Y is defined by

$$\rho_{XY} = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y}.$$

Lemma 3.4 (See [22, Thm. 4.5.3])

For any random variables X and Y,

$$\mathrm{Cov}(X, Y) = \mathbf{E}\left[XY\right] - \mathbf{E}\left[X\right]\mathbf{E}\left[Y\right].$$

Lemma 3.5 (Conditional expectation of bivariate Gaussians [9][3])

Let $X \sim \mathcal{N}(\mu_x, \sigma_x^2)$ and $Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$. Then

$$\mathbf{E}\left[X \;\middle|\; Y\right] = \mu_x + \rho_{XY}\frac{\sigma_x}{\sigma_y}(Y - \mu_y).$$

Lemma 3.6 (Sum of independent Gaussians [22, Thm. 4.2.14])

Let $X \sim \mathcal{N}(\mu, \sigma^2)$ and $Y \sim \mathcal{N}(\gamma, \tau)$ be independent Gaussian random variables. Then the random variable $Z = X + Y$ has a $\mathcal{N}(\mu + \gamma, \sigma^2 + \tau^2)$ distribution.

Lemma 3.7 (Law of total expectation [22, Thm. 4.4.3])

If X and Y are two random variables, then

$$\mathbf{E}\left[X\right] = \mathbf{E}\left[\mathbf{E}\left[X \;\middle|\; Y\right]\right],$$

provided that the expectations exist.

[3]This result is available only in the first edition of this book. An online version of this result is available at http://www.athenasc.com/Bivariate-Normal.pdf.

Lemma 3.8 (Law of iterated expectations [54, Sec. 3.6])

If X, Y and Z are random variables, then

$$\mathbf{E}\left[X \mid Y\right] = \mathbf{E}\left[\mathbf{E}\left[X \mid Y, Z\right] \mid Y\right],$$

provided that the expectations exist.

Lemma 3.9 (Random sum)

Let X_i be i.i.d. random variables and let N be a random variable independent of X_i. Then

$$\mathbf{E}\left[\sum_{i=1}^{N} X_i\right] = \mathbf{E}\left[N\right]\mathbf{E}\left[X_1\right],$$

provided that the expectations exist.

Proof. For a fixed N, $\mathbf{E}\left[\sum_{i=1}^{N} X_i\right] = \sum_{i=1}^{N} \mathbf{E}\left[X_i\right] = N\mathbf{E}\left[X_1\right]$. Using the law of total expectation (Lemma 3.7),

$$\begin{aligned}
\mathbf{E}\left[\sum_{i=1}^{N} X_i\right] &= \mathbf{E}\left[\mathbf{E}\left[\sum_{i=1}^{N} X_i \mid N\right]\right] \\
&= \mathbf{E}\left[N\sum_{i=1}^{N} X_i\right] \\
&= \mathbf{E}\left[N\right]\mathbf{E}\left[X_1\right].
\end{aligned}$$

□

Lemma 3.10

Let $X \sim \mathcal{N}(0, \sigma_x^2)$ and $Y \sim \mathcal{N}(0, \sigma_y^2)$ be two random variables, and let $U = X + Y$ and $V = X - Y$. Then

$$\mathbf{E}\left[V \mid U = u\right] = \rho u, \quad \rho = \frac{\sigma_x^2 - \sigma_y^2}{\sigma_x^2 + \sigma_y^2}.$$

Proof. U and V are both Gaussian random variables with zero mean and variance $\sigma_x^2 + \sigma_y^2$

(by Lemma 3.6). Since U and V have zero mean and by Lemma 3.4, the covariance of U and V is

$$\begin{aligned}\mathrm{Cov}(U,V) &= \mathbf{E}\left[UV\right] \\ &= \mathbf{E}\left[(X+Y)(X-Y)\right] \\ &= \mathbf{E}\left[X^2\right] - \mathbf{E}\left[Y^2\right] \\ &= \sigma_x^2 - \sigma_y^2,\end{aligned}$$

and the correlation is

$$\begin{aligned}\rho &= \frac{\mathrm{Cov}(U,V)}{\sigma_u \sigma_v} \\ &= \frac{\sigma_x^2 - \sigma_y^2}{\sigma_x^2 + \sigma_y^2}.\end{aligned}$$

By Lemma 3.5

$$\begin{aligned}\mathbf{E}\left[V \;\middle|\; U = u\right] &= \mathbf{E}\left[V\right] + \rho\frac{\sigma_v}{\sigma_u}(u - \mathbf{E}\left[U\right]) \\ &= \rho u.\end{aligned}$$

□

Lemma 3.11 (Variance of a truncated zero mean Gaussian [56], [75, Sec. 4.3])
Let $X \sim \mathcal{N}(0, \sigma^2)$. Then

$$\mathbf{Var}\left[X \;\middle|\; -T < X < T\right] = \sigma^2\left(1 - \frac{2T\phi\left(\frac{T}{\sigma}\right)}{\sigma\left(\Phi\left(\frac{T}{\sigma}\right) - \Phi\left(\frac{-T}{\sigma}\right)\right)}\right).$$

Lemma 3.12 (Variance of a complementary truncated zero mean Gaussian)
Let $X \sim \mathcal{N}(0, \sigma^2)$. Then

$$\mathbf{Var}\left[X \;\middle|\; |X| > T\right] = \sigma^2\left(1 + \frac{2T\phi(\frac{T}{\sigma})}{\sigma\left(1 + \Phi(-\frac{T}{\sigma}) - \Phi(\frac{T}{\sigma})\right)}\right).$$

Proof. The *pdf* of X conditioned by $|X| > T$ is

$$f_X(x \mid |X| > T) = \frac{1}{c} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \mathbb{I}_{|x|>T},$$

where c is the normalization constant

$$c = 1 + \Phi\left(-\frac{T}{\sigma}\right) - \Phi\left(\frac{T}{\sigma}\right).$$

By the symmetry of the *pdf* and of the conditioning interval, $\mathbf{E}\left[X \mid |X| > T\right] = 0$. Thus, to compute the conditional variance it suffices to compute

$$\mathbf{E}\left[X^2 \mid |X| > T\right] = \int_{-\infty}^{\infty} f_X(x \mid |X| > T)\, dx.$$

Since[4]

$$\frac{1}{\sqrt{2\pi}\sigma} \int x^2 e^{-\frac{x^2}{2\sigma^2}}\, dx = \frac{\sigma}{2}\left(\sigma \operatorname{erf}\left(\frac{x}{\sqrt{2}\sigma}\right) - \sqrt{\frac{2}{\pi}} x e^{-\frac{x^2}{2\sigma^2}}\right) + \text{constant},$$

where $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$ is the *error function*, and using the facts that $\lim_{x\to-\infty} \operatorname{erf}(x) = -1$, $\lim_{x\to\infty} \operatorname{erf}(x) = 1$, and $\operatorname{erf}(x) = 2\Phi(\sqrt{2}x) - 1$ we can write

$$\begin{aligned} c\mathbf{Var}\left[X^2 \mid |X| > T\right] &= \frac{1}{\sqrt{2\pi}\sigma} \int_{|x|>T} x^2 e^{-\frac{x^2}{2\sigma^2}}\, dx \\ &= \frac{\sigma}{2}\left[\left(\sigma \operatorname{erf}\left(\frac{x}{\sqrt{2}\sigma}\right) - \sqrt{\frac{2}{\pi}} x e^{-\frac{x^2}{2\sigma^2}}\right)\Bigg|_{-\infty}^{-T} + \left(\sigma \operatorname{erf}\left(\frac{x}{\sqrt{2}\sigma}\right) - \sqrt{\frac{2}{\pi}} x e^{-\frac{x^2}{2\sigma^2}}\right)\Bigg|_{T}^{\infty}\right] \\ &= \sigma^2\left(1 + \Phi\left(-\frac{T}{\sigma}\right) - \Phi\left(\frac{T}{\sigma}\right) + \frac{2T}{\sigma}\phi\left(\frac{T}{\sigma}\right)\right). \end{aligned}$$

□

Theorem 3.13

Consider a signal ensemble satisfying the JSM-2 model, with the nonzero coefficients i.i.d.

[4]Computed using Wolfram Alpha: http://bit.ly/ZFz3nF. The expression can be verified by taking the derivative. See also http://bit.ly/WaJpLg.

drawn at random from a Gaussian distribution with variance σ_θ^2, i.e., $\theta_j(k) \sim \mathcal{N}(0, \sigma_\theta^2)$ for $k \in \Omega$. Denote the sparsity of each signal by $S = |\Omega|$. Then

$$\mathbf{E}\left[I_j\right] = E_{C1} - E_{C2}$$

with

$$E_{C1} = Sp_1(1 - p_1^{J-1})\rho\sigma_y^2 \left(1 - \frac{2T\phi\left(\frac{T}{\sigma_y}\right)}{\sigma_y p_1}\right),$$

$$E_{C2} = (N - S)p_2(1 - p_2^{J-1})\sigma_w^2 \left(1 - \frac{2T\phi\left(\frac{T}{\sigma_w}\right)}{\sigma_w p_2}\right),$$

where

$$p_1 = \Phi\left(\frac{T}{\sigma_y}\right) - \Phi\left(\frac{-T}{\sigma_y}\right),$$

$$p_2 = \Phi\left(\frac{T}{\sigma_w}\right) - \Phi\left(\frac{-T}{\sigma_w}\right),$$

$$\sigma_y^2 = \sigma_\theta^2 + \sigma_w^2, \qquad \rho = \frac{\sigma_\theta^2 - \sigma_w^2}{\sigma_\theta^2 + \sigma_w^2},$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \quad \text{and} \quad \Phi(x) = \int_\infty^x \phi(t)dt.$$

Proof. Equation (3.7) can be written as

$$I_j = \sum_k \left(\theta_j(k) - \widehat{\theta}_j^I(k)\right)^2 - \sum_k \left(\theta_j(k) - \widehat{\theta}_j^V(k)\right)^2. \tag{3.8}$$

For a given k, the terms in the two summations will cancel out whenever the two

methods produce the same result. As discussed previously, joint denoising produces a result different than independent denoising when Condition 1 or Condition 2 happens. We can write the expected value of I_j as $\mathbf{E}[I_j] = E_{C1} - E_{C2}$, where E_{C1} quantifies the improvement due to the occurrence of Condition 1, and E_{C2} quantifies the decline of the improvement due to the occurrence of Condition 2. The proof follows by computing each term.

(i) Condition 1

When this condition happens, independent denoising contributes with an error term equal to $\theta_j^2(k)$, $k \in \Omega$ (as in equation (3.6c)), while the joint denoising contributes with an error term equal to $w_j^2(k)$. Thus, E_{C1} is given by

$$E_{C1} = \mathbf{E}\left[\sum_{k\in\Omega_{C1}} \left(\theta_j^2(k) - w_j^2(k)\right)\right], \tag{3.9}$$

where Ω_{C1} is the index set of the coefficients that satisfy Condition 1.

Since the non-zero coefficients are i.i.d. drawn from a Gaussian distribution, and from equation (3.4), it follows that

$$y_j(k) \sim N(0, \sigma_y^2) \quad k \in \Omega,$$

with $\sigma_y^2 = \sigma_\theta^2 + \sigma_w^2$. The probability that the absolute value of a given measurement is smaller than T is

$$p_1 = P\left\{|y_j(k)| < T \;\middle|\; k \in \Omega\right\} = \Phi\left(\frac{T}{\sigma_y}\right) - \Phi\left(\frac{-T}{\sigma_y}\right),$$

with $\Phi(x) = \int_\infty^x \phi(t)dt$ and $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$. For a given observation $y_j(k)$ with $k \in \Omega$

the probability that Condition 1 happens is

$$\begin{aligned} p_{C1} &= P\left\{|y_j(k)| < T \quad \text{and any} \quad |y_i(k)| > T \;\middle|\; k \in \Omega\right\} \qquad i,j \in \Lambda, i \neq j \\ &= p_1 P\left\{\text{any} \quad |y_i(k)| > T \;\middle|\; k \in \Omega\right\} \\ &= p_1\left(1 - P\left\{\text{all} \quad |y_i(k)| < T \;\middle|\; k \in \Omega\right\}\right) \\ &= p_1\left(1 - p_1^{J-1}\right). \end{aligned} \tag{3.10}$$

Since we are assuming a sparsity level S, the total number of times that Condition 1 occurs (the cardinality of Ω_{C1}) follows a binomial distribution (we have S independent trials, each with probability p_{C1}):

$$|\Omega_{C1}| \sim B(S, p_{C1}),$$

where $B(n,p)$ denotes the binomial distribution with parameters n and p, and

$$\mathbf{E}\left[|\Omega_{C1}|\right] = S p_{C1} = S p_1\left(1 - p_1^{J-1}\right). \tag{3.11}$$

Using Lemma 3.9 and Eq. (3.11) we can write Eq. (3.9) as:

$$E_{C1} = \mathbf{E}\left[|\Omega_{C1}|\right] \mathbf{E}\left[\theta_j^2(k) - w_j^2(k) \;\middle|\; |\theta_j(k) + w_j(k)| < T\right]. \tag{3.12}$$

Using the change of variables

$$\begin{aligned} u &= \theta_j(k) + w_j(k) \\ v &= \theta_j(k) - w_j(k), \end{aligned}$$

the conditional expectation of Eq. (3.12) becomes

$$\mathbf{E}\left[uv \;\middle|\; |u| < T\right].$$

Using the law of iterated expectations (see Lemma 3.8), we can write

$$\begin{aligned}\mathbf{E}\left[uv \;\middle|\; |u| < T\right] &= \mathbf{E}\left[\mathbf{E}\left[uv \;\middle|\; |u| < T, u\right] \;\middle|\; |u| < T\right] \\ &= \mathbf{E}\left[u\mathbf{E}\left[v \;\middle|\; u\right] \;\middle|\; |u| < T\right].\end{aligned} \tag{3.13}$$

By Lemma 3.10,

$$\mathbf{E}\left[v \;\middle|\; u\right] = \rho u,$$

with

$$\rho = \frac{\sigma_\theta^2 - \sigma_w^2}{\sigma_\theta^2 + \sigma_w^2}.$$

We can now write Eq. (3.13) as

$$\begin{aligned}\mathbf{E}\left[uv \;\middle|\; |u| < T\right] &= \mathbf{E}\left[\rho u^2 \;\middle|\; |u| < T\right] \\ &= \rho \int_{-\infty}^{\infty} u^2 f_u\left(u \;\middle|\; |u| < T\right) du.\end{aligned} \tag{3.14}$$

This integral corresponds to the variance of a zero mean truncated Gaussian. By Lemma 3.11

$$\mathbf{E}\left[uv \;\middle|\; |u| < T\right] = \rho\sigma_u^2 \left(1 - \frac{2T\phi\left(\frac{T}{\sigma_u}\right)}{\sigma_u\left(\Phi\left(\frac{T}{\sigma_u}\right) - \Phi\left(\frac{-T}{\sigma_u}\right)\right)}\right). \tag{3.15}$$

By Lemma 3.6, $\sigma_u^2 = \sigma_\theta^2 + \sigma_w^2 = \sigma_y^2$. Since $p_1 = \Phi\left(\frac{T}{\sigma_y}\right) - \Phi\left(\frac{-T}{\sigma_y}\right)$, we get

$$\begin{aligned}E_{C1} &= \mathbf{E}\left[|\Omega_{C1}|\right]\mathbf{E}\left[\theta_j^2(k) - w_j^2(k) \;\middle|\; |\theta_j(k) + w_j(k)| < T\right] \\ &= Sp_1(1 - p_1^{J-1})\rho\sigma_y^2\left(1 - \frac{2T\phi\left(\frac{T}{\sigma_y}\right)}{\sigma_y p_1}\right).\end{aligned} \tag{3.16}$$

(ii) Condition 2

The derivation for E_{C2} follows the same steps as E_{C1}. Since under this condition $\theta_j(k)$

is 0, we have

$$E_{C2} = \mathbf{E}\left[\sum_{k\in\Omega_{C2}} w_j^2(k)\right]. \tag{3.17}$$

where Ω_{C2} denotes the index set of the observations that satisfy Condition 2.

Since $y_j(k) = w_j(k)$ when $k \notin \Omega$ and since $w_j(k) \sim N(0, \sigma_w^2)$, the probability that $y_j(k)$ is smaller than T is

$$p_2 = P(|w_j(k)| < T) = \Phi\left(\frac{T}{\sigma_w}\right) - \Phi\left(\frac{-T}{\sigma_w}\right).$$

and for $k \notin \Omega$ the probability that Condition 2 happens is

$$p_{C2} = p_2\left(1 - p_2^{J-1}\right),$$

The expected value of the cardinality of Ω_{C2} is

$$\mathbf{E}\left[|\Omega_{C2}|\right] = (N-S)p_{C2} = (N-S)p_2\left(1 - p_2^{J-1}\right),$$

and we can write

$$\begin{aligned} E_{C2} &= \mathbf{E}\left[|\Omega_{C2}|\right]\mathbf{E}\left[w_j^2(k) \;\middle|\; |w_j(k)| < T\right] \\ &= (N-S)p_2(1-p_2^{J-1})\sigma_w^2\left(1 - \frac{2T\phi\left(\frac{T}{\sigma_w}\right)}{\sigma_w p_2}\right). \end{aligned}$$

□

Note that although this theorem is about the signal dependent quantity $\mathbf{E}\left[I_j\right]$, the expression we get does not depend on j. It is easier to derive the theorem by focusing in the improvement for a single signal, but the improvement is the same for all values of $j \in \Lambda$.

The proposed veto estimator is able to produce significant improvements over the In-

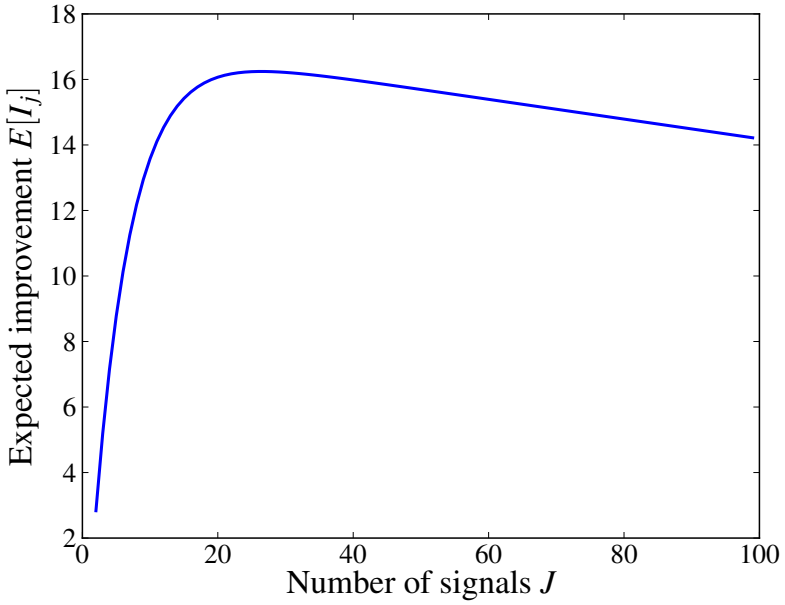


Figure 3.4: Expected improvement of Joint Denoising over Independent Denoising for signal j for different numbers of signals J. We fix the signal length to $N = 1024$, the sparsity level to $S = 50$, the standard deviation of the nonzero coefficients to $\sigma_\theta = 1$, and the standard deviation of the noise to $\sigma_w = 0.4$. The threshold T is set to $T = \sigma_w\sqrt{2\log N} = 1.49$. We observe how initially the improvement increases with the number of signals J, but then it starts decreasing.

dependent Denoising method as the number of signals increases. However, its asymptotic behavior is not ideal. To see why, we show its behavior by fixing the signal length to $N = 1024$, the sparsity level to $S = 50$, the standard deviation of the nonzero coefficients to $\sigma_\theta = 1$, and the standard deviation of the noise to $\sigma_w = 0.4$. The threshold T is set to $T = \sigma_w\sqrt{2\log N} = 1.49$. We plot the expected improvement of Joint Denoising over Independent Denoising for signal j for different number of signals J as given by Theorem 3.13 (see Fig. 3.4). Although initially the expected improvement increases significantly as the number of signals J increases, we observe that it peaks and then it starts decreasing slowly.

This less than optimal asymptotic behavior happens because the current implementation of the veto mechanism is agnostic with respect to the number of signals J, and as J increases Condition 2 occurs more often and the performance of Joint Denoising deteriorates.

3.2.2 A Better Joint Estimator

To improve the asymptotic behavior of the veto scheme we propose the following estimator based on a voting scheme. Let $V(k)$ be the number of observations $y_j(k)$, $j \in \Lambda$, with magnitude larger than T:

$$V(k) = \sum_{j \in \Lambda} \mathbb{I}_{|y_j(k)|>T},$$

where $\mathbb{I}_A$ denotes the indicator function of the set A.

The *voting estimator* is given by[5]

$$\widehat{\theta}_j^R(k) = \begin{cases} y_j(k), & V(k) \geq N_J, \\ 0, & V(k) < N_J, \end{cases}$$

for a given $1 \leq N_J \leq J-1$, $N_J \in \mathbb{N}$. Recall that the veto estimator will "save" an observation if any other measurement is larger than T at the same location. The voting estimator, on the other hand, requires at least N_J observations larger than T to "save" a coefficient. Note that $N_J = 1$ corresponds to the veto joint estimator.

As before, it is useful to identify the conditions under which the vote and the independent estimator differ. Table 3.1 shows the four possible combinations of the outputs of both estimators. There are two cases, rows (ii) and (iii) of the table, where the estimates are different. Since for each of these cases a coefficient $\theta_j(k)$ can be either zero or nonzero, there are four conditions that need to be analyzed.

Condition A

(i) a coefficient $\theta_j(k)$ is non-zero,

(ii) an observation $y_j(k)$ is smaller than the threshold T, and

[5]Since we already use the superscript V to identify the veto estimator, we use the superscript R, as in the first letter of *referendum*, to identify the voting estimator.

Table 3.1: Outcomes of the independent and the vote denoising estimators. The output of the independent estimator ($\widehat{\theta}_j^I(k)$) and the vote estimator ($\widehat{\theta}_j^R(k)$) can be combined in four possible ways. Under two of these combinations—rows (ii) and (iii)—the outputs are different.

	$\lvert y_j(k)\rvert \geq T$	$V(k) \geq N_J$	$\widehat{\theta}_j^I(k)$	$\widehat{\theta}_j^R(k)$
(i)	F	F	0	0
(ii)	F	T	0	$y_j(k)$
(iii)	T	F	$y_j(k)$	0
(iv)	T	T	$y_j(k)$	$y_j(k)$

(iii) at least N_J observations $y_j(k)$ are greater than T, i.e., $V(k) \geq N_J$;

Condition B

(i) a coefficient $\theta_j(k)$ is zero,

(ii) an observation $y_j(k)$ is smaller than the threshold T, and

(iii) at least N_J observations $y_i(k)$ are greater than T, i.e., $V(k) \geq N_J$;

Condition C

(i) a coefficient $\theta_j(k)$ is non-zero,

(ii) an observation $y_j(k)$ is larger than the threshold T, and

(iii) the number of observations $y_j(k)$ greater than T is smaller than N_J, i.e., $V(k) < N_J$;

Condition D

(i) a coefficient $\theta_j(k)$ is zero,

(ii) an observation $y_j(k)$ is larger than the threshold T, and

(iii) the number of observations $y_j(k)$ greater than T is smaller than N_J, i.e., $V(k) < N_J$.

The following theorem quantifies the behavior of the voting estimator.

Theorem 3.14

Consider a signal ensemble satisfying the JSM-2 model, with the nonzero coefficients i.i.d. drawn at random from a Gaussian distribution with variance σ_θ^2, i.e., $\theta_j(k) \sim \mathcal{N}(0, \sigma_\theta^2)$ for $k \in \Omega$. Denote the sparsity of each signal by $S = |\Omega|$. Define the improvement of the voting estimator over the independen estimators as $I_j^R = \left|\left|\theta_j - \widehat{\theta}_j^I\right|\right|^2 - \left|\left|\theta_j - \widehat{\theta}_J^R\right|\right|^2$, $j \in \Lambda$. Then

$$\mathbf{E}\left[I_j^R\right] = E_{CA} - E_{CB} - E_{CC} + E_{CD} \tag{3.18}$$

with

$$E_{CA} = Sp_1 \left(1 - F(N_J - 1; J - 1, 1 - p_1)\right) \rho\sigma_y^2 \left(1 - \frac{2T\phi\left(\frac{T}{\sigma_y}\right)}{\sigma_y p_1}\right),$$

$$E_{CB} = (N - S)p_2 \left(1 - F(N_J - 1; J - 1, 1 - p_2)\right) \sigma_w^2 \left(1 - \frac{2T\phi\left(\frac{T}{\sigma_w}\right)}{\sigma_w p_2}\right),$$

$$E_{CC} = Sp_1 \left(1 - F(N_J - 2; J - 1, 1 - p_1)\right) \rho\sigma_y^2 \left(1 + \frac{2T\phi\left(\frac{T}{\sigma_y}\right)}{\sigma_y(1 - p_1)}\right),$$

$$E_{CD} = (N - S)p_2 \left(1 - F(N_J - 2; J - 1, 1 - p_2)\right) \sigma_w^2 \left(1 + \frac{2T\phi\left(\frac{T}{\sigma_w}\right)}{\sigma_w(1 - p_2)}\right),$$

where

$$F(x; n, p) = \sum_{i=0}^{\lfloor x \rfloor} \binom{n}{i} p^i (1 - p)^{n-i}$$

is the *cumulative distribution function* of the binomial distribution, and p_1, p_2, σ_y^2, ρ, $\phi(x)$, and $\Phi(x)$ are as defined in Theorem 3.13.

Proof. As in the proof of Theorem 3.13, the proof follows by quantifying the contribution of each condition to the improvement.

(i) Condition A

For $k \in \Omega$, the probability that condition A happens is[6]

$$\begin{aligned} p_{CA} &= P\left\{|y_j(k)| < T \text{ and } V(k) \geq N_J \;\middle|\; k \in \Omega\right\} \\ &= P\left\{|y_j(k)| < T \;\middle|\; k \in \Omega\right\} P\left\{V(k) \geq N_J \;\middle|\; |y_j(k)| < T, k \in \Omega\right\} \\ &= p_1 P\left\{V_j(k) \geq N_J \;\middle|\; k \in \Omega\right\} \\ &= p_1\left(1 - P\left\{V_j(k) \leq N_J - 1 \;\middle|\; k \in \Omega\right\}\right) \\ &= p_1\left(1 - F(N_J - 1; J - 1, 1 - p_1)\right), \end{aligned} \tag{3.19}$$

where $V_j(k)$ is the number of observations $y_i(k)$, $i \neq j$ with magnitude larger than T:

$$V_j(k) = \sum_{i \in \Lambda, i \neq j} \mathbb{I}_{|y_i(k)| > T}.$$

By following the same steps as in the proof of Theorem 3.13(i) we get the expression for E_{CA}.

(ii) Condition B

For $k \notin \Omega$, the probability that condition B happens is

$$\begin{aligned} p_{CB} &= P\left\{|y_j(k)| < T \text{ and } V(k) \geq N_J \;\middle|\; k \in \Omega^c\right\} \\ &= P\left\{|y_j(k)| < T \;\middle|\; k \in \Omega^c\right\} P\left\{V(k) \geq N_J \;\middle|\; |y_j(k)| < T, k \in \Omega^c\right\} \\ &= p_2 P\left\{V_j(k) \geq N_J \;\middle|\; k \in \Omega^c\right\} \\ &= p_2\left(1 - P\left\{V_j(k) \leq N_J - 1 \;\middle|\; k \in \Omega^c\right\}\right) \\ &= p_2\left(1 - F(N_J - 1; J - 1, 1 - p_2)\right). \end{aligned} \tag{3.20}$$

By following the same steps as in the proof of Theorem 3.13(ii) we get the expression for E_{CB}.

[6]Note that the events $|y_j(k)| < T$ and $V(k) \geq N_J$ are not independent, thus to compute this probability we use the fact that $P\{A \cap B\} = P\left\{A \;\middle|\; B\right\} P\{B\}$.

(iii) Condition C

For $k \in \Omega$, the probability that condition C happens is

$$\begin{aligned} p_{CC} &= P\left\{|y_j(k)| \geq T \text{ and } V(k) \leq N_J - 1 \;\middle|\; k \in \Omega\right\} \\ &= P\left\{|y_j(k)| \geq T \;\middle|\; k \in \Omega\right\} P\left\{V(k) \leq N_J - 1 \;\middle|\; |y_j(k)| \geq T, k \in \Omega\right\} \\ &= (1-p_1) P\left\{V_j(k) \leq N_J - 2 \;\middle|\; k \in \Omega\right\} \\ &= (1-p_1)\left(1 - F(N_J - 2; J-1, 1-p_1)\right). \end{aligned} \tag{3.21}$$

We can write

$$\begin{aligned} E_{CC} &= \mathbf{E}\left[\sum_{k \in \Omega_{CC}} \left(w_j(k)^2 - \theta_j(k)^2\right)\right] \\ &= \mathbf{E}\left[|\Omega_{CC}|\right] \mathbf{E}\left[\left(w_j(k)^2 - \theta_j(k)^2\right) \;\middle|\; |w_j(k) + \theta_j(k)| \geq T\right], \end{aligned} \tag{3.22}$$

Where Ω_{CC} is the index set of the coefficients that satisfy Condition C. By the same argument used in the proof of Theorem 3.13

$$\mathbf{E}\left[|\Omega_{CC}|\right] = S p_{CC}.$$

Using the change of variables $u = w_j(k) + \theta_j(k)$ and $v = w_j(k) - \theta_j(k)$ and Lemmas 3.7 and 3.10, and by the same arguments followed to derive Eq. (3.13), the second expectation in Eq. (3.22) becomes

$$\begin{aligned} \mathbf{E}\left[uv \;\middle|\; |u| \geq T\right] &= \mathbf{E}\left[u\mathbf{E}\left[v \;\middle|\; u\right] \;\middle|\; |u| \geq T\right] \\ &= \mathbf{E}\left[\rho_C u^2 \;\middle|\; |u| \geq T\right] \\ &= \rho_C \int_{-\infty}^{\infty} u^2 f_u\left(u \;\middle|\; |u| < T\right) du, \end{aligned}$$

with $\rho_C = \frac{\sigma_w^2 - \sigma_\theta^2}{\sigma_w^2 + \sigma_\theta^2} = -\rho$. Since this is the variance of a complementary-truncated zero

mean Gaussian, using Lemma 3.12 we can write

$$\rho_C \int_{-\infty}^{\infty} u^2 f_u\left(u \;\middle|\; |u| < T\right) du = \rho_C \sigma_u^2 \left(1 + \frac{2T\phi(\frac{T}{\sigma_u})}{\sigma_u\left(1 + \Phi(-\frac{T}{\sigma_u}) - \Phi(\frac{T}{\sigma_u})\right)}\right)$$
$$= \rho_C \sigma_y^2 \left(1 + \frac{2T\phi(\frac{T}{\sigma_y})}{\sigma_y\left(1 - p_1\right)}\right).$$

(iv) Condition D

For $k \notin \Omega$, the probability that condition D happens is

$$\begin{aligned} p_{CC} &= P\left\{|y_j(k)| \geq T \text{ and } V(k) \leq N_J - 1 \;\middle|\; k \in \Omega^c\right\} \\ &= P\left\{|y_j(k)| \geq T \;\middle|\; k \in \Omega^c\right\} P\left\{V(k) \leq N_J - 1 \;\middle|\; |y_j(k)| \geq T, k \in \Omega^c\right\} \\ &= (1 - p_2) P\left\{V_j(k) \leq N_J - 2 \;\middle|\; k \in \Omega^c\right\} \\ &= (1 - p_2)\left(1 - F(N_J - 2; J - 1, 1 - p_2)\right). \end{aligned} \tag{3.23}$$

By following the same steps as in the proof of Theorem 3.13(ii) and using Lemma 3.12 we get the expression for E_{CD}.

□

Figure 3.5 shows the expected improvement for the voting estimator for different values of the numbers of signals J and different values of the number of votes N_J. The signal length is set to $N = 1024$, the sparsity level set to $S = 50$, the standard deviation of the nonzero coefficients set to $\sigma_x = 1$, and the standard deviation of the noise set to $\sigma_w = 0.4$. For each value of J there is a value of N_J that maximizes the improvement. Since Theorem 3.14 provides an expression to compute the improvement it is possible to compute the optimal N_J as

$$N_J^* = \underset{N_J \in [1, J-1]}{\text{argmax}} \left\{\text{voteImprovement}(N_J; N, S, J, \sigma_w, \sigma_x)\right\}, \tag{3.24}$$

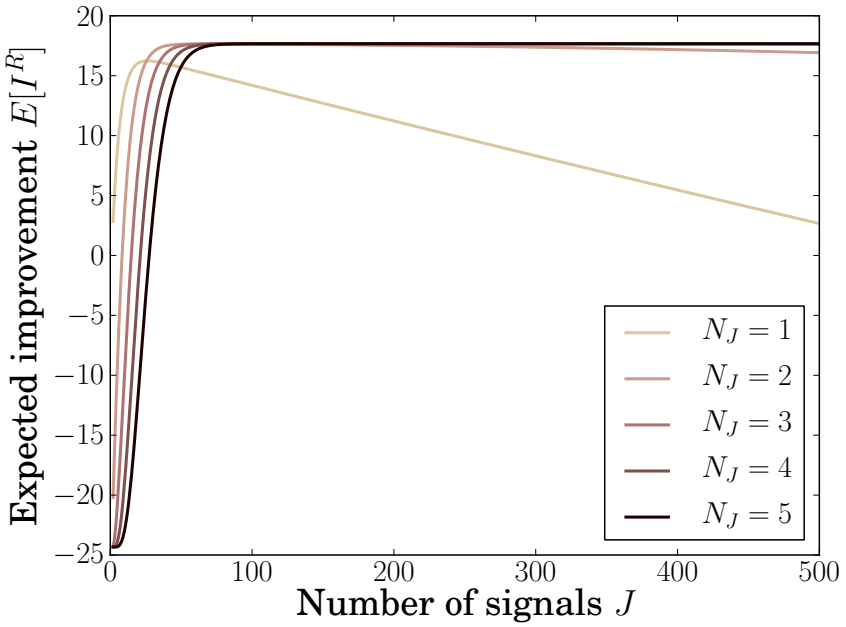


Figure 3.5: Expected improvement of voting over independent estimator, for different number of signals J and different values of N_J. We fix the signal length to $N = 1024$, the sparsity level to $S = 50$, the standard deviation of the nonzero coefficients to $\sigma_x = 1$, and the standard deviation of the noise to $\sigma_w = 0.4$. The threshold T is set to $T = \sigma_w\sqrt{2\log N} = 1.49$. We observe that for a given J, there is a value of N_J that maximizes the improvement.

where voteImprovement is the function induced by Eq. (3.18). Note that computing N_J^* only involves evaluating the improvement function $J-1$ times and then selecting the N_J leading to the larger value. It is, thus, easy to compute.

Figure 3.6 shows the expected improvement of the voting estimator, this time using the optimal number of votes N_J^*, calculated using Eq. (3.24). For comparison we also show the expected improvement for the veto estimator. We observe that the voting estimator using an optimal number does not exhibit the undesired asymptotically behavior of the veto estimator.

3.3 Experimental Results

To validate the proposed estimators and their corresponding analysis, we first test our approach using synthetic signals. In the following experiments we consider signals sparse

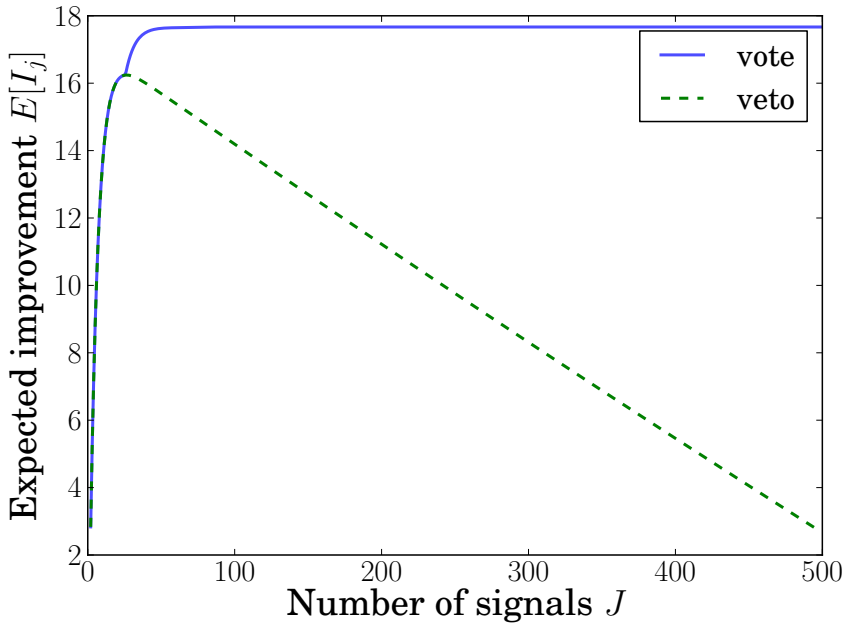
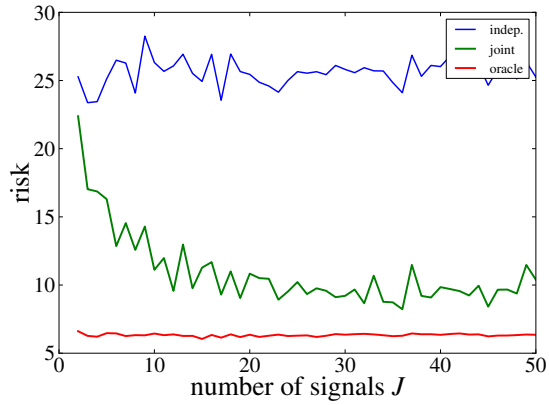


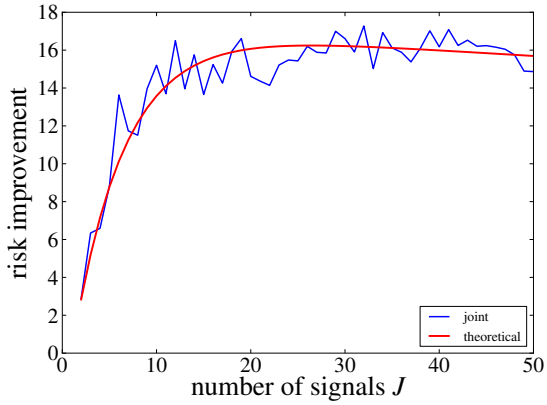
Figure 3.6: Expected improvement of voting over independent estimator, for different number of signals J for N_J set to the optimal value N_J^*. We fix the signal length to $N = 1024$, the sparsity level to $S = 50$, the standard deviation of the nonzero coefficients to $\sigma_x = 1$, and the standard deviation of the noise to $\sigma_w = 0.4$. The threshold T is set to $T = \sigma_w\sqrt{2\log N} = 1.49$. For comparison we also show the improvement of the veto over the independent estimator for the same conditions. We observe that the voting estimator with optimal N_J exhibits the desired asymptotic behavior lacking in the veto estimator.

in the time domain (this corresponds to setting Ψ equal to the identity matrix). We set the signal length to $N = 2048$ and the sparsity level to $S = 100$. The location of the nonzero coefficients is chosen uniformly at random, and their amplitudes are i.i.d. drawn at random from a standard Gaussian distribution with zero mean and variance $\sigma_\theta^2 = 1$. The observations are corrupted by Gaussian Gaussian i.i.d. zero-mean additive white noise with variance $\sigma_w^2 = 0.237$.

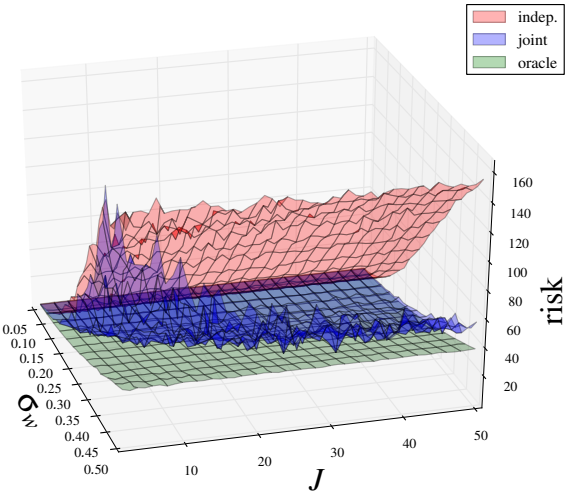
Figure 3.7(a) shows the risk for both methods together with the risk for the oracle estimator. Notice that in this case, as the number of signals J increases, the veto denoising risk gets close to the oracle risk. Figure 3.7(b) shows the theoretical and simulation values for the risk improvement. We observe that the results from the experiment match the risk improvement predicted by Theorem 3.13.



(a) Risk



(b) Risk improvement



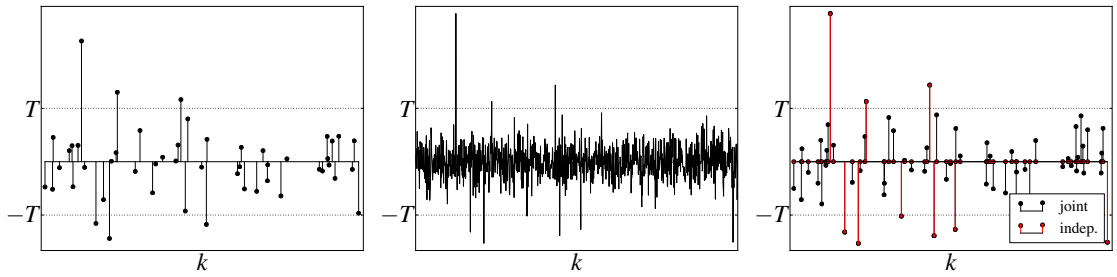
(c) Risk

Figure 3.7: Simulation results for a signal ensemble sparse in the time domain for different values of the number of signals J. (a) Risk for independent, veto, and oracle estimator. (b) Experimental and theoretical risk improvement. (c) Effect of the noise variance on the risk.

In the second experiment we consider different values of the noise variance σ_w^2 and number of signals J. Figure 3.7(c) shows the risk for the oracle, independent and veto denoising[7]. We observe that the improvement of our method is more significant at higher levels of signal noise. In other words, the noisier the observations, the greater are the benefits of exploiting the inter-signal structure.

In the third experiment we show the behavior of joint denoising by showing the outcome for one of the signals. We show the original signal (Fig. 3.8(a)), the noisy observations (Fig. 3.8(b)), and the outcome of independent and joint denoising (Fig. 3.8(c)). We observe

[7]For the values of J used in this experiment, the veto and that vote estimators are the same.



(a) (b) (c)

Figure 3.8: Simulation results for one of the signals in the ensemble. (a) Original signal. (b) Noisy observation. (c) Joint and independent estimates.

that joint denoising saved many of the small coefficients that otherwise would be killed by the independent denoising.

In the fourth experiment we validate the result of Theorem 3.14. Fig. 3.9 shows the risk of the voting estimator, with the number of votes set to the optimal N_J^*, for different values of J. For comparison, we also show the risk of the independent, veto, and oracle estimators. We also show the improvement of the voting and vetoing estimators together with their corresponding theoretical values, as predicted by Theorems 3.14 and 3.13, respectively. This result confirm the soundness of the theorems; it also shows that the asymptotic behavior of the voting estimator gets very close to the one of the oracle.

In the final experiment we validate our solution using real world signals. We used the dataset from the temperatures collected in the Intel Berkeley Research lab.[8] These signals are sparse in the Fourier domain, and satisfy approximately the JSM-2 model. Figure 3.10 shows the results. We observe that our method[9] does a better job than independent denoising, particularly at the locations corresponding to higher frequencies. This is because the higher frequency components are relatively small, thus, are killed by the independent denoising algorithm.

[8] http://www.select.cs.cmu.edu/data/labapp3/index.html

[9] For this example the veto and voting estimators are the same, since $N_J^* = 1$ for the given signal parameters.

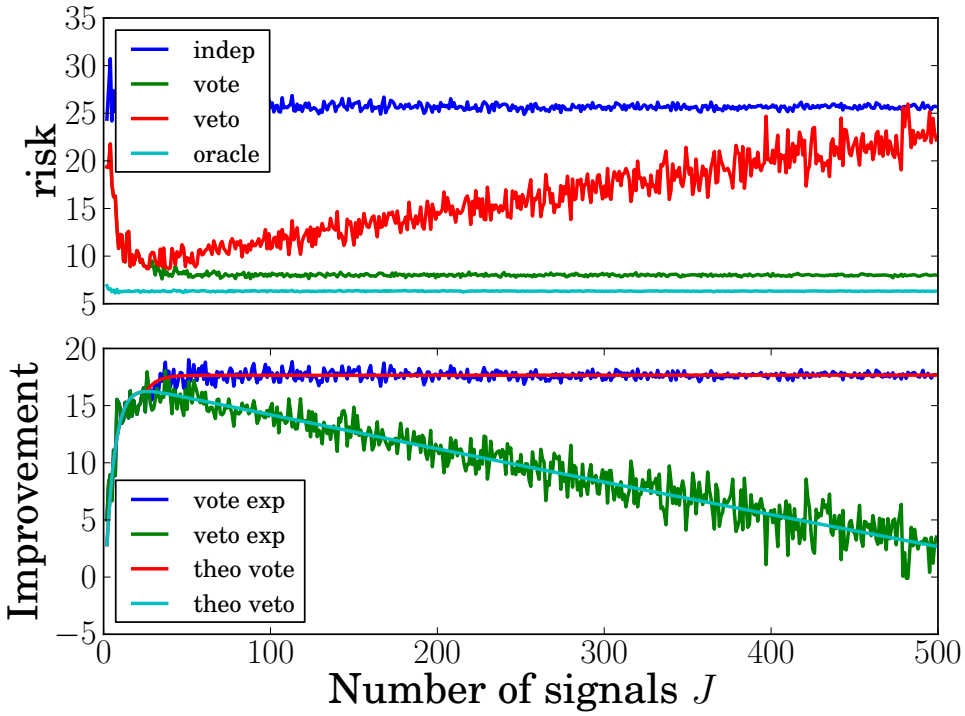
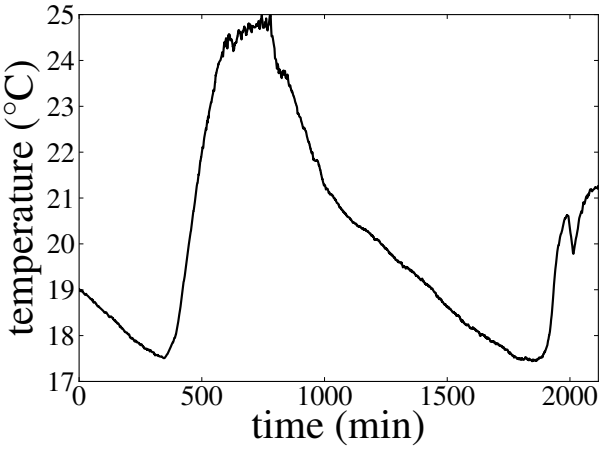


Figure 3.9: Simulation results for the voting estimator. The first panel shows the risk for different values of the number of signals J. For comparison we also shows the performance of the veto and oracle estimator. The second panel shows the corresponding improvements over the independent estimator, together with the expected improvement predicted by Theorems 3.14 and 3.13. Note that the risk of the voting estimators gets very close to the risk of the oracle estimator as the number of signals increases.

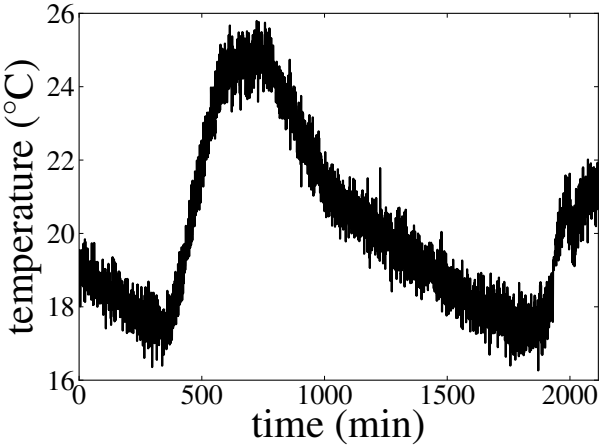
3.4 Remarks

In this chapter we proposed two methods to denoise a signal ensemble. The first one, a veto scheme, exhibited good performance, but its asymptotic behavior was not as good as desired. The second one, a voting scheme, exhibited a behavior uniformly better than the veto scheme, including a good asymptotic behavior.

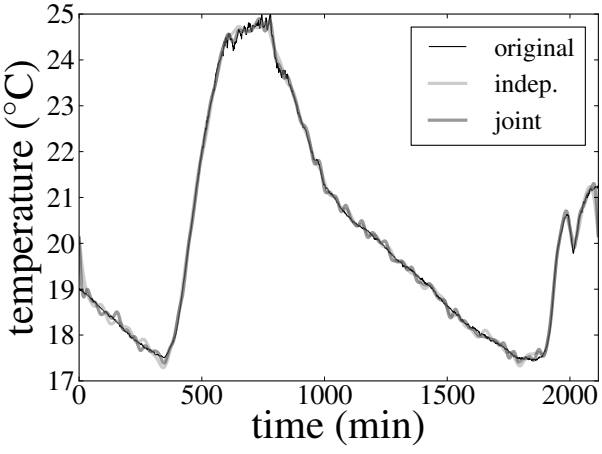
There are several aspects of the joint denoising problem that deserve further considerations. Firstly, we would like to extend the proposed methods to other joint sparse models [5], as the JSM-1 model—where each signal in the ensemble can be decomposed in a common sparse component plus a sparse innovation component—and the JSM-3 model—where each signal in the ensemble can be decomposed in a dense common component plus a sparse



(a)



(b)



(c)

Figure 3.10: Simulation results for temperature signals from a sensor network. (a) One of the original signals. (b) Noisy observation. (c) Joint and independent estimates.

innovation component. Secondly, we would like to study an alternative approach to joint denoising, based on the framework of hypothesis testing. At this point it is not clear what kind of performance such methods would exhibit with respect to the ones proposed so far. For this reason, we think it would be interesting to explore the use of this formulation.

CHAPTER 4

DECLIPPING A SIGNAL IN SPARSELAND

> "Nothing in nature is random ... A thing appears random only through the incompleteness of our knowledge."
>
> Spinoza

In many practical situations, either because a sensor has the wrong dynamic range or because signals arrive that are larger than anticipated, it is common to record signals whose amplitudes have been clipped. Any method for restoring the values of the clipped samples must—implicitly or explicitly—assume some model for the structure of the underlying signal. For example, one of the first attempts to "de-clip" a signal was the work of Abel and Smith [1], who assumed that the underlying signal had limited bandwidth relative to the sampling rate (i.e., that it was oversampled) and recovered the original signal by solving a convex feasibility problem. Godsill et al. [51] later tackled the de-clipping problem using a parametric model and a Bayesian inference approach. Along the same lines, Olofsson [76] proposed a maximum *a posteriori* estimation technique for restoring clipped ultrasonic signals based on a signal generation model and a bandlimited assumption.

Meanwhile, recent research in fields such as Compressive Sensing (CS) [19] has shown the incredible power of sparse models for recovering certain signal information. Many signals can be naturally assumed to be sparse in that they have few non-zero coefficients when expanded in a suitable basis; the name "Sparseland" has been used to describe the broad universe of such signals [43]. Although a typical CS problem involves an incomplete set of random measurements (as opposed to a complete—but clipped—set of deterministic samples), sparse models have made a limited appearance in the de-clipping literature. In particular, Gemmeke [49] et al. imputed noisy speech features by considering the spectrogram of the signal

as an image with missing samples, representing the spectrogram in terms of an overcomplete dictionary, and using sparse recovery techniques to recover the missing samples.

It is also interesting to note that observing only the sign of a signal—or equivalently registering its zero crossings—can be considered as an extreme case of clipping. Several works in "1-bit CS" [12, 55, 82] consider measurements of the form $y = \text{sign}(\Phi x)$. However, this setup is different than the one we consider, since it involves recording the sign of the signal *after* it has been multiplied by a random matrix Φ.[1] Our scenario is also different than the one considered by Boufounos and Baraniuk [13], who tackled the problem of reconstructing sparse signals from their zero crossings. In this case, the key difference with respect to our work is that they restrict the problem to periodic analog signals whose nonzero Fourier series coefficients all fall within one octave of frequency.[2] In contrast, we formulate the problem purely in the discrete domain and make no assumptions regarding the locations of the nonzero Discrete Fourier Transform (DFT) coefficients.

In this work, we present two methods for de-clipping a signal under the assumption that the original signal is sparse in the frequency domain, i.e., that it can be represented as a concise sum of harmonic sinusoids. This model is general enough to embrace a wide set of signals that could be recorded from certain communication systems, resonant physical systems, etc. This model is also commonplace in the CS literature, particularly in settings involving random time-domain measurements. Note that our choice of this frequency domain model is an important one, as many other sparse bases could not be used for de-clipping. Consider, for instance, a signal that is 1-sparse in the Haar wavelet domain (see Fig. 4.1 for an example); it is evident that any amount of clipping will render impossible the recovery of the signal. Additionally, in any wavelet basis, a nonzero coefficient corresponding to a

[1] The exact nature of the random measurement operator Φ appears to play an important role in 1-bit CS. For instance, recovery is possible when Φ has independent standard Gaussian entries, but not when it has independent Bernoulli entries [82].

[2] To illustrate the limitations of using the zero crossings as the only available observations, consider the signals $x_1(n) = \sin(2\pi n/N)$ and $x_2(n) = \sin(2\pi n/N) + 0.25\sin(2\pi 3n/N)$ (see Fig. 4.6 for a plot of x_2). Since the zero crossings of x_1 and x_2 are the same, it would be impossible to discern between these two signals using only this information.

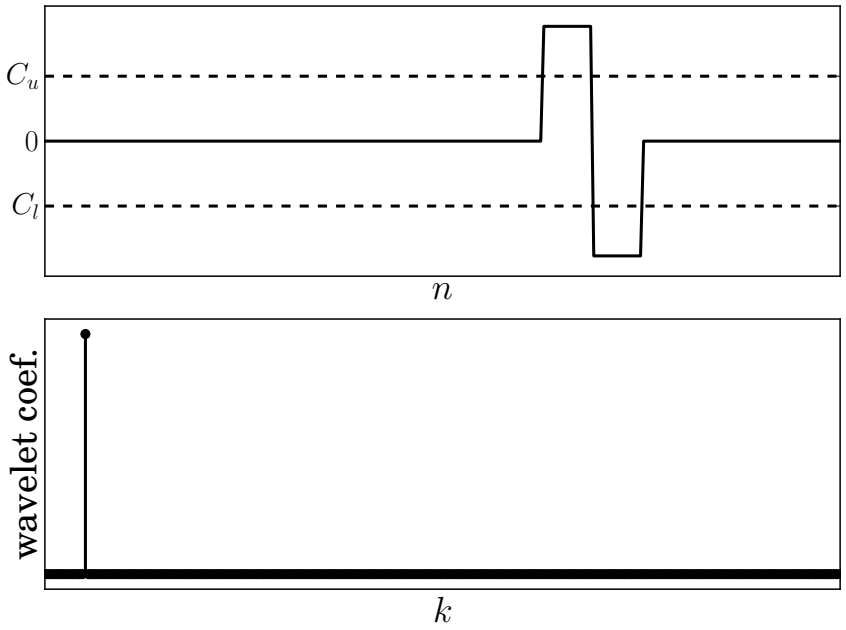


Figure 4.1: A 1-sparse signal sparse in the Haar wavelet domain. This signal can be represented by one nonzero Haar wavelet coefficient. Any amount of clipping makes the recovery of the original signal impossible.

wavelet whose support is entirely contained in a clipped region cannot generally be recovered. To some extent, this issue can be understood using the concept of mutual incoherence [36]: as is known in CS, to successfully recover a sparse signal from incomplete measurements, the sparsifying basis must be uncorrelated with the domain where the observations are taken. The clipping process takes samples in the time domain, and the wavelet basis consists of functions that are also localized in time. On the other hand, the Fourier basis is known to be maximally incoherent with the time domain.

Although the measurements we consider are not random—in fact they are "adversarial" in that clipping eliminates the samples with the highest energy content—we do find that certain ideas from CS can be leveraged. In particular, we have modified several CS algorithms in an attempt to account for the clipping constraints. In this work, we first consider a modified version of the canonical recovery algorithm known as Basis Pursuit (BP) [43], and defined as (P_1) in Sec. 2.2. This modified version, dubbed Basis Pursuit with Clipping Constraints (BPCC), is essentially the same recovery algorithm proposed by Mansour et

al. [69] in the framework of image desaturation. Since by design they select image patches with small numbers of saturated samples, however, they do not evaluate its behavior for a wide range of clipping levels. Smaragdis [88] also proposed to recover a clipped signal using an optimization problem equivalent to BPCC. Since Smaragdis' approach selectively attenuates the samples, however, the non-clipped observations are significantly different than the ones considered in our work.[3] Next, we consider a modified version of Reweighted ℓ_1 minimization [21] that uses BPCC in the inner loop and a modified version of the Thresholding algorithm [43], also known as Trivial Pursuit (TP) [5]. By a considerable margin, these are the two methods with the best performance among all the alternatives that we consider. This is surprising since (i) the performance improvements using Reweighted ℓ_1 minimization are much more substantial than are typically observed in CS problems, and (ii) TP, a very simple greedy algorithm, is one of the poorest performing algorithms in conventional CS problems [43]. We also show that, when tested on frequency sparse signals, these two methods outperform *constraint*-Orthogonal Matching Pursuit (OMP). The work in this chapter was published in [103].

4.1 Preliminaries

Let $x \in \mathbb{R}^N$ be a K-sparse signal in the Fourier domain, i.e., $x = \Psi\alpha$ and $\|\alpha\|_0 = K$, where Ψ is the $N \times N$ inverse DFT matrix and $\|\cdot\|_0$ denotes the number of non-zero entries of a vector.[4] Because of the Hermitian symmetry property of real signals, the sparsity level K is in general twice the number of harmonics in x (the exceptions are harmonics of frequency 0 or π, which contribute only one DFT coefficient each). Let the clipped version of x be x_c,

[3] Note that although similar, this is different than the problem considered in this work, since Smaragdis' approach requires modifying the sampling architecture, e.g., by using two synchronized Analog to Digital Converters (ADCs) with different gains.

[4] We focus on the purely discrete setting. While one could consider the vector x as arising from sampling an analog signal, issues concerning the selection of the sample rate and duration are beyond the scope of our work.

where

$$x_c(n) = \begin{cases} C_u & \text{if } x(n) \geq C_u, \\ C_l & \text{if } x(n) \leq C_l, \\ x(n) & \text{otherwise,} \end{cases}$$

and C_u and C_l are the known upper and lower clipping values, respectively. Our goal is to recover the original signal x from the observed clipped signal x_c.

Denote by Ω_u, Ω_l, and Ω_{nc} the index sets of the upper clipped, lower clipped, and non-clipped samples, respectively:

$$\Omega_u = \{n | x_c(n) = C_u\},\ \Omega_l = \{n | x_c(n) = C_l\},\ \Omega_{nc} = (\Omega_u \cup \Omega_l)^c.$$

Similarly, denote by Ψ_u and Ψ_l the matrices formed with the rows $i \in \Omega_u$ and $j \in \Omega_l$ of Ψ, respectively. We can write the non-clipped values of x_c as $y = \Phi x$, where Φ is a *restriction operator* formed with the rows $j \in \Omega_{nc}$ of the $N \times N$ identity matrix.

4.1.1 Basis Pursuit, Basis Pursuit with Clipping Constraints, and Reweighted ℓ_1 with Clipping Constraints

The canonical CS method for recovering a sparse signal is known as Basis Pursuit [43], and defined as (P_1) in Sec. 2.2. Given a set of non-clipped linear measurements $y = \Phi x = \Phi\Psi\alpha$, Basis Pursuit involves solving the following convex optimization problem:

$$\begin{aligned} \alpha = &\underset{\alpha \in \mathbb{C}^N}{\operatorname{argmin}} \|W\alpha\|_1 \\ &\text{s.t.} \quad \Phi\Psi\alpha = y, \end{aligned} \tag{BP}$$

where W is a diagonal weighting matrix with the norm of the columns of $\Phi\Psi$ in its main

diagonal and zeros elsewhere.[5] In the de-clipping problem, we also know that samples clipped by the upper limit must have values greater or equal than C_u, and samples clipped by the lower limit must have values smaller or equal than C_l. We can then propose a version of Basis Pursuit with clipping constraints:

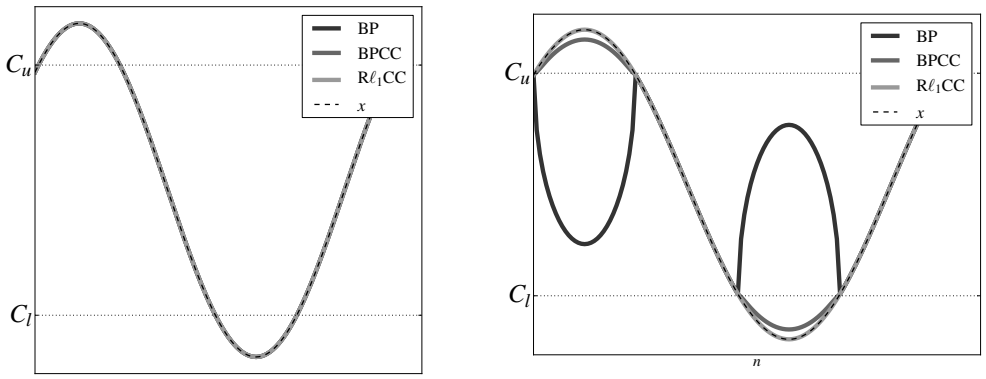
$$\begin{aligned} \alpha = & \underset{\alpha \in \mathbb{C}^N}{\operatorname{argmin}} \|W\alpha\|_1 \\ \text{s.t.} \quad & \Phi\Psi\alpha = y \\ & \Psi_u\alpha \geq C_u \\ & \Psi_l\alpha \leq C_l. \end{aligned} \tag{BPCC}$$

Another technique commonly used in CS is "Reweighted ℓ_1 minimization" [21]. In its original formulation, this method iterates over a weighted version of (BP), adjusting the weights based on the solution obtained in the previous iteration. This method typically has better signal recovery performance than Basis Pursuit but at the expense of a higher computational load. We adapt this method to the de-clipping problem by replacing (BP) at each iteration with (BPCC). We dub this method Reweighted ℓ_1 with Clipping Constraints (Rℓ_1CC). Algorithm 6 shows the complete method.

Algorithm 6 Reweighted ℓ_1 minimization with clipping constraints (Rℓ_1CC)

input: Φ, Ψ, Ψ_u, Ψ_l, y, C_l, C_u, ℓ_{max}, ϵ, δ
  $\ell = 1$, $W_i^{(1)} = 1$, $i = 1, \ldots, N$
  repeat
    $\alpha^{(\ell)} = \arg\min \|W^{(\ell)}\alpha\|_1$
      s.t. $\Phi\Psi\alpha = y$, $\Psi_u\alpha \geq C_u$, $\Psi_l\alpha \leq C_l$
    $W_i^{(\ell+1)} = \frac{1}{|\alpha_i^{(\ell)}|+\epsilon}$, $i = 1, \ldots, N$
    $\ell = \ell + 1$
  until $\ell \geq \ell_{max} + 1$ OR $\|\alpha^{(\ell)} - \alpha^{(\ell-1)}\|_2 < \delta$
output: $\alpha^{\ell-1}$

[5]The definition of (P_1) in Sec. 2.2 does not consider the weighting matrix W since in the standard CS setting the columns of $\Phi\Psi$ are normalized; in the declipping problem we do not have the degrees of freedoms to make this happens.



(a) (b)

Figure 4.2: *Reconstruction of* $x(n) = \sin(2\pi n/N + \pi/4)$ *by* (BP), (BPCC), *and Algorithm 1 (R*ℓ_1*CC). (a) Clipping level* ± 0.75*. All three approaches recover the signal. (b) Clipping level* ± 0.72*. Only R*ℓ_1*CC recovers the signal exactly.*

We test these three approaches with the signal

$$x(n) = \sin(2\pi n/N + \pi/4) \tag{4.1}$$

for $N = 128$. Figure 4.2(a) shows the result for a clipping level of ± 0.75, at which there are 70 non-clipped samples, and Fig. 4.2(b) shows the result for a clipping level of ± 0.72, at which there are 66 non-clipped samples. These numbers of non-clipped samples[6] correspond to the transition between the recovery and non-recovery zones of operation of (BP) and (BPCC).

In this experiment and in others (see Sec. 4.3), we observe that adding clipping constraints to Basis Pursuit does not help to perfectly recover signals with lower clipping thresholds. Rℓ_1CC, on the other hand, can recover signals with more significant levels of clipping. This improvement of Rℓ_1CC over (BP) and (BPCC) is actually substantially better than is typically observed in CS [21].

Thinking in terms of CS principles, the Restricted Isometry Property (RIP) (see Sec 2.2 and [19] for more details) is commonly used for theoretical analysis of compressive mea-

[6]Due to the nature of this signal $x(n)$, it is not possible to set the clipping level so that the number of non-clipped samples is between 66 and 70.

surement operators. The RIP can be shown to hold with high probability for a randomly generated matrix with a small number of rows, and when it holds, such a matrix can be used to exactly recover any sparse signal (up to a certain sparsity level). This perspective is not the right one to analyze the de-clipping problem, however, and it cannot be used to explain why (BP) and (BPCC) fail in the previous example. First, since the matrix $\Phi\Psi$ is not random, we cannot use any of the standard probabilistic tools to predict whether it will satisfy the RIP. Second, while the RIP guarantees that a fixed measurement matrix can be used to recover *any* sparse signal, in the de-clipping problem the matrix $\Phi\Psi$ is relevant *only* for the small set of signals that, when clipped, actually produce the samples given by this matrix. In other words, Φ itself is dependent on the unknown signal x. This dependency is not only unusual in CS, it is also contrary to what makes a measurement matrix favorable in CS: while random matrices tend to capture a representative sample of signal entries, both large and small, the clipping process deliberately excludes all of the large signal entries and keeps only the small ones.

4.1.2 About the Uniqueness of the Solutions

Mangasarian [68] provides some characterizations to determine if a linear program has a unique solution. Since the optimization programs (BP) and (BPCC) can be cast as linear programs, we use one of these characterizations to study the uniqueness of the solutions. In particular, let the linear program and its dual be

$$\begin{array}{lll} \underset{x}{\text{minimize}} & p^T x & \underset{u,v}{\text{maximize}} \quad b^T u + d^T v \\ \text{subject to} & Ax = b & \text{subject to} \quad A^T u + C^T v = p \\ & Cx \geq d & \qquad\qquad v \geq 0, \end{array}$$

and denote its solutions by $\bar{x}$, $\bar{u}$ and $\bar{v}$. Let C_I be a matrix formed by the rows $i \in I$ of C,

and define the following index sets:

$$K = \{i | \bar{v}_i > 0\} = \{i | C_i \bar{x} = d_i, \bar{v}_i > 0\}$$
$$L = \{i | C_i \bar{x} = d_i, \bar{v}_i = 0\}.$$

Then, the linear program has a unique solution if and only if the rows of $[A^T \; C_K^T \; C_L^T]$ are linearly independent and the linear program

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \mathbf{1}^T C_L x \\ \text{subject to} \quad & Ax = 0 \\ & C_k x = 0 \\ & C_L x \geq 0 \end{aligned}$$

has a zero maximum. The convex program (BP) can be written as the aforementioned linear program by defining

$$p = \begin{bmatrix} \mathbf{0} & \mathbf{1} \end{bmatrix}^T, \quad A = \begin{bmatrix} \Phi\Psi & \mathbf{0} \end{bmatrix}, \quad b = y,$$
$$C = \begin{bmatrix} W & I \\ -W & I \end{bmatrix}, \quad d = \mathbf{0}.$$

For the convex program (BPCC), we need to change the definition of C and d as

$$C = \begin{bmatrix} W & I \\ -W & I \\ \Psi_u & \mathbf{0} \\ -\Psi_l & \mathbf{0} \end{bmatrix}, \qquad d = \begin{bmatrix} \mathbf{0} \\ C_u \mathbf{1} \\ -C_l \mathbf{1} \end{bmatrix}$$

We now analyze the uniqueness of the solution of the problems introduced in Sec. 4.1.1.[7] Table 4.1 shows the recovery—declared as successful when the output of the optimization problem is equal, up to numerical precision, to the original signal—and the uniqueness—determined by the Mangasarian test—of the convex programs (BP) and (BPCC), at the two clipping levels where the recovery transition happens. As suggested previously, we can see, at least for this example, that the "recovery transition" coincides with the "uniqueness transition".

Table 4.1: Recovery and uniqueness of the solution using (BP) and (BPCC), for the 1-sparse signal defined by Eq. (4.1).

Method	Measurements	Correct recovery	Uniqueness	$\|\alpha\|_1$
P1	66	No	No	11.3
	70	Yes	Yes	11.3
P2	66	No	No	11.3
	70	Yes	Yes	11.3

Does this behavior generalize to other sparsity levels? We try with a 2-sparse signal defined as

$$x(n) = \sin(2\pi n/N) + 0.5\sin(2\pi 3n/N)\,. \tag{4.2}$$

For this signal the recovery transition occurs when the number of measurements decreases from 76 to 74. Table 4.1 shows the uniqueness, as computed by the Mangasarian test, and the ℓ_1 norm of the solutions. As can be seen, now all the solutions are unique.

To get a more definite answer about this issue, we perform a more thorough experiment, where we analyze the relationship between recovery and uniqueness for different values of the sparsity level and the number of measurements. For comparison, we also show the results for standard Compressive Sensing on the same unclipped signal using measurements in the time domain, for the same levels of sparsity and number of measurements. For each combination

[7] Appendix A describes a sanity check used to check the correctness of our implementation.

Table 4.2: Recovery and uniqueness of the solution using (BP) and (BPCC), for the 2-sparse signal defined by equation (4.2).

Method	Measurements	Recovery	Uniqueness	$\|\alpha\|_1$
P1	74	No	Yes	10.6
	76	Yes	Yes	12.0
P2	74	No	Yes	11.9
	76	Yes	Yes	12.0

of sparsity level and number of measurements, we check if we can recover the original signal, and if the solution to the optimization problem is unique. Table 4.3 shows the symbols used to indicate the four possible outcomes, and Figure 4.3 shows the results.

Table 4.3: Symbols used to indicate recovery and uniqueness.

Recovery	Uniqueness	Symbol
No	No	□
No	Yes	△
Yes	No	○
Yes	Yes	◇

We can observe that for all sparsity methods, and the three methods, all the solutions are unique. This experiment suggests that the lack of uniqueness associated with the non-recovery condition only happens for single-tone signals. It remains an open question why this phenomenon is happening, and if it has any significance.

4.2 Trivial Pursuit with Clipping Constraints

Let us note that the DFT of the clipped signal x_c generally contains, in addition to the harmonics introduced by the clipping process, all of the harmonics present in the original signal x. Interestingly, the harmonics with the biggest magnitude typically coincide with the ones from the original signal. Figure 4.4 shows an example for a signal with sparsity level

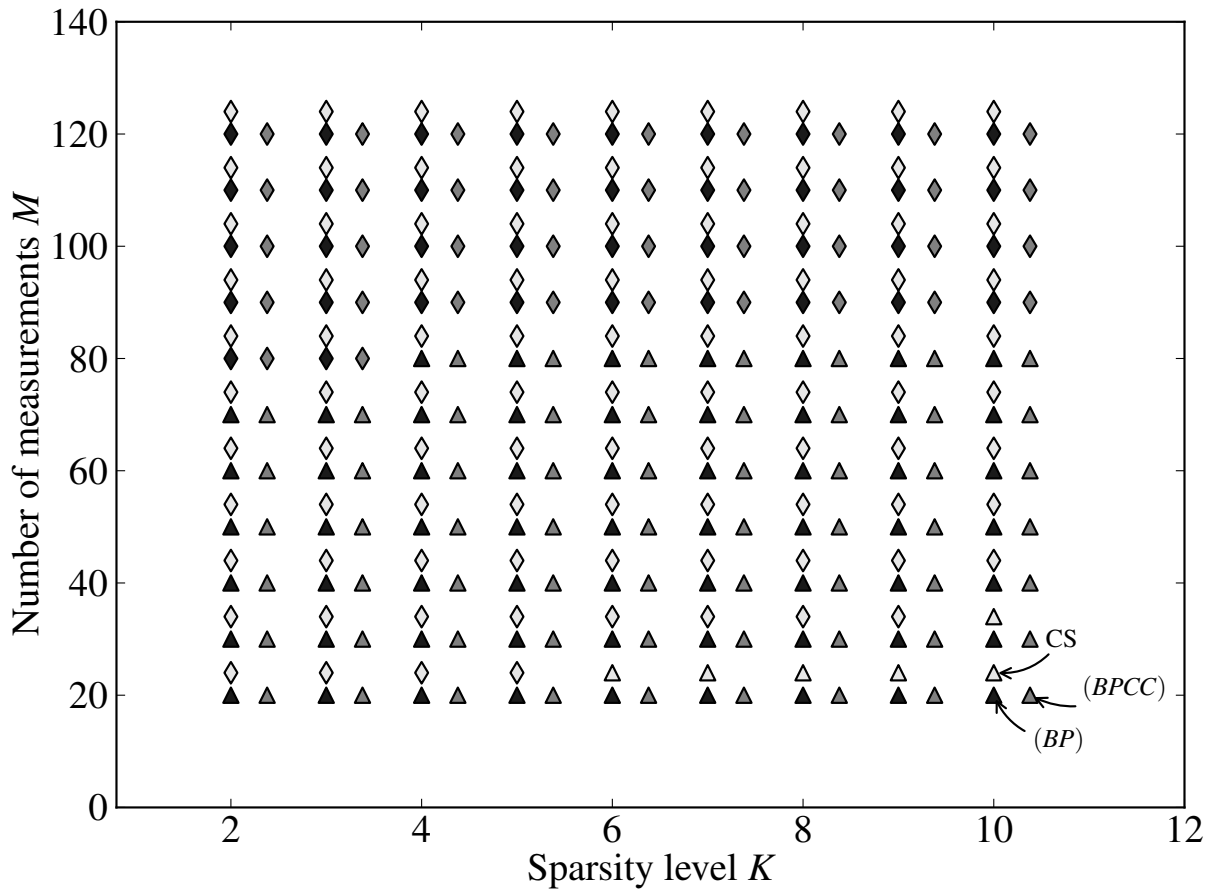
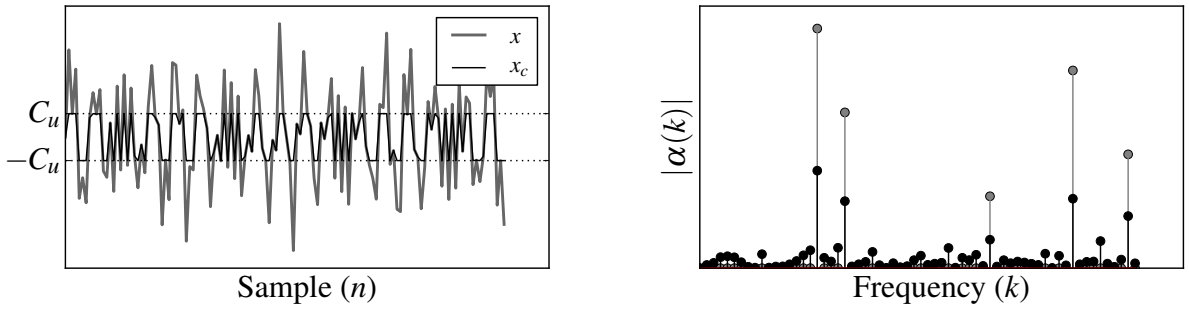


Figure 4.3: Uniqueness analysis. For each sparsity K and number of measurements M, we show the recovery and uniqueness of the solutions of (BP), (BPCC) and the equivalent Compressive Sensing (CS) problem. See Table 4.3 for the meaning of the symbols.

$K = 10$ clipped at an amplitude corresponding to 20% of its peak value. We see that the 5 biggest harmonics of x_c are at the same locations as the 5 harmonics of x. Why is it that the harmonics of the original signal are present in its clipped version? Intuitively, one can think of what happens when we listen to saturated audio signals: in spite of being distorted, we can still understand the signal content. This situation suggests that the harmonics of the original signal are still present. A bit more formally, this phenomenon can be explained by decomposing the clipped signal as $x_c = x + x_d$, where x_d represents the distortion introduced by the clipping process. Figure 4.5 shows an example of such decomposition. Since



(a) (b)

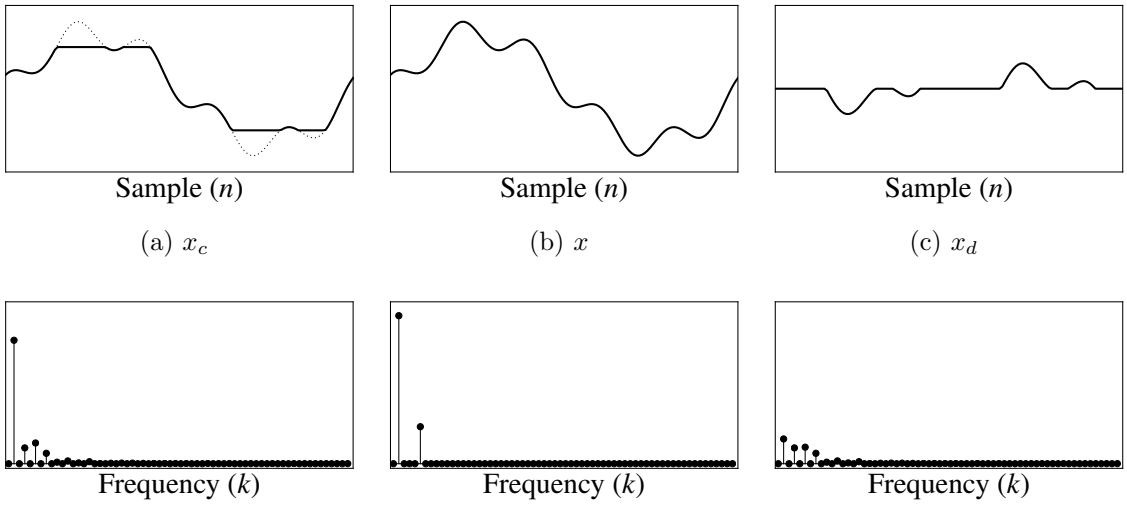
Figure 4.4: *Support estimation using the DFT of the clipped signal. (a) A signal x with sparsity level $K = 10$ and its clipped version x_c, with $C_u/\|x\|_\infty = 0.2$, corresponding to $M = 40$ non-clipped samples. (b) DFT of x and x_c for $0 \leq k < \frac{N}{2}$. Note that the 5 biggest harmonics of x_c are at the same locations as the harmonics of x.*

$\text{DFT}\{x_c\} = \text{DFT}\{x\} + \text{DFT}\{x_d\}$, the harmonics of x will generally be present in x_c.[8] Our second proposed de-clipping algorithm exploits this observation.

The method is very simple and consists of two stages. First, we identify the support (the location of the non-zero Fourier coefficients) of the signal. Second, we estimate the value of the coefficients on this support using a least-squares approach, similar to that used in other greedy methods such as Matching Pursuit or OMP [43]. If we know the sparsity level K *a priori*, we can estimate the support simply by finding the K biggest harmonics of x_c. In the more general case where we do not know K, we select the elements of the support one at a time in a greedy manner, until the reconstruction error on the non-clipped samples is small enough.

Algorithm 7 shows a detailed description of the method. In the **match** step we compute the DFT of the clipped signal—*this happens only once*. Then we repeat the following steps until the residual r is arbitrary small (we use $\epsilon = 10^{-6}$ in our experiments). In the **identify** step we add the indices associated with the current largest harmonic to the support index set Λ, and we then set those coefficients to zero to avoid selecting them again. In the **update** step we compute the DFT coefficients of a signal—restricted to the support Λ—that best

[8]Note that at some point the amount of distortion introduced by the clipping process will render it impossible to distinguish the harmonics of x.



(a) x_c (b) x (c) x_d

(d) $|\alpha_c|$ (e) $|\alpha|$ (f) $|\alpha_d|$

Figure 4.5: *Decomposition of a clipped signal. A clipped signal x_c can be decomposed as $x_c = x+x_d$. Since the DFT is a linear transformation, the DFT of the clipped signal can be decomposed in the same way. (a) Clipped signal x_c. (b) Original signal x. (c) Distortion term x_d. (d)-(f) Absolute value of the DFT coefficients of x_c, x, and x_d for $0 \leq k < \frac{N}{2}$.*

approximates the non-clipped samples y in a least-squares sense. Note that the coefficients α_Λ on this support are easily computed as $\alpha_\Lambda = (\Phi\Psi)_\Lambda^\dagger y$, where $(\Phi\Psi)_\Lambda^\dagger$ is the pseudoinverse of the columns of $\Phi\Psi$ indexed by Λ.

Although perhaps not evident at first sight, Algorithm 7 corresponds to a modified version of the method known as Trivial Pursuit (TP) [5, 43]. Given a set of non-clipped linear measurements

$$y = \Phi x = \Phi\Psi\alpha,$$

TP would estimate the support of α simply by computing the score $h_{TP} = (\Phi\Psi)^T y$ and selecting the indices of the largest entries of h_{TP}. We can write

$$h_{TP} = (\Phi\Psi)^T y = \Psi^T \Phi^T y$$

and note that the vector $\Phi^T y \in \mathbb{R}^N$ corresponds to a zero-padded version of y with the non-clipped samples at the proper locations. Since multiplying by Ψ^T is equivalent to computing

Algorithm 7 Trivial Pursuit with Clipping Constraints

input: Φ, Ψ, x_c, y, ϵ
  initialize: $r = y$, $\Lambda^{(1)} = \emptyset$, $\ell = 1$
  match: $h = \text{DFT}\{x_c\}$ (indexed from 0 to $N-1$)
  while $\|r\|_2 > \epsilon$ **do**
    identify: $k = \text{argmax}_{0 \le j \le \frac{N}{2}} |h(j)|$
    $\Lambda^{(\ell+1)} = \Lambda^{(\ell)} \cup \{k, (N-k) \mod N\}$
    $h(k) = 0$
    update: $\alpha = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Lambda^{(\ell+1)}} \|y - \Phi\Psi z\|_2$
    $r = y - \Phi\Psi\alpha$
    $\ell = \ell + 1$
  end while
output: $\widehat{x} = \Psi\alpha = \Psi\, \text{argmin}_{z:\ \text{supp}(z) \subseteq \Lambda^{(\ell)}} \|y - \Phi\Psi z\|_2$

the DFT of a vector, h_{TP} is in fact the DFT of the zero-padded version of y. The vector h computed in the **match** of Algorithm 7 is actually very similar to h_{TP}, except that instead of computing the DFT of the zero-padded version of y, we compute the DFT of x_c, which is equal to y padded with the clipped values instead of zeros. In other words, Algorithm 7 exploits the knowledge of the clipped values. For this reason we dub our method Trivial Pursuit with Clipping Constraints (TPCC).

To illustrate the effectiveness of TPCC we experiment with the signal $x(n) = \sin(2\pi n/N) + 0.25\sin(2\pi 3n/N)$ of length $N = 128$. We clip this signal, shown in Fig. 4.6, at a level just below the "bumps". It might seem impossible to recover the signal once the oscillations due to the third harmonic are missing. Remarkably, however, TPCC not only recovers this signal at the clipping level of $C_u = 0.7$, but it can even recover this signal down to the clipping level of $C_u = 0.2$, at which point there are only 10 non-clipped samples.

Although in CS TP is arguably the simplest reconstruction method for sparse signals, it is also one of the methods with the poorest performance in terms of the number of measurements required for successful signal recovery [43]. For this reason it is quite surprising that in this experiment and in others (see Sec. 4.3) TPCC can be so effective for de-clipping sparse signals.

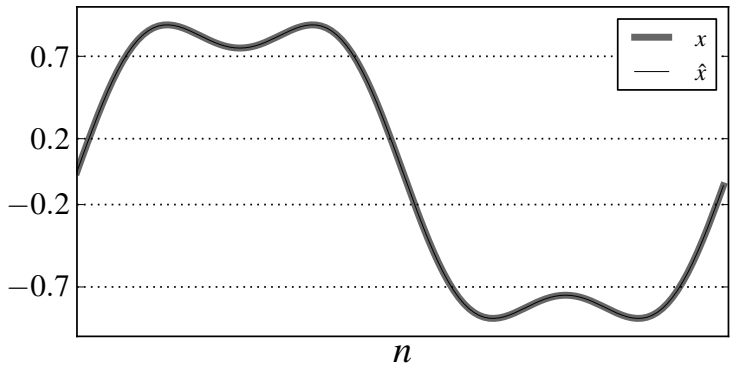


Figure 4.6: *Recovering a two-tone signal using TPCC. The clipping level $C_u = 0.7$ is just below the high-frequency "bumps". It is possible to recover the signal with a clipping level down to $C_u = 0.2$. We set the signal length to $N = 128$.*

4.3 Experimental Results

In this section we empirically evaluate the methods described previously. We also compare with *constraint*-OMP.[9] For all experiments that follow we generate, for each value of the sparsity level K, signals of length $N = 128$ having K non-zero coefficients with frequencies selected randomly, amplitudes chosen randomly from a uniform distribution between 0.5 and 1.5, and phases selected randomly. We also set $C_l = -C_u$.[10]

In the first experiment, we find the average minimum number M_{min} of non-clipped samples required to recover a signal as a function of K. We compute the average over 100 simulation runs. Figure 4.7(a) shows the results. BP and BPCC perform very poorly, being unable to recover the original signal except when the clipping is very mild. *Constraint*-OMP performs better, while TPCC and Rℓ_1CC perform much better still. In fact, both TPCC and Rℓ_1CC can reliably recover the signal using a number of non-clipped samples that is not much larger than K, while BP and BPCC require a number of non-clipped samples much closer to N.

In the second experiment, we compare Rℓ_1CC and TPCC in a different way. We fix

[9] We have found that *constraint*-OMP exhibits better performance with signals sparse in the Direct Cosine Transform (DCT) domain than with signals sparse in the DFT domain. We thus use the DCT as the sparsity basis for testing this method.

[10] MATLAB code is available at https://github.com/aweinstein/declipping.

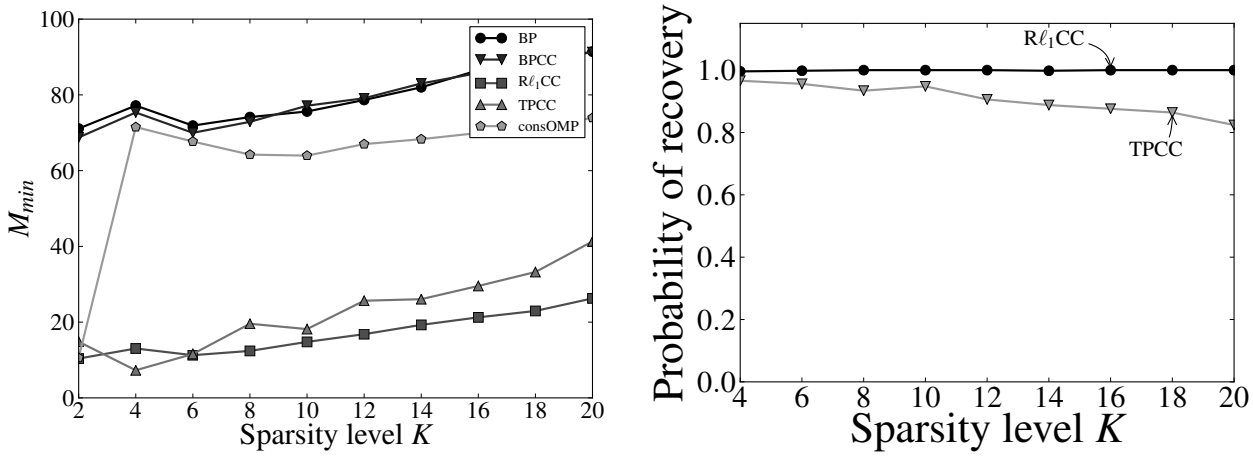
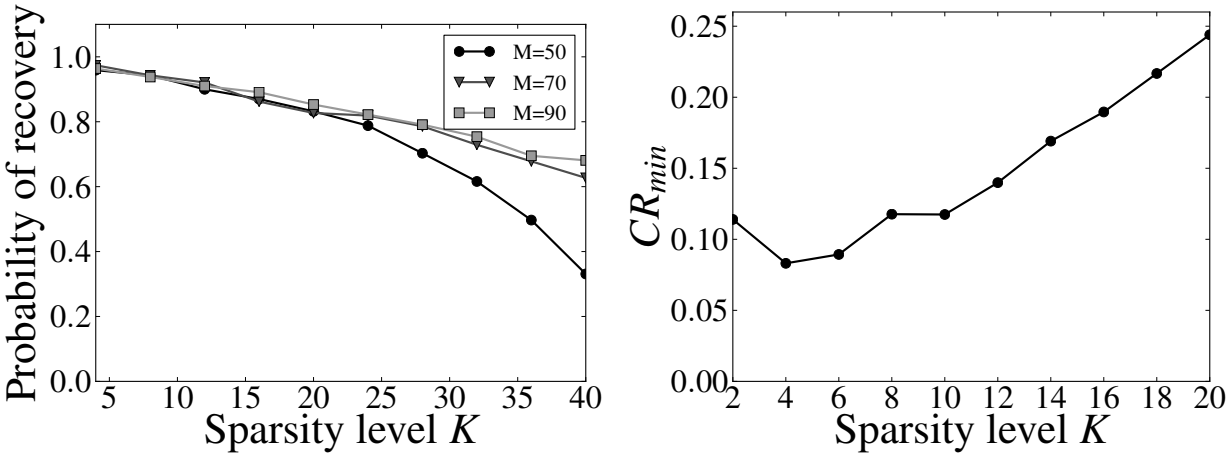


Figure 4.7: *Recovering a clipped signal using BP, BPCC,* constraint-*OMP, Rℓ_1CC, and TPCC. (a) The average minimum number of non-clipped samples M_{min} required to recover signals of different sparsity levels K. (b) The probability of perfect recovery as a function of the sparsity level K for $M = 70$ non-clipped samples.*

the number of non-clipped samples to $M = 70$ and plot the probability of perfect recovery (declared when $\|x - \widehat{x}\| \leq 10^{-3}$) as a function of the sparsity level K. Figure 4.7(b) shows the results using 500 trials. Although Rℓ_1CC performs somewhat better than TPCC, it is important to underscore that TPCC requires significantly fewer computations.

In the two next experiments we examine the performance of TPCC more closely. We plot the probability of perfect recovery as a function of K for different values of M. Figure 4.8(a) shows the results using 500 simulation runs for each combination of M and K. As expected, the probability of recovery increases as the sparsity level K decreases and as the number of non-clipped samples M increases. Again, in general, we can expect a high probability of recovery from TPCC (over our random signal model) when M is a small multiple of K. Another relevant figure of merit is the clipping ratio $CR = C_u/\|x\|_\infty \in [0, 1]$. Over 1000 simulation runs, we find the average minimum clipping ratio CR_{min} required for signal recovery as a function of K. Figure 4.8(b) shows the result. Even for relatively high values of K, TPCC is able to recover signals with clipping levels smaller than $\frac{1}{4}$ of the maximum absolute signal value. The apparent anomaly that for $K = 2$ the performance is worse than



(a) (b)

Figure 4.8: *Recovering a clipped signal using TPCC. (a) The probability of perfect recovery as a function of the sparsity level K for different numbers of non-clipped samples M. (b) The average minimum clipping ratio required to recover signals of different sparsity levels K.*

for $K = 4$ is explained by the fact that samples of a high frequency single-tone signal tend to concentrate around a few values (the limit case being $x(n) = \cos(\pi n)$). Thus, for such signals, a small amount of clipping translates to a small number of observations.

It is also possible to extend TPCC to the case where the original signal x is contaminated by additive noise. All we need to do, as is commonly done with greedy methods [43, Sec. 5.31], is to modify the value of ϵ used in the stopping condition of Algorithm 7 as a function of the noise level. In particular, we consider the noisy signal $x_n = x + z$, where z is a bounded noise term with $\|z\|_2 < \delta$. We observe the clipped signal x_c equal to x_n if $C_l < x_n < C_u$, equal to C_u if $x_n \geq C_u$, and equal to C_l if $x_n \leq C_l$.

We illustrate the effectiveness of this approach with two signal realizations. Both signals have sparsity level K equal to 10 and fixed support (equal to the 10 lowest frequencies). We fix the noise level $\|z\|_2$ to 1 and ϵ to 1. Figure 4.9(a) shows the simulation result for a signal with 54 non-clipped samples, and Fig. 4.9(b) shows the result for a signal with 46 non-clipped samples.

Next, we examine the performance of TPCC under noisy observations in more detail.

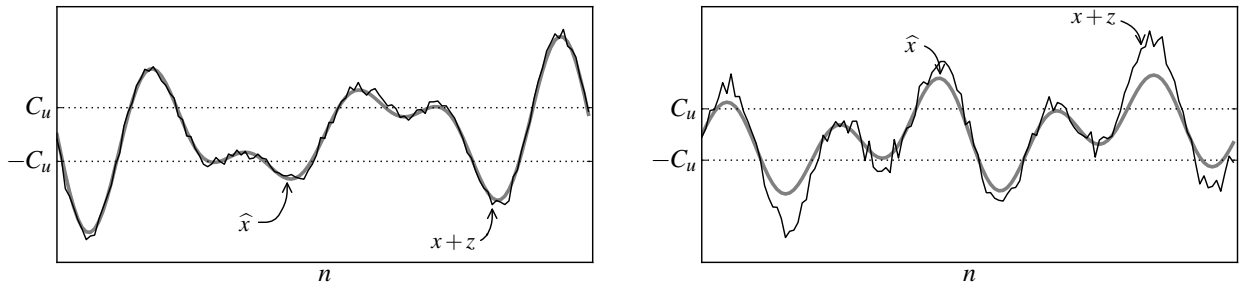


Figure 4.9: *Recovering a clipped noisy signal using TPCC. We plot the original noisy signal $x+z$ and the recovered signal $\widehat{x}$ for two signal realizations. We fix both the noise level $\|z\|_2$ and ϵ to 1. The number of non-clipped samples is (a) 54 and (b) 46.*

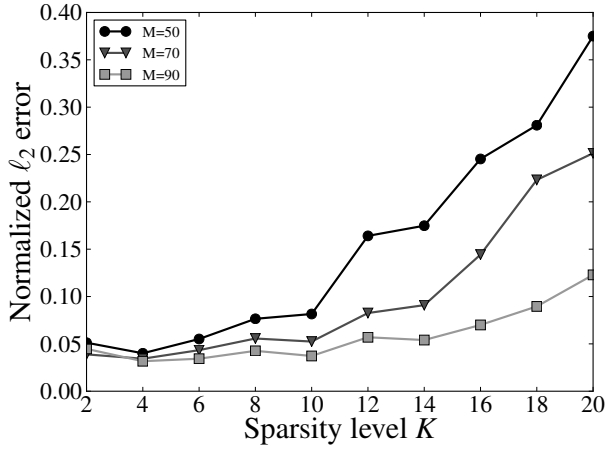


Figure 4.10: *Recovering a noisy clipped signal using TPCC. We plot the average normalized ℓ_2 error $\|x-\widehat{x}\|_2^2/\|x\|_2^2$ over 500 simulation runs as a function of the sparsity level K for different numbers of non-clipped samples M. We set both the noise level $\|z\|_2$ and ϵ to 1.*

In this experiment, for a fixed number of non-clipped samples M and noise level $\|z\|_2$, we compute the average normalized ℓ_2 error $\|x-\widehat{x}\|_2^2/\|x\|_2^2$ over 500 simulation runs as a function of the sparsity level K. We set both the noise level $\|z\|_2$ and ϵ to 1. Figure 4.10 shows the results. We observe a behavior similar to the case of noise-free clipped signals, where performance degrades as the sparsity level K increases.

CHAPTER 5

SPARSE MODELS FOR REINFORCEMENT LEARNING

> "Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."
>
> Richard Bellman [39].

Reinforcement Learning (RL) is a branch of machine learning[1] [8, 16, 83, 92, 93]. It considers an *agent* that interacts with a given *environment*. The agent is able to take actions and to observe its current state. After taking an action, the agent observes its new state together with an immediate reward—this reward does not need to be positive, if negative, it can be considered as a cost. The goal is to design a policy, or control law, such that the sum of all the observed rewards is maximized. This problem is challenging because typically to maximize the total reward the agent needs to take actions that do not always look promising. This is why it is common to say that in RL "things need to get worse before they get better". This is also challenging because we are interested in cases where the agent does not have access to a model of the environment; all it can do is to interact with it.

RL has been applied successfully in different domains. An early success case was *TD-Gammon* [96], a program that learned to play Backgammon. It is also common to use RL to solve classic control problems (e.g., controlling an inverted pendulum [92]), in robotics (autonomous helicopter [74], obstacle avoidance [70]), and in Operations Research (e.g., maintenance with limited resources, channel allocation in cellular systems [8]).

As discussed in Sec. 5.2, solving an RL problem requires the use of a function approx-

[1] RL is closely related to the concept of *Dynamic Programming* (DP). The exact meaning of DP is not well defined (see the epigraph at the beginning of this chapter and [39]). Commonly it is said that DP provides a set of techniques to solve an RL problem given a perfect model of the environment [92, Ch. 4].

imation scheme. Among the several function approximation architectures typically used in RL, the linear approximation approach is one of the most common ones. An important step in any linear approximation solution is the design of the feature vectors (or alternatively, the design of an approximation basis). Typically, this step involves designing these features or basis functions by hand, and it can become quite involved as the problem at hand becomes more complex. For this reason, researchers have focused on simplifying this step.

In this chapter we show how the use of sparse approximations [43] helps to alleviate the difficulties practitioners encounter when designing an approximation architecture. After reviewing some of the sparse approximation solutions proposed so far in the literature, we show a new approach that exploits the additional structure existing in the functions of interest.

The chapter follows with a description of Markov Decision Process (MDP) and their use to formalize the RL problem. Section 5.2 explains the use of function approximations in RL. Section 5.3 describes the role of sparse approximation in RL, and introduces the new proposed algorithms. We finalize with some empirical results.

5.1 Markov Decision Processes

In RL the interaction between the agent and the environment is modeled by an MDP. An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R \rangle$. The finite set[2] $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ denotes all the possible states where the agent can be, while the finite set $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ denotes the set of actions available to the agent. The agent interacts with the environment by taking actions sequentially. The outcome of taking an action is governed by the transition probability function $\mathcal{P}(s, a, s')$ that returns the probability that the next state is s', given that the current state is s, and that the agent executed action a. The agent also gets a

[2]Although this formulation considers a finite number of states, it is possible to deal with countable and continuous state environments by using function approximation schemes (see Sec. 5.2).

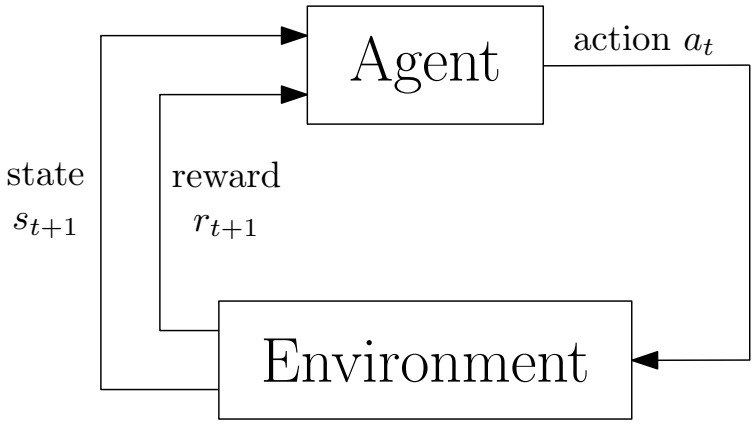


Figure 5.1: The agent-environment interaction. At time t and based on its current state, the agent executes action a_t. Then it gets a reward and observes the new state s_{t+1} and immediate reward r_{t+1}.

reward[3] $R(s, a, s')$ determined by the function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Figure 5.1 summarizes the interaction between the agent and the environment.

Let $t = 0, 1, 2, \ldots$ and let $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, $r_t \in \mathbb{R}$ denote the sequence of states, actions, and rewards observed and executed by the agent, respectively. The agent objective is to maximize the *discounted return*

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ is the *discount factor*, with $0 < \gamma < 1$.

At this point it is useful to clarify the following. The word "Markov" is used to describe systems that satisfy the *Markov property*, that is, systems for which the next state depends only on the present state. In addition to MDPs, other systems that also exhibit this property are Markov Chains, Hidden Markov Models (HMMs), and Partially Observed Markov Decision Processes (POMDPs). The difference among these class of systems lays in two facts: if it is possible to have control over the state transitions, and if it is possible to observe the state completely. Figure 5.2 shows the relationship between these facts and the

[3]It is also possible to formulate the MDP using a reward function defined as $R(s, a)$ or as $R(s)$. Given an MDP defined using one of these definition, it is always possible to transform it into an equivalent MDP that uses one of the other reward function definitions [86, Ex. 17.5].

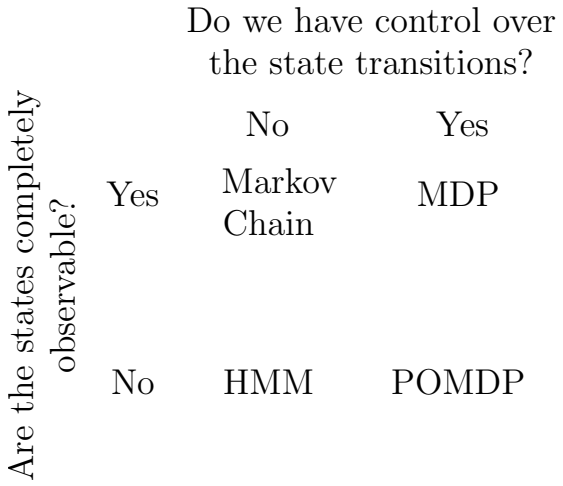


Figure 5.2: Depending on if it is possible to have control over the state transitions and if it is possible to observe the states completely, systems that satisfy the Markov property can be classified as Markov Chains, Markov Decision Processes (MPDs), Hidden Markov Models (HMMs) or Partially Observable Markov Decision Processes (POMPDs) (original concept of this figure taken from http://www.cassandra.org/pomdp/pomdp-faq.shtml).

four class of systems.

5.1.1 The Chain Environment

To make the discussion concrete, we introduce the *chain environment* [61]. In this environment the state space is given by $\mathcal{S} = \{1, 2, \ldots, N\}$, and the agent can choose between taking the action '`Left`' (L) or '`Right`' (R), i.e., $\mathcal{A} = \{L, R\}$. The states are ordered in ascending order from left to right. If the agent takes the action '`Right`' ('`Left`') the next state will be the one to its right (left) with probability 0.9 and the one to its left (right) with probability 0.1; the exception to this rule being when the agent is at one of the chain ends ($s = 1$ or $s = N$), in which case actions may cause the agent to "bounce back" to the same

state. In summary, the transition probability function is given by

$$\begin{aligned}
\mathcal{P}(s, L, s+1) &= 0.1 && s = 1, \ldots, N-1, \\
\mathcal{P}(s, L, s-1) &= 0.9 && s = 2, \ldots, N, \\
\mathcal{P}(s, R, s+1) &= 0.9 && s = 1, \ldots, N-1, \\
\mathcal{P}(s, R, s-1) &= 0.1 && s = 2, \ldots, N, \\
\mathcal{P}(1, L, 1) &= 0.9, \\
\mathcal{P}(1, R, 1) &= 0.1, \\
\mathcal{P}(N, R, N) &= 0.9, \\
\mathcal{P}(N, L, N) &= 0.1.
\end{aligned}$$

The agent gets a reward of 1 when it reaches the state $s = 1 + \lfloor N/5 \rfloor$ or the state $s = N - \lfloor N/5 \rfloor$ and a reward of 0 otherwise. Thus, the reward function is

$$R(s, a, s') = \begin{cases} 1, & s' = 1 + \lfloor \frac{N}{5} \rfloor, \\ 1, & s' = N - \lfloor \frac{N}{5} \rfloor, \\ 0, & \text{otherwise.} \end{cases}$$

Figure 5.3 shows an instance of the chain environment for $N = 5$.

5.1.2 The Four-Rooms Grid Environment

Another environment that we use in this chapter is the *four-rooms grid*. In this environment the state space is given by $\mathcal{S} = \{1, \ldots, N\}$, with the states organized as a two-dimensional grid with G_N rows and G_M columns, and $G_N \times G_M = N$ the largest factors of N—for instance, if the number of states is set to $N = 40$, the number of rows is set to

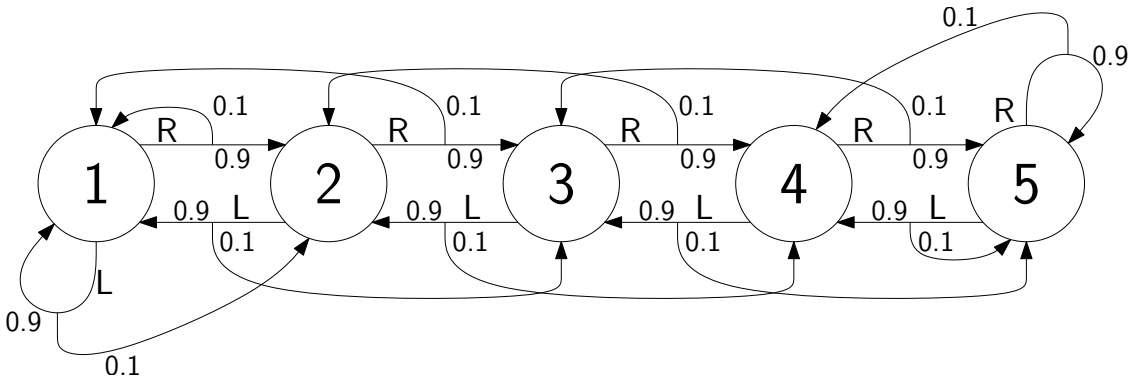


Figure 5.3: A chain environment with five states. Each arrow represents a possible action, left (L) or right (R). The bifurcation of the arrows follow the possible next states. The numbers indicate the transition probabilities.

$G_N = 8$, and the number of columns is set to $G_M = 5$. The grid represents four interconnected rooms. See Fig 5.4 for an example with the number of states set to $N = 60$. The mapping between the state $s \in \mathcal{S}$ and its (i, j) grid location is given by the *row-major order* [71, Sec. 7.9.2]

$$s \mapsto \left(\left\lceil \frac{s}{G_M} \right\rceil, (s \bmod G_M) + 1 \right), \qquad (i, j) \mapsto (i-1)G_M + j.$$

Note that using row-major order is an arbitrary decision; what is relevant is to be consistent when translating from s to (i, j), and vice versa.

We consider two different options for the action space. (i) The agent can move in the four cardinal directions, i.e., $\mathcal{A} = \{N, W, S, E\}$. (ii) In addition to the cardinal directions the agent can also move in the four ordinal directions, i.e., $\mathcal{A} = \{N, NW, W, SW, S, SE, E, NE\}$. We call agents able to move in the eight cardinal plus ordinal directions "king move" agents, due to the similarity with the movements of the corresponding chess piece.

5.1.3 Policies and Value Functions

The agent executes actions according to a policy function $\pi : \mathcal{S} \to \Omega(\mathcal{A})$, where $\Omega(\mathcal{A})$ is the set of all probability distributions over $\mathcal{A}$. Let $\pi(s, a)$ be the probability of choosing action

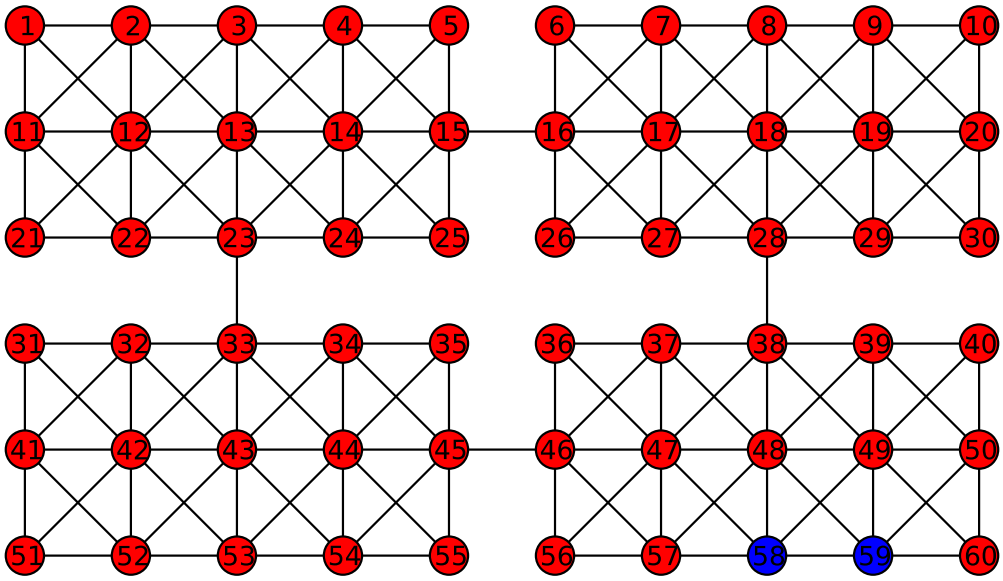


Figure 5.4: Four-room grid environment for a king moves agent. The state space $\mathcal{S} = \{1, \dots N\}$ is organized as a two-dimensional grid representing four interconnected rooms. The mapping between the state and the location in the grid is given by the row-major order. Shown in blue are the two goal states. In this example the number of states is set to $N = 60$.

a after observing that the current state is s. Although the theoretical derivation considers stochastic policies, in practice we usually only consider deterministic policies where, for a given state, the agent always selects the same action, i.e., $\pi(s, a) = 1$ if a is the action selected by the deterministic policy and 0 otherwise. In such cases we denote the selected action as $a = \pi(s)$.

Central to the RL problem is to evaluate the quality of a given policy. This evaluation is commonly done using *value functions*, which are functions that indicate "how good" a state is, in terms of the expected return that can be obtained by beginning in that state and executing a given policy.

Let the *state-value function for policy* π be

$$V^{\pi}(s) = \mathbf{E}_{\pi}\left[R_t \;\middle|\; s_t = s\right] = \mathbf{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s\right].$$

For any policy π and state s the state-value function satisfies the following recursive

relationship [92]:

$$\begin{aligned} V^{\pi}(s) &= \mathbf{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right] \\ &= \mathbf{E}_{\pi}\left[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_t = s\right] \\ &= \sum_{a\in\mathcal{A}(s)}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\left[R(s,a,s') + \gamma\mathbf{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_{t+1} = s'\right]\right] \\ &= \sum_{a\in\mathcal{A}(s)}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\left[R(s,a,s') + \gamma V^{\pi}(s')\right]. \end{aligned} \tag{5.1}$$

Equation (5.1) is known as *Bellman's equation for* V^{π} [8, 92]. Notice that it is a linear system of equations, with $|\mathcal{S}|$ unknowns and $|\mathcal{S}|$ equations[4], and it can be written in matrix form as

$$V^{\pi} = \mathcal{R} + \gamma\Pi_{\pi}PV^{\pi}, \tag{5.2}$$

where Π_{π} is an $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ matrix given by

$$\Pi_{\pi} = [\operatorname{diag}(\pi(s_1,a_1),\ldots,\pi(s_{|\mathcal{S}|},a_1))|\cdots|\operatorname{diag}(\pi(s_1,a_{|\mathcal{A}|}),\ldots,\pi(s_{|\mathcal{S}|},a_{|\mathcal{A}|}))],$$

P is an $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ matrix given by

$$P = \begin{bmatrix} P_{a_1} \\ \vdots \\ P_{a_{|\mathcal{A}|}} \end{bmatrix}, \quad P_{a_k}(i,j) = P(s_i,a_k,s_j), \quad a_k \in \mathcal{A},$$

and $\mathcal{R}$ is a vector of length $|\mathcal{S}|$ with entries given by

$$\mathcal{R}(i) = \sum_{a\in\mathcal{A}}\pi(s_i,a)\sum_{s'\in\mathcal{S}}P(s_i,a,s')R(s_i,a,s') \qquad i = 1,\ldots|\mathcal{S}|.$$

[4]It can be shown that it has a unique solution [92].

By defining the *Bellman operator* as $(T_\pi)(\cdot) = \mathcal{R} + \gamma \Pi_\pi P(\cdot)$, Eq. (5.2) becomes $V^\pi = T_\pi V^\pi$. Note that this means that the value function V^π is a fixed-point of the Bellman operator.[5]

Example 5.1

Consider the chain environment with $N = 5$ states, $\gamma = 0.9$, and deterministic policy[6] $\pi = [R\ R\ L\ L\ L]$. We have

$$\Pi_\pi = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0.9 & 0.1 & 0.0 & 0.0 & 0.0 \\ 0.9 & 0.0 & 0.1 & 0.0 & 0.0 \\ 0.0 & 0.9 & 0.0 & 0.1 & 0.0 \\ 0.0 & 0.0 & 0.9 & 0.0 & 0.1 \\ 0.0 & 0.0 & 0.0 & 0.9 & 0.1 \\ 0.1 & 0.9 & 0.0 & 0.0 & 0.0 \\ 0.1 & 0.0 & 0.9 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.0 & 0.9 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.0 & 0.9 \\ 0.0 & 0.0 & 0.0 & 0.1 & 0.9 \end{bmatrix}$$

$$\mathcal{R} = \begin{bmatrix} 0.9 \\ 0.0 \\ 1.0 \\ 0.0 \\ 0.9 \end{bmatrix} \qquad V^\pi = \begin{bmatrix} 5.2 \\ 4.7 \\ 5.2 \\ 4.7 \\ 5.2 \end{bmatrix}.$$

[5] This fact is useful, since by studying the properties of this operator, it is possible to infer some of the properties of V^π [8].

[6] For a deterministic policy we summarize the policy by a vector, under the understanding that $\pi(s_i)$ is equal to the ith entry of the vector.

Matrix Π_π is built by concatenating two diagonal matrices, each diagonal matrix with the probabilities of choosing '`Left`' and '`Right`' for each state, respectively. Given that for the first state $\pi(1) = R$, i.e., the agent chooses action '`Right`' with probability one, there is a 1 in the first entry of the first location of the second diagonal matrix (note that the actions are ordered arbitrarily as $[L, R]$). The construction of the remaining rows follows the same logic.

Matrix P encodes the transition probabilities. The first five rows corresponds to probabilities conditioned on action '`Left`' being selected, and the last five to probabilities conditioned on action '`Right`' being selected. For instance, the first row contains the probabilities of the next state, given that the current state is $s_1 = 1$ and action '`Left`' is selected. In this case, the probability of staying in the same state is 0.9, and the probability of moving to the state to the right is 0.1. The probabilities of reaching the remaining states are 0.

Vector $\mathcal{R}$ contains the expected rewards. In this case, for instance, when the agent is in the first state (corresponding to the first entry of the vector), it will execute action '`Right`', reaching state $s_2 = 2$ with probability 0.9 and getting a reward equal to 1, and staying in the same state and getting a reward equal to 0 with probability 0.1. Thus, the expected reward for the first state is $0.1 \cdot 0 + 0.9 \cdot 1 = 0.9$.

Finally, once Π_π, P, and $\mathcal{R}$ are built, V^π is found by solving Bellman's equation.

Many algorithms use the *action-value function for policy* π [93], defined as

$$Q^\pi(s, a) = \mathbf{E}_\pi \left[R_t \;\middle|\; s_t = s, a_t = a \right] = \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right].$$

Similarly to the value function, the action-value function can be written in matrix form as

$$Q^\pi = \mathcal{R} + \gamma P \Pi_\pi Q^\pi,$$

where Q^π and $\mathcal{R}$ are vectors of length $|\mathcal{S}||\mathcal{A}|$, and the entries of $\mathcal{R}$ are

$$\mathcal{R}(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s') R(s,a,s').$$

As before, Q^π is the fixed-point of the Bellman operator $(T_\pi)(\cdot) = \mathcal{R} + \gamma P \Pi_\pi(\cdot)$. Although similar, this is not the same operator as the one used to compute V^π. In particular the vector $\mathcal{R}$ that appears in both expressions is not the same. To avoid notation clutter, we use the same symbols for both expressions.

For a given MDP, value functions define a partial ordering over different policies, i.e., $\pi \geq \pi'$ if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$. Note that here we use the formal mathematical meaning of "partial order" [41]. That means that not every pair of policies π and π' can be related by the binary operator "$\geq$". For example, if $V^\pi(s_i) > V^{\pi'}(s_i)$ and $V^\pi(s_j) < V^{\pi'}(s_j)$ for $i \neq j$, it is not possible to order these two policies. A policy that satisfies the relationship $\pi^* \geq \pi$ for all π is called an *optimal policy*. This policy defines the *optimal state-value function*, denoted V^*, and the *optimal action-value function*, denoted $Q^*(s,a)$, defined as

$$V^*(s) = \max_\pi V^\pi(s), \qquad \forall s \in \mathcal{S},$$
$$Q^*(s,a) = \max_\pi Q^\pi(s,a), \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}(s).$$

Note that although for a given MDP the optimal state- and action-value functions are unique, optimal policies are not [92]. We can also write Q^* as a function of V^* as follows:

$$Q^*(s,a) = \mathbf{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) \;\middle|\; s_t = s, a_t = a \right].$$

Because V^* is the optimal value, its consistency condition can be written without reference to the optimal policy:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^*(s,a) \\
&= \max_a \mathbf{E}_{\pi^*} \left[R_t \;\middle|\; s_t = s, a_t = a \right] \\
&= \max_a \mathbf{E}_{\pi^*} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right] \\
&= \max_a \mathbf{E}_{\pi^*} \left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s, a_t = a \right] \\
&= \max_a \mathbf{E} \left[r_{t+1} + \gamma V^*(s_{t+1}) \;\middle|\; s_t = s, a_t = a \right] \qquad (5.3) \\
&= \max_a \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]. \qquad (5.4)
\end{aligned}
$$

Equations (5.3) and (5.4) are two forms of the *Bellman optimality equation* for V^*. The Bellman optimality equation is a system of $|\mathcal{S}|$ non-linear equations with $|\mathcal{S}|$ unknowns. This system of equations has a unique solution and can, in principle, be solved by a variety of techniques.

The Bellman optimality equation for Q^* is

$$
\begin{aligned}
Q^*(s,a) &= \mathbf{E} \left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \;\middle|\; s_t = s, a_t = a \right] \\
&= \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s', a') \right].
\end{aligned}
$$

The Bellman optimality equation for Q^* is a system of $|\mathcal{S}||\mathcal{A}|$ non-linear equations with $|\mathcal{S}||\mathcal{A}|$ unknowns. This system of equations has a unique solution and can be solved by a variety of techniques similar to the ones used to find V^* [92].

It is easy to find the optimal policy from V^*: for each state s, search for the action that maximizes the expected optimal value:

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} P(s, a, s') V^*(s').$$

Similarly, we can find the policy using $Q^*(s, a)$:

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a).$$

5.1.4 Policy Evaluation

The linear system of equations defined by the value function (5.1) can be solved by traditional linear algebra methods. However, it is useful to consider how to solve it using an iterative formulation. Let V_k denote the approximate value of $V^\pi(s)$ at iteration k. We can use equation (5.1) to write the recursion

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}(s)} \pi(s, a) \sum_{s' \in \mathcal{S}} P(s, a, s') \left[R(s, a, s') + \gamma V_k(s') \right]. \tag{5.5}$$

By definition if $V_k = V^\pi$, then V_k is a solution of the Bellman equation. This implies that it is a fixed point of the iteration rule (5.5). If the conditions for the existence of $V^\pi(s)$ are met[7], it can be shown that $V_k \to V^\pi$ as $k \to \infty$. This algorithm is called *iterative policy evaluation*.

Note that the new values of the value function depend on all the old values of the function. In practice this means that we need to keep two arrays in memory. We call this a *full backup* [92]. It is also possible to update the values "in place," using the new updates of the function value as soon as they are available. We call this a *sweep* through the states [92]. The "in-place" approach usually converges faster than the full backup. However, the particular order used to back up the states is significant, and can change the rate of

[7]The conditions are: either $\gamma < 1$ or there is a guarantee that there is an eventual termination for a given episode.

convergence [92]. As a stopping condition we use $\max_{s \in \mathcal{S}} |V_{k+1}(s) - V_k(s)| < \theta$ for some small value of θ.

5.1.5 Policy Improvement

Given a policy π, is it possible to modify it to get a better policy? To answer this let us focus first on modifying π for a single state, i.e., at the state s, select the action $a \neq \pi(s)$, and follow π for the rest of the states. Following steps similar to the ones used in equation (5.1), we can write

$$Q^{\pi}(s,a) = \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s') \right],$$

and use this expression to compute the value of executing action a. If this value is greater than $V^{\pi}(s)$, then the modified policy is better than the original. This idea is generalized by the *policy improvement theorem*. Let π and π' be any pair of deterministic policies such that

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \qquad \forall s \in \mathcal{S}.$$

Then π' must be as good or better than π, i.e.,

$$V^{\pi'}(s) \geq V^{\pi}(s) \qquad \forall s \in \mathcal{S}.$$

Using $Q^{\pi}(s,a)$ we can find a new policy π' that improves over all the states simultaneously:

$$\begin{aligned} \pi'(s) &= \operatorname*{argmax}_a Q^{\pi}(s,a) \\ &= \operatorname*{argmax}_a \sum_{s'} P(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s') \right]. \end{aligned}$$

This is known as *policy improvement*. It can be shown that if the new policy π' has the same value function as π, then π' is the optimal policy.

It is possible to improve a given policy until it becomes the optimal policy by sequentially applying a policy evaluation/improvement cycle:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*,$$

where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy iteration. This is called *policy iteration*.

5.1.6 Value Iteration

It is possible to compute the optimal value function iteratively, by using the Bellman optimality equation (5.4):

$$V_{k+1}(s) = \max_a \sum_{s'} P(s, a, s') \left[R(s, a, s') + \gamma V_k(s') \right].$$

As in the policy evaluation case, we stop the iterations when the difference between two consecutive value functions is small enough.

5.2 Function Approximation

Computing state- and action-value functions requires finding the value of these functions for all the states and state-action pairs, respectively. This becomes problematic as the environment dimension increases. Not only does the memory footprint become too large, but the time required to fill it becomes unrealistic. In addition, by approximating the value function, we are able to generalize from limited experience of a subset of the state-action space. This function approximation problem can be tackled by a variety of techniques

originating in diverse fields, such as machine learning, artificial neural networks, pattern recognition, and statistical curve fitting [8, 16, 92].

Linear approximations are among the most popular approximation architectures used in RL. In this approach the state-value function is approximated as

$$\widehat{V} = \Phi w, \tag{5.6}$$

where Φ is an $|\mathcal{S}| \times k$ matrix and w is a vector of length k. Each row of Φ is set to $\phi(s)$, where $\phi(\cdot)$ is a feature function that returns a vector of length k for each state $s \in \mathcal{S}$. Examples of features commonly used in RL includes Radial Basis Functions (RBFs) and polynomials. See Sec. 5.2.5 for details about the feature vectors used in this work. Note that using functions like RBFs and polynomials requires having a meaningful ordering of the states, otherwise notions as smoothness and location lose their meaning. Mahadevan et al. [66] introduced the notion of *proto-value functions* as an alternative to the commonly used features to ameliorate this issue.

Action-value functions can also be approximated as

$$\widehat{Q} = \Phi w. \tag{5.7}$$

In this case[8] Φ is an $|\mathcal{S}||\mathcal{A}| \times k$ matrix and w is a vector of length $k \times |\mathcal{A}|$. Each row of Φ is set to $\phi(s, a)$, where $\phi(\cdot, \cdot)$ is a feature function that returns a vector of length $k \times \mathcal{A}$ for each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$. See Sec. 5.3 for a more detailed discussion about the construction of Φ.

Two methods commonly used to find w are discussed next.

[8]The matrix Φ and vector w used to approximate the action-value function are different than the ones used to approximate the state-value function. To avoid notation clutter, we use the same symbol in both cases.

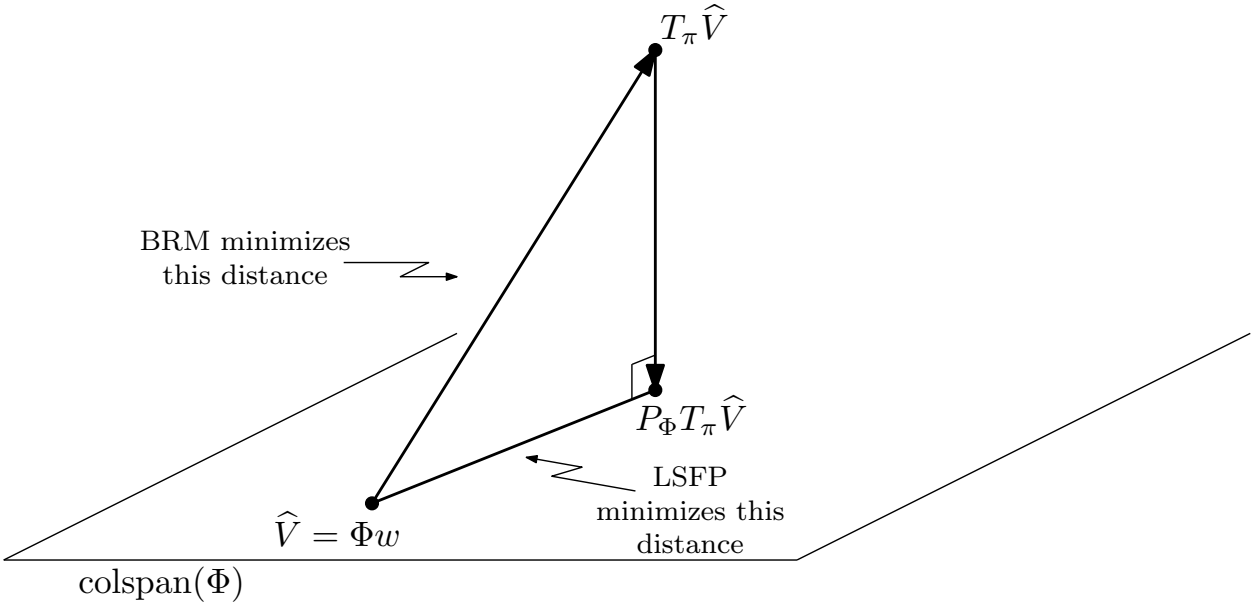


Figure 5.5: Approximation of the value-function using a linear architecture. While $\widehat{V} = \Phi w$ lives in the column span of Φ, in general $T_\pi \widehat{V}$ does not. BRM approximates $\widehat{V}$ by minimizing $\left|\left|\widehat{V} - T_\pi \widehat{V}\right|\right|$, while LSFP approximates $\widehat{V}$ by minimizing $\left|\left|\widehat{V} - P_\Phi T_\pi \widehat{V}\right|\right|$, where P_Φ is the orthogonal projection onto the column span of Φ.

5.2.1 Bellman Residual Minimizing Approximation

The state-value function V^π is approximated as $\widehat{V}^\pi = \Phi w^\pi$. One option to compute this approximation is to choose $\widehat{V}^\pi$ such that the Bellman equation is satisfied as closely as possible (see Fig. 5.5):

$$\begin{aligned} w &= \underset{w \in \mathbb{R}^k}{\operatorname{argmin}} \left|\left| \widehat{V}^\pi - \left(\mathcal{R} + \gamma P_\pi \widehat{V}^\pi\right) \right|\right|_2 \\ &= \underset{w \in \mathbb{R}^k}{\operatorname{argmin}} \left|\left| \Phi w - (\mathcal{R} + \gamma P_\pi \Phi w) \right|\right|_2 \\ &= \underset{w \in \mathbb{R}^k}{\operatorname{argmin}} \left|\left| (\Phi - \gamma P_\pi \Phi)\, w - \mathcal{R} \right|\right|_2, \end{aligned} \tag{5.8}$$

where $P_\pi = \Pi_\pi P$. Since Eq. (5.8) corresponds to a least-squares problem, w^π is given by

$$w = (\Phi - \gamma P_\pi \Phi)^\dagger \mathcal{R}. \tag{5.9}$$

The solution given by Eq. (5.9) is known as the *Bellman Residual Minimization* (BRM) approximation [61].

Following the same analysis it is possible to find the Bellman Residual Minimization approximation of the action-value function $\widehat{Q}(s,a) = \Phi w$. In this case the solution is given by

$$w = \left(\Phi - \gamma P \Pi_\pi \Phi\right)^\dagger \mathcal{R}. \tag{5.10}$$

5.2.2 Least-Squares Fixed-Point Approximation

Another option to compute the value-function approximation is to force $\widehat{V}^\pi$ to be a fixed-point of the Bellman operator, i.e., we want $\widehat{V}^\pi \approx T_\pi \widehat{V}^\pi$. In general this approximation can not be exact, since typically $T_\pi \widehat{V}^\pi$ does not live in the column span of Φ. The best we can do is to find a $\widehat{V}^\pi$ that is invariant under one application of T_π followed by the orthogonal projection onto the column span of Φ (see Fig. 5.5). The orthogonal projection onto the column span of Φ is given by $P_\Phi = \Phi(\Phi^T\Phi)^{-1}\Phi^T$. Thus, to find the approximation we solve

$$\begin{aligned}
\widehat{V} &= \Phi(\Phi^T\Phi)^{-1}\Phi^T\left(\mathcal{R} + \gamma P_\pi \widehat{V}\right) \\
\Phi w &= \Phi(\Phi^T\Phi)^{-1}\Phi^T\left(\mathcal{R} + \gamma P_\pi \Phi w\right) \\
\left(\Phi - \gamma\Phi(\Phi^T\Phi)^{-1}\Phi^T P_\pi \Phi\right) w &= \Phi(\Phi^T\Phi)^{-1}\Phi^T\mathcal{R} \\
\Phi(\Phi^T\Phi)^{-1}\left(\Phi^T\Phi - \gamma\Phi^T P_\pi\Phi\right) w &= \Phi(\Phi^T\Phi)^{-1}\Phi^T\mathcal{R} \\
\left(\Phi^T\Phi - \gamma\Phi^T P_\pi\Phi\right) w &= \Phi^T\mathcal{R}.
\end{aligned}$$

which corresponds to a $k \times k$ linear system of equations with solution

$$w = \left(\Phi^T\Phi - \gamma\Phi^T P_\pi\Phi\right)^{-1}\Phi^T\mathcal{R}. \tag{5.11}$$

The solution given by Eq. (5.11) is known as the *Least-Squares Fixed-Point* (LSFP) approximation [61]. Since both $\widehat{V}$ and $P_\Phi T_\pi \widehat{V}$ live in the column span of Φ, this solution

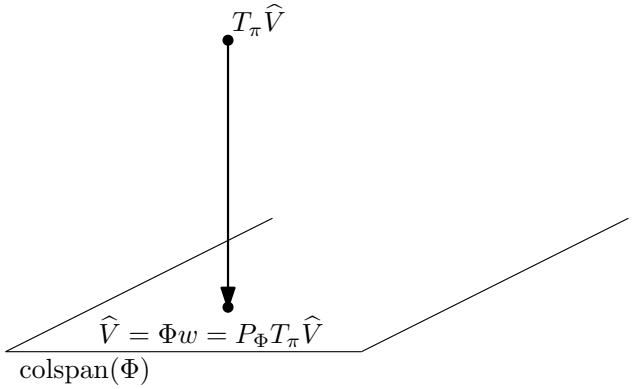


Figure 5.6: LSFP solution. Since both $\widehat{V}$ and $P_\Phi T_\pi \widehat{V}$ live in the column span of Φ, by minimizing $\left|\left|\widehat{V} - P_\Phi T_\pi \widehat{V}\right|\right|$ LSFP makes this distance equal to zero.

makes the $\left|\left|\widehat{V} - P_\Phi T_\pi \widehat{V}\right|\right|$ equal to zero (see Fig. 5.6). It follows then than finding w is equivalent to finding the fixed point of the function of w

$$f(w) = \underset{u \in \mathbb{R}^k}{\operatorname{argmin}} \, ||\Phi u - (\mathcal{R} + \gamma P_\pi \Phi w)||_2^2 \,. \tag{5.12}$$

This fact will be used later in this chapter to derive an extension to this approximation approach.

Following the same analysis it is possible to find the LSFP approximation of the action-value function $\widehat{Q}(s,a) = \Phi w$. In this case the solution is given by

$$w = \left(\Phi^T \Phi - \gamma \Phi^T P \Pi_\pi \Phi\right)^{-1} \Phi^T \mathcal{R}. \tag{5.13}$$

5.2.3 Learning Through the Agent-Environment Interaction

The methods presented in the two previous sections require access to the exact model of the underlying MDP—they require one to know Φ, P, Π_π and $\mathcal{R}$. As stated at the beginning of this chapter, we are interested in cases where we only have access to samples of the form (s, a, r, s') obtained by an agent interacting with the environment. Since we need to evaluate the action-value function rather than the state-value function (see Sec. 5.2.4), in

the sequel we focus in approximating the former. In particular, we need to find how to solve, approximately and using only the samples gathered by the agent, Eqs. (5.10) and (5.13).

Solving Eq. (5.13) is equivalent to solving the linear system of equations

$$Aw = b,$$

with

$$A = \Phi^T(\Phi - \gamma P \Pi_\pi \Phi)$$

and

$$b = \Phi^T \mathcal{R}.$$

The problem of approximating the action-value function then becomes estimating A and b from the samples, and then just solving this system of equations.

Using the definitions of Φ, P and Π_π (see Sec. 5.1.3), we can write[9] A as [61]

$$\begin{aligned} A &= \Phi^T(\Phi - \gamma P \Pi_\pi \Phi) \\ &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \phi(s,a) \left(\phi(s,a) - \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\phi(s',\pi(s')) \right)^T \\ &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \left[\phi(s,a) \left(\phi(s,a) - \gamma\phi(s',\pi(s')) \right)^T \right], \end{aligned}$$

and b as

$$\begin{aligned} b &= \Phi^T \mathcal{R} \\ &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \phi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') R(s,a,s') \\ &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \left[\phi(s,a) R(s,a,s') \right]. \end{aligned}$$

[9] In this and the following derivations we use the fact that $\sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') = 1$.

Note that A is a sum of matrices of the form

$$\phi(s,a)\left(\phi(s,a)-\gamma\sum_{s'\in\mathcal{S}}\phi(s',\pi(s'))\right)^T$$

weighted by the transition probabilities $\mathcal{P}(s,a,s')$, and that b is a sum of vectors of the form

$$\phi(s,a)R(s,a,s'),$$

also weighted by the transition probabilities $\mathcal{P}(s,a,s')$.

If the agent observes a set of samples

$$D=\{(s_i,a_i,r_i,s_i')\mid i=1,\ldots,L\},$$

then A and b can be estimated by

$$\widetilde{A}=\frac{1}{L}\sum_{i=1}^{L}\left[\phi(s_i,a_i)\left(\phi(s_i,a_i)-\gamma\phi(s_i',\pi(s_i'))\right)^T\right],$$

$$\widetilde{b}=\frac{1}{L}\sum_{i=1}^{L}\left[\phi(s_i,a_i)r_i\right].$$

It is convenient to write

$$\widetilde{A}=\frac{1}{L}\widetilde{\Phi}^T\left(\widetilde{\Phi}-\gamma\widetilde{\Phi}'\right)$$

and

$$\widetilde{b}=\frac{1}{L}\widetilde{\Phi}^T\widetilde{R},$$

with

$$\widetilde{\Phi}=\begin{bmatrix}\phi(s_1,a_1)^T\\ \vdots\\ \phi(s_L,a_L)^T\end{bmatrix},\quad \widetilde{\Phi}'=\begin{bmatrix}\phi(s_1',\pi(s_1'))^T\\ \vdots\\ \phi(s_L',\pi(s_L'))^T\end{bmatrix}\quad \widetilde{R}=\begin{bmatrix}r_1\\ \vdots\\ r_L\end{bmatrix}.$$

Algorithm 8 LSTDQ

input: $D = \{(s_i, a_i, r_i, s_i') \mid i = 1, \dots L\}$, $\phi(\cdot, \cdot)$, γ.

$\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$

$\widetilde{\Phi}' = [\phi(s_1', \pi(s_1')) \ \cdots \ \phi(s_L', \pi(s_L'))]^T$

$\widetilde{R} = [r_1 \ \cdots \ r_L]^T$

$\widetilde{A} = \frac{1}{L}\widetilde{\Phi}^T\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)$

$\widetilde{b} = \frac{1}{L}\widetilde{\Phi}^T\widetilde{R}$

$w = \widetilde{A}^{-1}\widetilde{b}$

output: w

Approximating the action-value function by solving $\widetilde{A}w = \widetilde{b}$ is known as *Least Squares Temporal Difference* for $Q(s, a)$ (LSTDQ) [61]. LSTDQ is specified entirely in Algorithm 8.

For BRM, we need to solve Eq. (5.10). This is equivalent to solving the linear system of equation

$$Aw = b$$

with

$$A = (\Phi - \gamma P\Pi_\pi\Phi)^T (\Phi - \gamma P\Pi_\pi\Phi)$$

and

$$b = (\Phi - \gamma P\Pi_\pi\Phi)^T\mathcal{R}.$$

As before, using the definitions of Φ, P and Π_π (see Sec. 5.1.3), we can write[10] A as [61]

$$\begin{aligned}
A &= (\Phi - \gamma P\Pi_\pi\Phi)^T (\Phi - \gamma P\Pi_\pi\Phi) \\
&= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\left(\phi(s,a) - \gamma\sum_{s''\in\mathcal{S}}\mathcal{P}(s,a,s'')\phi(s'',\pi(s''))\right)\left(\phi(s,a) - \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}(s,a,s')\phi(s',\pi(s'))\right)^T \\
&= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}\sum_{s'\in\mathcal{S}}\mathcal{P}(s,a,s')\sum_{s''\in\mathcal{S}}\mathcal{P}(s,a,s'')\left[(\phi(s,a) - \gamma\phi(s'',\pi(s'')))\,(\phi(s,a) - \gamma\phi(s',\pi(s')))^T\right]
\end{aligned}$$

[10] Here we use the fact that for a matrix M, $M^\dagger = (M^TM)^{-1}M^T$.

and

$$
\begin{aligned}
b &= \left(\Phi - \gamma P \Pi_\pi \Phi\right)^T \mathcal{R} \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \left(\phi(s,a) - \gamma \sum_{s'\in\mathcal{S}} P(s,a,s')\phi(s',\pi(s')) \right) \sum_{s''\in\mathcal{S}} P(s,a,s'')R(s,a,s'') \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s') \sum_{s''\in\mathcal{S}} P(s,a,s'') \left(\phi(s,a) - \gamma\phi(s',\pi(s'))\right) R(s,a,s'').
\end{aligned}
$$

Note that in the expressions above there is a sum over s' and a sum over s''. This implies that to get an unbiased estimator of A and b an agent needs to collect *double samples* [61]. That is, samples of the form

$$
\begin{aligned}
D_1 &= \{(s_i, a_i, r_i', s_i')\} \quad i = 1, \ldots, L \\
D_2 &= \{(s_i, a_i, r_i'', s_i'')\} \quad i = 1, \ldots, L.
\end{aligned}
$$

Then the agent can estimate A as

$$
\widetilde{A} = \frac{1}{L} \sum_{i=1}^{L} \left(\phi(s_i,a_i) - \gamma\phi(s_i'',\pi(s_i''))\right) \left(\phi(s_i,a) - \gamma\phi(s_i',\pi(s_i'))\right)^T
$$

and b as

$$
\widetilde{b} = \frac{1}{L} \sum_{i=1}^{L} \left(\phi(s_i,a_i) - \gamma\phi(s_i'',\pi(s_i''))\right) r_i''.
$$

As before, for BRMQ it is convenient to write

$$
\widetilde{A} = \frac{1}{L} \left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T \left(\widetilde{\Phi} - \gamma\widetilde{\Phi}''\right)
$$

and

$$
\widetilde{b} = \frac{1}{L} \left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T \widetilde{R},
$$

Algorithm 9 BRMQ

input: $D_1 = \{(s_i, a_i, r_i, s_i')\}$ $D_2 = \{(s_i, a_i, r_i'', s_i'')\}$ $i = 1, \dots L$, $\phi(\cdot, \cdot)$, γ.

$\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$

$\widetilde{\Phi}' = [\phi(s_1', \pi(s_1')) \ \cdots \ \phi(s_L', \pi(s_L'))]^T$

$\widetilde{\Phi}'' = [\phi(s_1'', \pi(s_1'')) \ \cdots \ \phi(s_L'', \pi(s_L''))]^T$

$\widetilde{R} = [r_1 \ \cdots \ r_L]^T$

$\widetilde{A} = \frac{1}{L}\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T \left(\widetilde{\Phi} - \gamma\widetilde{\Phi}''\right)$

$\widetilde{b} = \frac{1}{L}\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T \widetilde{R}$

$w = \widetilde{A}^{-1}\widetilde{b}$

output: w

with $\widetilde{\Phi}$, $\widetilde{\Phi}'$, and $\widetilde{R}$ defined as before, and $\widetilde{\Phi}'' = [\phi(s_1'', \pi(s_1'')) \ \cdots \ \phi(s_L'', \pi(s_L''))]^T$.

BRMQ is specified entirely in Algorithm 9.

5.2.4 Least Squares Policy Iteration

The two methods described before can be used to find approximations of the state- and the action-value function of a given policy. These approaches can be used to find an optimal policy using Least Squares Policy Iteration (LSPI).

LSPI, proposed by Lagoudakis and Parr [61], is an iterative mechanism to find the optimal policy. The key idea is that the policy is represented implicitly by $\widehat{Q}$, or equivalently, by w (see Eq. (5.7)):

$$\begin{aligned}\widehat{\pi}(s; w) &= \operatorname*{argmax}_{a \in \mathcal{A}} \widehat{Q}(s, a) \\ &= \operatorname*{argmax}_{a \in \mathcal{A}} \phi(s, a)^T w.\end{aligned}$$

LSPI starts by setting all the entries of the vector w to zero. Inside an iteration loop, it uses the policy induced by the current w to compute an approximated action-value function, i.e., a new w' vector that approximates the action-value function induced by w; this new w' vector is used again to compute a new approximation of the action-value function.

Algorithm 10 LSPI

input: $D = \{(s_i, a_i, r_i, s_i') \mid i = 1, \ldots L\}$, $\phi(\cdot, \cdot)$, policy_evaluation$(\cdot)$, γ, ϵ.
  $w' = \mathbf{0}$
  repeat
    $w = w'$
    $w' =$ policy_evaluation$(D, \phi(\cdot, \cdot), \gamma, \widehat{\pi}(\cdot, w))$
  until $||w - w'|| < \epsilon$
output: w'

The iteration stops when the difference between two consecutive w vectors—or correspondingly, two consecutive policies induced by w—is small enough. LSPI is specified entirely in Algorithm 10.

LSPI comes with the following theoretical guarantee.

Theorem 5.2 (Lagoudakis and Parr)

Let $\pi_0, \pi_1, \ldots, \pi_m$ be the sequence of policies generated by LSPI and let $\widehat{Q}^{\pi_1}, \widehat{Q}^{\pi_2}, \ldots, \widehat{Q}^{\pi_m}$ be the corresponding approximate action-value functions as computed by LSTDQ. Let ϵ be a positive scalar that bounds the errors between the approximate and the true value functions over all iterations:

$$\left|\left|\widehat{Q}^{\pi_m} - Q^*\right|\right|_\infty \leq \epsilon, \quad m = 1, 2, \ldots.$$

Then, this sequence eventually produces policies whose performance is at most a constant multiple of ϵ away from the optimal performance:

$$\limsup_{m \to \infty} \left|\left|\widehat{Q}^{\pi_m} - Q^*\right|\right|_\infty \leq \frac{2\gamma\epsilon}{(1-\gamma)^2}.$$

5.2.5 Feature Vectors

To approximate the value function we need to define the feature vector $\phi(s)$. In our experiments we use RBFs to construct this vector. Among the different types of RBFs, we

use the Gaussian RBF [84, Sec. 3.7] defined as

$$f_{\eta,a}(x) = e^{-\eta(x-a)^2},$$

where η and a are the scale and location parameters, respectively. For a given state space $\mathcal{S} = \{s_1, \ldots, s_N\}$, number of features k, and scale η, the feature vector is given by

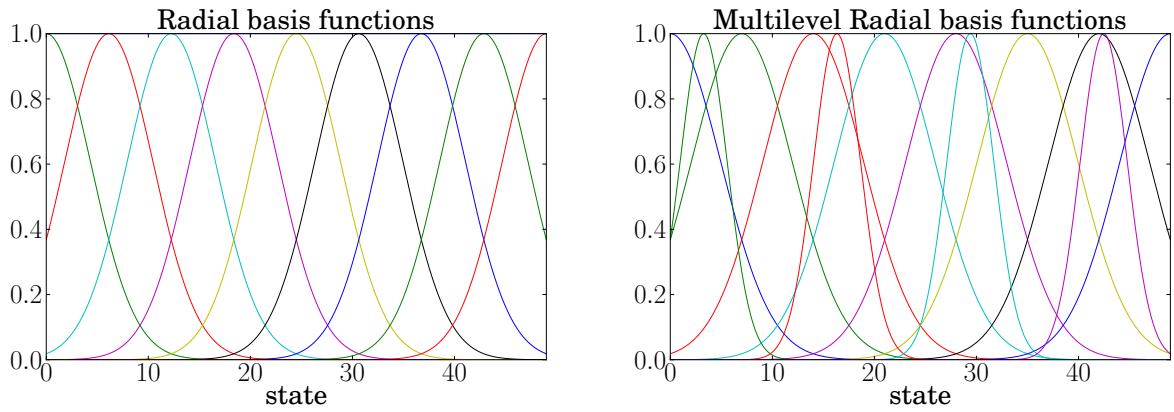
$$\phi(s) = [1 \ f_{\eta',a_1}(s) \ f_{\eta',a_2}(s) \ \cdots f_{\eta',a_{k'}}(s)]^T \quad s \in \mathcal{S},$$

with $k' = k - 1$, $a_i = 1 + (i-1)\frac{N-1}{k'-1}$, $i = 1, \ldots, k'$, and $\eta' = \frac{\eta(k'-1)}{N-1}$. Note that, to be able to approximate the continuous component of value functions efficiently, we augment the feature vector with a constant term. Figure 5.7(a) shows an example of these features for $N = 50$ and $k = 10$ (the constant term is omitted).

We also consider feature vectors defined by multilevel RBFs, where we concatenate RBFs at different scales. Let L be the number of levels, and let l_i, $i = 1, \ldots, L$, be the number of RBFs at level i. We set the number of levels for each scale such that l_i is approximately twice l_{i-1}. We find these values by solving

$$\begin{aligned} 2x_1 &= x_2 \\ 2x_2 &= x_3 \\ &\vdots \\ 2x_{L-1} &= x_L \\ x_1 + x_2 + \cdots + x_L &= k', \end{aligned}$$

with $k' = k - 1$, and then setting $l_i = \text{round}(x_i)$ for $i = 1, \ldots, L-1$, and $l_L = k' - \sum_{i=1}^{L-1} l_i$.



(a) (b)

Figure 5.7: RBF features. (a) RBF features for $N = 10$ and $k = 9$. (b) Multilevel RBF features for $N = 50$, $k = 250$ and $L = 5$. Only a fraction of the 250 features are shown.

The feature vector is given by

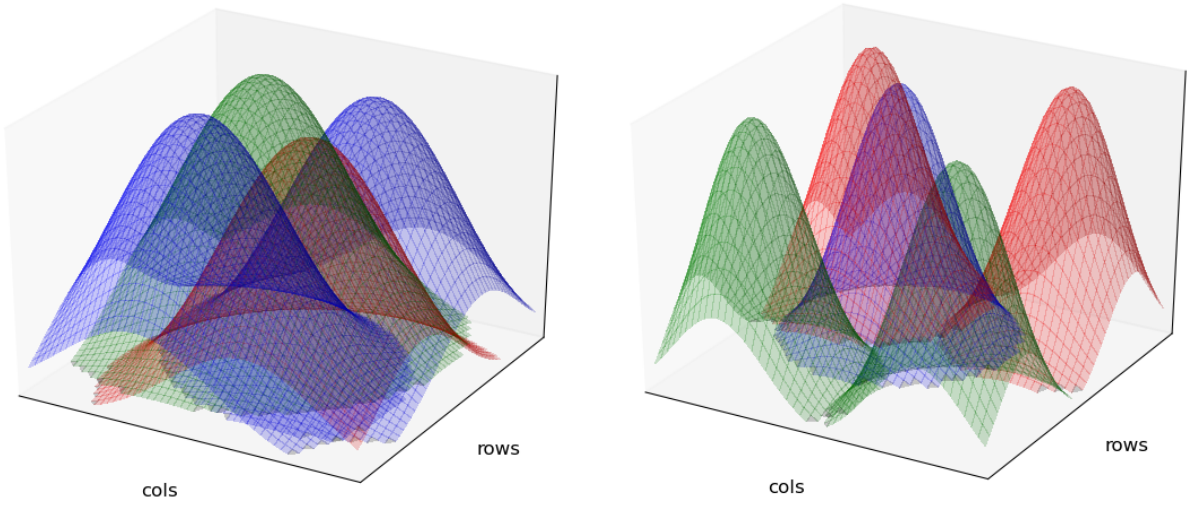
$$\phi(s) = [1 \underbrace{f_{\eta'_1 a^1_1}(s) \cdots f_{\eta'_1 a^1_{l_1}}(s)}_{\text{level 1 RBFs}} \cdots \underbrace{f_{\eta'_L a^L_1}(s) \cdots f_{\eta'_L a^L_{l_L}}(s)}_{\text{level } L \text{ RBFs}}]^T,$$

with $a^j_{i_j} = 1 + (i_j - 1)\frac{N-1}{l_j - 1}$, $i_j = 1, \ldots, l_j$, $j = 1, \ldots, L$, and $\eta'_j = \frac{\eta(l_j - 1)}{N-1}$. Note that as before, we augment the feature vector with a constant term. Figure 5.7(b) shows an example of these features for $N = 50$, $k = 250$ and $L = 5$ (only a fraction of the 250 features are shown).

If the state space can be considered as a two dimensional space—as is the case for grid-like environments—it is convenient to use two dimensional RBFs. We define the two dimensional Gaussian RBF as

$$f_{\eta,a}(x, y) = e^{-\eta_1(x-a_1)^2 - \eta_2(y-a_2)^2},$$

where $\eta = (\eta_1, \eta_2)$ and $a = (a_1, a_2)$ are the 2-dimensional scale and location parameters, respectively. For a given state space $\mathcal{S} = \{1, \ldots, N\}$, a mapping from the state representation $s \in \mathcal{S}$ to the two-dimensional representation $(x, y) \in \mathcal{G} = [0, x_{max}] \times [0, y_{max}]$, scale η, and a



(a) (b)

Figure 5.8: Two dimensional RBF features. (a) 2-by-2 grid. (b) 4-by-4 grid (only five of the sixteen features are shown).

d-by-d grid, feature vector is given by

$$\phi(s = x, y) = [1 \; f_{\eta',a_1}(x, y) \; \cdots \; f_{\eta',a'_k}(x, y)]^T,$$

with $a_i \in \{(\frac{x_{max}}{d+1}p, \frac{y_{max}}{d+1}q) \mid p, q = 1, \ldots, d\}$, $i = 1, \ldots, d^2$, and $\eta' = (\frac{\eta(d+1)}{x_{max}}, \frac{\eta(d+1)}{y_{max}})$. With this setup one ends with $k = k' + 1 = d^2 + 1$ features. It is also possible to concatenate features for grids with different numbers of points. Figure 5.8 shows an example of two dimensional RBFs for a 2-by-2 and a 4-by-4 grid.

5.3 Sparse Approximations

Sparse approximation algorithms can be used to approximate the state- and action-value function used in RL. Sparse approximations allow us to simplify the design of the feature vectors. For instance, LSTDQ requires the definition of k features, where k must be relatively small. Selecting the right features is critical to obtain good state- and action-value approximations and to learn an optimal policy. Sparse approximations, on the other hand, allow the use of a much larger number of features, which makes the design process much

simpler.

Several sparse approximation techniques for RL have been proposed in the literature. Kolter and Ng [60] proposed a method known as LARS-TD, an algorithm that combines LSTD with the LASSO estimator [97]. Loth et al. [64], on the other hand, applied the LASSO regression directly to the BRM error function. Finally, Painter-Wakefield and Parr [78] proposed to use greedy algorithms, applying OMP directly to the BRM, and also using a modified version of OMP. We now review the sparse approximation techniques used in this chapter.

5.3.1 LARS-TD

LARS-TD extends the LASSO/LARS estimators (see Sec. 2.3) to the RL problem. The key idea is to add an ℓ_1 regularization term to Eq. (5.12), used to compute the LSTD solution:

$$f(w) = \underset{u \in \mathbb{R}^k}{\operatorname{argmin}} \frac{1}{2} ||\Phi u - (\mathcal{R} + \gamma P_\pi \Phi w)||_2^2 + \beta ||u||_1 , \tag{5.14}$$

where β is a regularization parameter.

The optimality conditions of the optimization problem in Eq. (5.14) are [60]

$$\begin{aligned} -\beta \leq (\Phi^T((\mathcal{R} + \gamma \Phi' w) - \Phi u))_i \leq \beta \\ (\Phi^T((\mathcal{R} + \gamma \Phi' w) - \Phi u))_i = \beta \Rightarrow u_i \geq 0 \\ (\Phi^T((\mathcal{R} + \gamma \Phi' w) - \Phi u))_i = -\beta \Rightarrow u_i \leq 0 \\ -\beta < (\Phi^T((\mathcal{R} + \gamma \Phi' w) - \Phi u))_i < \beta \Rightarrow u_i = 0, \end{aligned} \tag{5.15}$$

for $i = 1, \ldots, k$. Note that these are the conditions on u for a given w. For w to be a fixed

point of Eq. (5.14) we need that the conditions in Eq. (5.15) hold for $u = w$, i.e., that

$$\begin{aligned} -\beta \leq (\Phi^T\mathcal{R} - \Phi^T(\Phi - \gamma\Phi')w)_i \leq \beta \\ (\Phi^T\mathcal{R} - \Phi^T(\Phi - \gamma\Phi')w)_i = \beta \Rightarrow w_i \geq 0 \\ (\Phi^T\mathcal{R} - \Phi^T(\Phi - \gamma\Phi')w)_i = -\beta \Rightarrow w_i \leq 0 \\ -\beta < (\Phi^T\mathcal{R} - \Phi^T(\Phi - \gamma\Phi')w)_i < \beta \Rightarrow w_i = 0, \end{aligned} \tag{5.16}$$

for $i = 1, \ldots, k$. Note that this is not the same as solving Eq. (5.14) with the additional fixed point constraint. See [60] for a detailed discussion of this issue.

The derivation of the LARS-TD algorithm follows from the optimality conditions defined in Eq. (5.15) and from following the same steps used to derive the LARS and Homotopy method (see Sec. 2.3). LARS-TD is described entirely by Algorithm 11.[11]

5.3.2 OMPBRM and OMP-TDQ

Unlike LSTD, BRM computes w just by solving the optimization problem given by Eq. (5.9) directly, not by finding a fixed point of it. For this reason, it is easy to enforce a sparse solution by invoking OMP (see Sec. 2.4) with $A = (\Phi - \gamma P_\pi \Phi)$ and $b = \mathcal{R}$. OMPBRMQ is specified entirely in Algorithm 12. The only caveat is that, as before, to estimate A and b from the samples gathered by the agent, we need to use doubled-samples. Note, however, that $\widetilde{A}$ is a square matrix, so it cannot be used to compute the correlation vector h. For this reason, h is computed using only single samples as[12] $h = |(\widetilde{\Phi} - \gamma\widetilde{\Phi}')^T r|$.

Painter-Wakefield and Parr [78, 79] also proposed a greedy algorithm that uses the LSFP (see Eq. (5.13)) instead of the BRM approximation. The method, called OMP-TDQ, is specified entirely in Algorithm 13. As in OMP, at each iteration it adds to the list of selected features $\mathcal{I}$ the one most correlated with the residual—in this case the residual is given by

[11] We show the version of the algorithm used to compute the action-value function.
[12] This fact was checked with the authors of [78] via personal communication.

Algorithm 11 LARS-TDQ

input: $\{s_i, a_i, r_i, s'_i\}$, $i = 1 \dots L$, $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$, $\pi : \mathcal{S} \to \mathcal{A}$, γ, β
$\tilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$, $\tilde{\Phi}' = [\phi(s'_1, \pi(s'_1)) \ \cdots \ \phi(s'_L, \pi(s'_L))]^T$, $R = [r_1 \ \cdots \ r_L]^T$
$\tilde{A} = \tilde{\Phi}^T(\tilde{\Phi} - \gamma\tilde{\Phi}')$
$w = 0$
$c = \tilde{\Phi}^T R$
$\{\bar{\beta}, i\} = \max_j\{|c_j|\}$
$\mathcal{I} = \{i\}$
while $\bar{\beta} > \beta$ **do**
$\Delta w_{\mathcal{I}} = \tilde{A}^{-1}_{\mathcal{I},\mathcal{I}} \operatorname{sign}(c_{\mathcal{I}})$
▷ Variables subscripted by $\mathcal{I}$ indicate submatrices and subvectors
$d = \tilde{\Phi}^T(\tilde{\Phi} - \gamma\tilde{\Phi}')\Delta w_{\mathcal{I}}$
$\{\alpha_1, i_1\} = \min^+_{j \notin \mathcal{I}} \left\{ \frac{c_j - \bar{\beta}}{d_j - 1}, \frac{c_j + \bar{\beta}}{d_j + 1} \right\}$
$\{\alpha_2, i_2\} = \min^+_{j \in \mathcal{I}} \left\{ -\frac{w_j}{\Delta w_j} \right\}$
$\alpha = \min\{\alpha_1, \alpha_2, \bar{\beta} - \beta\}$
$w_{\mathcal{I}} = w_{\mathcal{I}} + \alpha\Delta w_{\mathcal{I}}$
$\bar{\beta} = \bar{\beta} - \alpha$
$c = c - \alpha d$
if $\alpha_1 < \alpha_2$ **then**
$\mathcal{I} = \mathcal{I} \cup \{i_1\}$
else
$\mathcal{I} = \mathcal{I} \backslash \{i_1\}$
end if
end while
output: w

$(R + \gamma\widetilde{\Phi}'w - \widetilde{\Phi}w)$. It then uses the LSFP formula to compute the current approximation. It stops when the stopping criterion, typically stated in terms of the cardinality of $\mathcal{I}$, is met.

Note that calling this method OMP-TDQ is a misnomer. In OMP the residue is orthogonal to the currently selected features—remember that the "O" in OMP is for orthogonal. This fact is fundamental in the analysis of the algorithm [30, 43, 98]. On the other hand, in OMP-TDQ, since w is computed using the LSFP approximation, the residue is not orthogonal to the selected features. Thus, it is not possible to carry on most of the theoretical guarantees of OMP. To be consistent with the literature, we keep the name of the algorithm.

Algorithm 12 OMPBRM-Q

input: $D_1 = \{(s_i, a_i, r_i, s_i')\}$ $D_2 = \{(s_i, a_i, r_i'', s_i'')\}$ $i = 1, \dots L$, $\phi(\cdot,\cdot)$, γ, stopping criterion
  $\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$, $\widetilde{\Phi}' = [\phi(s_1', \pi(s_1')) \ \cdots \ \phi(s_L', \pi(s_L'))]^T$,
  $\widetilde{\Phi}'' = [\phi(s_1'', \pi(s_1'')) \ \cdots \ \phi(s_L'', \pi(s_L''))]^T$, $\widetilde{R} = [r_1 \ \cdots \ r_L]^T$
  $\widetilde{A} = \frac{1}{L}\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}''\right)$, $\widetilde{b} = \frac{1}{L}\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T\widetilde{R}$
  $r = \widetilde{b}$, $\mathcal{I} = \emptyset$
  while not converged **do**
    $h = |(\widetilde{\Phi} - \gamma\widetilde{\Phi}')^T r|$
    $\mathcal{I} = \mathcal{I} \cup \operatorname{argmax}_{j \notin \mathcal{I}} |h(j)|$
    $w = \operatorname{argmin}_{z:\ \operatorname{supp}(z) \subseteq \mathcal{I}} \|\widetilde{b} - \widetilde{A}z\|_2$
    $r = \widetilde{b} - \widetilde{A}w$
  end while
output: w

Algorithm 13 OMP-TDQ

input: $D = \{(s_i, a_i, r_i, s_i')\}, i = 1, \dots L$, $\phi(\cdot,\cdot)$, γ, stopping criterion
  $\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$, $\tilde{\Phi}' = [\phi(s_1', \pi(s_1')) \ \cdots \ \phi(s_L', \pi(s_L'))]^T$
  $\tilde{A} = \tilde{\Phi}^T(\tilde{\Phi} - \gamma\tilde{\Phi}')$, $\widetilde{R} = [r_1 \ \cdots \ r_L]^T$
  $w = 0$, $\mathcal{I} = \emptyset$
  while not converged **do**
    $c = |\widetilde{\Phi}^T(R + \gamma\widetilde{\Phi}'w - \widetilde{\Phi}w)|$
    $\mathcal{I} = \mathcal{I} \cup \operatorname{argmax}_{i \notin \mathcal{I}} c_i$
    $w_{\mathcal{I}} = A_{\mathcal{I},\mathcal{I}}^{-1}\Phi_{\mathcal{I}}^T R$ ▷ Variables subscripted by $\mathcal{I}$ indicate submatrices and subvectors
  end while
output: w

5.3.3 The Case for Group Sparsity

In addition to admitting a sparse representation, we claim that the action-value function possesses additional structure, and that this additional structure can be exploited to improve the existing sparse RL algorithms.

To approximate the action-value function $Q(s, a)$ as $\widehat{Q} = \Phi w$, one typically starts with a feature vector $\phi(s)$, as the ones described in Sec. 5.2.5. Using this vector, the feature vector for a state-action pair (s, a_i), with $s \in \mathcal{S}$, $a_i \in \mathcal{A}$, and $i \in \{1, \dots, |\mathcal{A}|\}$, is built by zero-padding the vector $\phi(s)$ such that $\phi(s, a_i)$ is a vector of length $k|\mathcal{A}|$, entries $(i-1)k+1$ to ik are set to $\phi(s)$, and the remaining entries are set to zero. In other words, $\phi(s, a_i)$ is

given by

$$\phi(s, a_i) = [— \ 0 \ — \ \cdots \ — \ \phi(s) \ — \ \cdots \ — \ 0 \ —]^T. \tag{5.17}$$

Note that although this is standard practice, and that it is the approach taken in this work, nothing prevents one from constructing the feature vector $\phi(s, a_i)$ differently (see [77] for an example of an alternative formulation of the feature vector). Given the structure of $\phi(s, a)$ defined by Eq. (5.17), it follows that matrix Φ has the following structure:

$$\begin{aligned}
\Phi &= \begin{bmatrix}
— \ \phi(s_1) \ — & — \ 0 \ — & \cdots & — \ 0 \ — \\
\vdots & \vdots & \vdots & \vdots \\
— \ \phi(s_{|\mathcal{S}|}) \ — & — \ 0 \ — & \cdots & — \ 0 \ — \\
— \ 0 \ — & — \ \phi(s_1) \ — & \cdots & — \ 0 \ — \\
\vdots & \vdots & \vdots & \vdots \\
— \ 0 \ — & — \ \phi(s_{|\mathcal{S}|}) \ — & \cdots & — \ 0 \ — \\
— \ 0 \ — & 0 & \cdots & — \ \phi(s_1) \ — \\
\vdots & \vdots & \vdots & \vdots \\
— \ 0 \ — & — \ 0 \ — & \cdots & — \ \phi(s_{|\mathcal{S}|}) \ —
\end{bmatrix} \\
&= \begin{bmatrix}
\Phi_{\mathcal{S}} & \mathbf{0} & \cdots & \mathbf{0} \\
\mathbf{0} & \Phi_{\mathcal{S}} & \cdots & \mathbf{0} \\
\vdots & \vdots & \vdots & \vdots \\
\mathbf{0} & \mathbf{0} & \cdots & \Phi_{\mathcal{S}}
\end{bmatrix},
\end{aligned} \tag{5.18}$$

where $\Phi_{\mathcal{S}}$ is an $|\mathcal{S}| \times k$ matrix, with rows set to $\phi(s_i)$, $i = 1, \ldots, \mathcal{S}$, and $\mathbf{0}$ a $|\mathcal{S}| \times k$ matrix with all its entries set to zero.

Firstly, this means that we can partition vector w as $w = [w_{a_1} \ \cdots \ w_{a_{|\mathcal{A}|}}]^T$ where each vector w_{a_i} has length k; and that we can write $Q(s, a_i) = \Phi_{\mathcal{S}} w_{a_i}$ for $s \in \mathcal{S}$, $a_i \in \mathcal{A}$, $i = 1, \ldots, |\mathcal{A}|$. Secondly, the general structure of the action-value function $Q(s, a)$ is such that all the functions $Q(s, a_i) = \Phi_{\mathcal{S}} w_{a_i}$ for $a_i \in \mathcal{A}$, although different, share in general

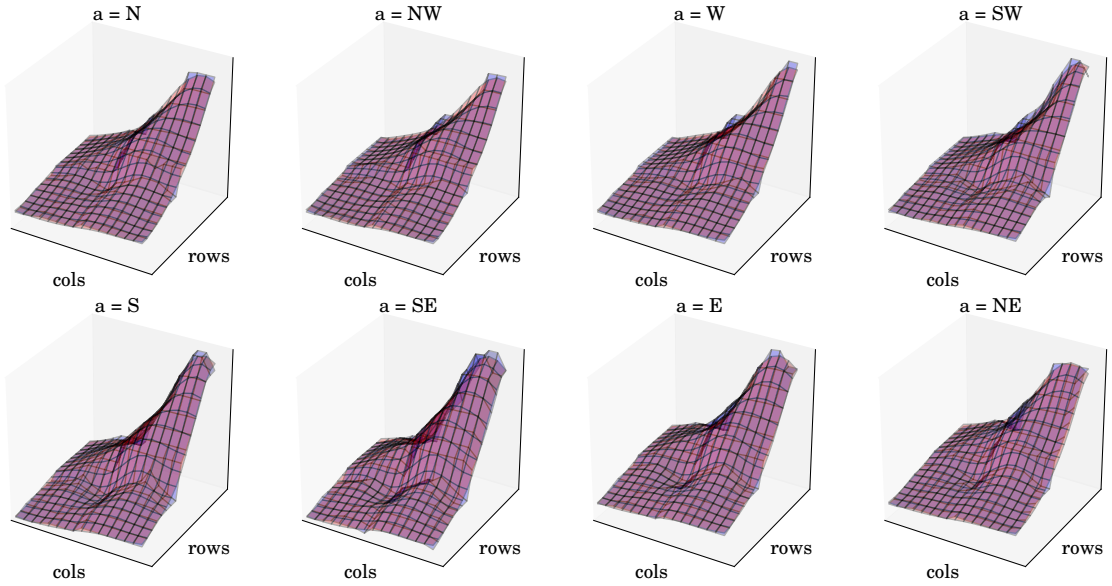


Figure 5.9: Action-value function for a four-rooms environment with king moves and $|\mathcal{S}| = 256$ states. The panels show the action-value function $Q(s, a_i)$ for $a_i \in \{\mathrm{N}, \mathrm{NW}, \ldots, \mathrm{NE}\}$. In blue the action-value function and in red the action-value function approximated by BOMP using 240 nonzero features. The approximation error is $\left|\left|Q - \widehat{Q}_{BOMP}\right|\right|_2 = 20.79$. In comparison, the approximation error using OMP with the same number of nonzero features is $\left|\left|Q - \widehat{Q}_{OMP}\right|\right|_2 = 24.00$.

the same structure. This behavior follows from the nested structure of value functions: $Q(s, a_i)$ is in general not much different than $Q(s, a_j)$, for $i \neq j$. To see this, compare, for instance, the action-value function of the chain environment (see Sec. 5.1.1). Fig. 5.11 shows $Q(s, \texttt{Left})$ and $Q(s, \texttt{Right})$ in panels (a) and (b), respectively. We observe that $Q(s, \texttt{Left})$ and $Q(s, \texttt{Right})$ are similar. We observe a similar situation for a four-rooms environment. Figure 5.9 shows the corresponding action-value function indexed by the different actions. Again we observe that all the $Q(s, a_i)$ are similar.

The similarity between $Q(s, a_i)$ and $Q(s, a_j)$ implies that w_{a_i} and w_{a_j} are also similar. Moreover, if an entry at a given location of w_{a_i} is nonzero, it is reasonable to expect that the entry of w_{a_j} at the same location is also nonzero. In other words, it is reasonable to expect that w_{a_i} and w_{a_j} have the same support.

To illustrate this phenomenon, let us consider the action-value function of a four-rooms

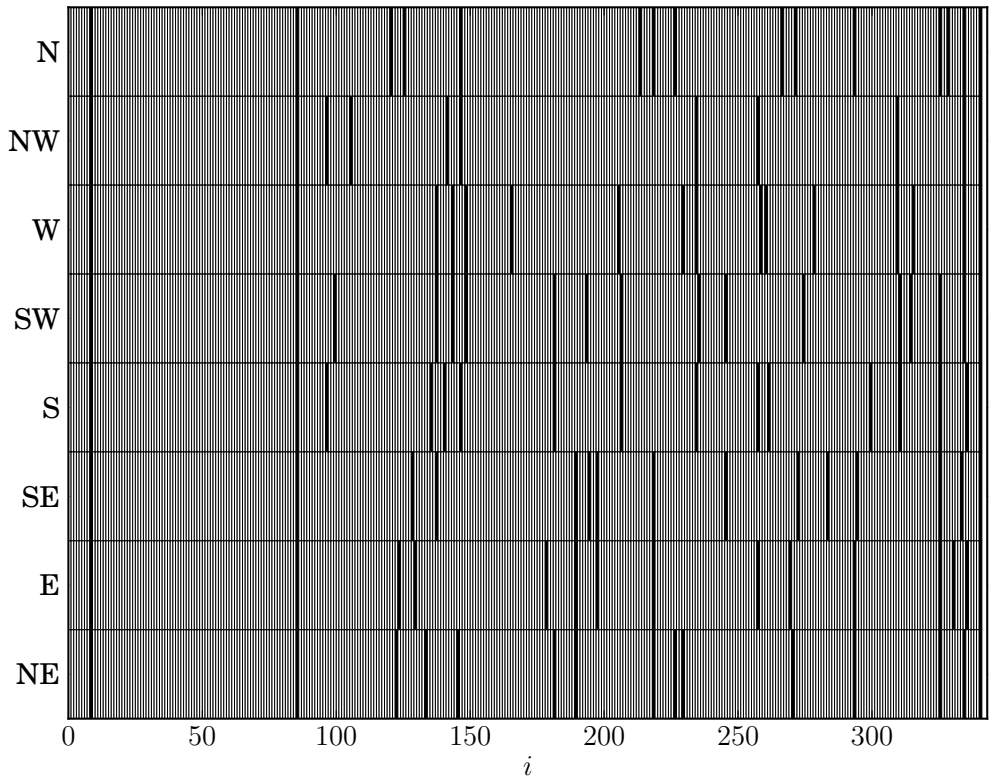


Figure 5.10: Support of the OMP approximation of the action-value function of a four-rooms environment with king moves and $|\mathcal{S}| = 256$ states. Each slot represents an entry of w_{OMP}. Nonzero entries are black, and zero entries are white. The vector is split in eight parts and stacked. Although OMP does not enforce any additional structure beyond sparsity, many nonzero coefficients are located in the same group.

environment with king moves and $|\mathcal{S}| = 256$ states (see Sec. 5.1.2). Using the known model of the environment we compute the action-value function $Q(s, a)$. Then we approximate the action-value function using OMP and BOMP. We concatenate two-dimensional RBFs (see Sec 5.2.5) over 2-by-2, 4-by-4, 8-by-8, and 16-by-16 grids to create the feature vectors, for a total of 2728 features. In summary, we are computing a sparse representation of Q of the form $\widehat{Q} = \Phi w$, with Q and $\widehat{Q}$ a vector of length 256, Φ a 256×2728 matrix with the structure shown by Eq. (5.18), and w a vector of length 2728.

We compute the approximation using OMP and BOMP, denoting them $\widehat{Q}_{OMP} = \Phi w_{OMP}$ and $\widehat{Q}_{BOMP} = \Phi w_{BOMP}$, respectively. For both cases we compute an approxi-

mation with sparsity set to 120. Figure 5.10 shows the support of w_{OMP}. In the figure, each slot represents an entry of this vector. Nonzero entries are black, and zero entries are white. The vector is split in eight parts and stacked. Although OMP does not enforce any additional structure beyond sparsity, many nonzero coefficients are located in the same group. This behavior confirms our claim that the action-value approximation has a structure beyond sparsity. Moreover, the BOMP approximation[13], shown in Fig. 5.9, is able to approximate the action-value function with a smaller error— $\left|\left|Q - \widehat{Q}_{BOMP}\right|\right|_2 = 20.79$ versus $\left|\left|Q - \widehat{Q}_{OMP}\right|\right|_2 = 24.00$— using the same number of features. Note also that since BOMP adds 8 (the number of available actions) indices to the support at each iteration, rather than 1 at time, it is able to compute the approximation almost eight times faster.

5.3.4 Group Sparse Methods for RL

In the previous section we motivated the use of group sparsity to approximate the action-value function. However, the example above approximated the action-value function computed using the model of the environment. While useful to motivate the discussion, this is not the problem we are interested in, since in general we do not have access to this model. In this section we present several methods that exploit the group sparsity in an RL setup, i.e., in cases where an agent needs to learn only through interactions with the environment.

Recall that OMPBRM-Q consisted of invoking OMP with with $A = (\Phi - \gamma P_\pi \Phi)$ and $b = \mathcal{R}$. By the same token, we can enforce the group sparsity condition by invoking BOMP with the same arguments. This method, dubbed BOMPBRM-Q, is specified entirely in Algorithm 14. The main difference is that now the correlation vector c is computed as $c_j = ||X_j r||$, $j = 1, \ldots, k$, where X_j denotes the submatrix corresponding to columns $j + (i-1)k$, $i = 1, \ldots, |\mathcal{A}|$, of X.

[13]The original formulation of BOMP considers the case where nonzero elements appear in *blocks*. Given the structure of Φ, in this case, rather than in blocks, the nonzero elements appear at known non-contiguous locations. From a theoretical and practical point of view, this difference is nil. Although this means that a better name for the algorithm would be "Group OMP," we use BOMP since is the name used in the

Algorithm 14 BOMPBRM-Q

input: $D_1 = \{(s_i, a_i, r_i, s'_i)\}$ $D_2 = \{(s_i, a_i, r''_i, s''_i)\}$ $i = 1, \dots L$, $\phi(\cdot,\cdot)$, $|\mathcal{A}|$, γ, k, stopping criterion

$\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$, $\widetilde{\Phi}' = [\phi(s'_1, \pi(s'_1)) \ \cdots \ \phi(s'_L, \pi(s'_L))]^T$,

$\widetilde{\Phi}'' = [\phi(s''_1, \pi(s''_1)) \ \cdots \ \phi(s''_L, \pi(s''_L))]^T$, $\widetilde{R} = [r_1 \ \cdots \ r_L]^T$

$\widetilde{A} = \frac{1}{L}\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T \left(\widetilde{\Phi} - \gamma\widetilde{\Phi}''\right)$, $\widetilde{b} = \frac{1}{L}\left(\widetilde{\Phi} - \gamma\widetilde{\Phi}'\right)^T \widetilde{R}$

$X = \widetilde{\Phi} - \gamma\widetilde{\Phi}'$

$r = \widetilde{b}$, $\mathcal{I} = \emptyset$

while not converged **do**

$c_j = ||X_j r||$, $j = 1, \dots, k$

$\mathcal{I} = \mathcal{I} \cup \text{argmax}_{j \notin \mathcal{I}} |c(j)|$

$w = \text{argmin}_{z:\ \text{supp}(z) \subseteq \mathcal{I}} \|\widetilde{b} - \widetilde{A}z\|_2$

$r = \widetilde{b} - \widetilde{A}w$

end while

output: w

Algorithm 15 BOMP-TDQ

input: $D = \{(s_i, a_i, r_i, s'_i)\}, i = 1, \dots L$, $\phi(\cdot,\cdot)$, γ, stopping criterion

$\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$, $\widetilde{\Phi}' = [\phi(s'_1, \pi(s'_1)) \ \cdots \ \phi(s'_L, \pi(s'_L))]^T$

$\tilde{A} = \tilde{\Phi}^T(\tilde{\Phi} - \gamma\tilde{\Phi}')$, $\widetilde{R} = [r_1 \ \cdots \ r_L]^T$

$w = 0$, $\mathcal{I} = \emptyset$

while not converged **do**

$h = \widetilde{\Phi}^T(R + \gamma\widetilde{\Phi}'w - \widetilde{\Phi}w)$

$c_j = ||h_j||$, $j = 1, \dots, k$

$\mathcal{I} = \mathcal{I} \cup \text{argmax}_{i \notin \mathcal{I}} c_i$

$w_{\mathcal{I}} = A^{-1}_{\mathcal{I},\mathcal{I}} \Phi^T_{\mathcal{I}} R$ ▷ Variables subscripted by $\mathcal{I}$ indicate submatrices and subvectors

end while

output: w

We also add the group sparsity condition to OMP-TDQ. The method, dubbed, BOMP-TDQ, is specified entirely in Algorithm 15. The main difference is that, similarly to BOMPBRM-Q, the correlation vector is computed considering the group sparsity constraint by setting $c_j = ||h_j||$, where h_j denotes the entries $j + (i-1)k$, $i = 1, \dots, |\mathcal{A}|$, of h.

Finally, we add the group sparsity condition to LARS-TD. The method, dubbed GLARS-TDQ, is specified entirely in Algorithm 16. Starting from the standard GLARS algorithm (see Algorithm 3) GLARS-TDQ follows from using $R - (\widetilde{\Phi} - \gamma\widetilde{\Phi}')w$ as a proxy for the residue,

literature.

Algorithm 16 GLARS-TDQ

input: $\{s_i, a_i, r_i, s'_i\}$, $i = 1 \dots L$, $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$, $\pi : \mathcal{S} \to \mathcal{A}$, γ, β

$\widetilde{\Phi} = [\phi(s_1, a_1) \ \cdots \ \phi(s_L, a_L)]^T$, $\widetilde{\Phi}' = [\phi(s'_1, \pi(s'_1)) \ \cdots \ \phi(s'_L, \pi(s'_L))]^T$, $R = [r_1 \ \cdots \ r_L]^T$

$w = 0$

$I = \left\{ \text{argmax}_j \left(\left| \left| \widetilde{\Phi}_j^T R \right| \right| \right) \right\}$

$k = 1$

while not converged **do**

$r = R - (\widetilde{\Phi}_j - \gamma \widetilde{\Phi}')w_j$

$d = \mathbf{0}$

$d(I) = \left(\widetilde{\Phi}_I^T \left(\widetilde{\Phi}_I - \gamma \widetilde{\Phi}'_I \right) \right)^{-1} \widetilde{\Phi}_I^T r_k$

$j' =$ choose any from I

for $j \in I^c$ **do**

$a = \left| \left| \widetilde{\Phi}_j^T \left(\widetilde{\Phi} - \gamma \widetilde{\Phi}' \right) d \right| \right|^2 - \left| \left| \widetilde{\Phi}_{j'}^T \left(\widetilde{\Phi} - \gamma \widetilde{\Phi}' \right) d \right| \right|^2$

$b = 2r^T \left(\widetilde{\Phi}_j \widetilde{\Phi}_j^T - \widetilde{\Phi}_{j'} \widetilde{\Phi}_{j'}^T \right) \left(\widetilde{\Phi} - \gamma \widetilde{\Phi}' \right) d$

$c = \left| \left| \widetilde{\Phi}_j^T r \right| \right|^2 - \left| \left| \widetilde{\Phi}_{j'}^T r \right| \right|^2$

$\gamma_j = \text{solve}(a\gamma^2 - b\gamma + c)$

end for

$\gamma_{j^*}, i^* = \min_j^+ \{\gamma_j\}$

$I = I \cup \{i^*\}$

$w = w + \gamma_{j^*} d$

$k = k + 1$

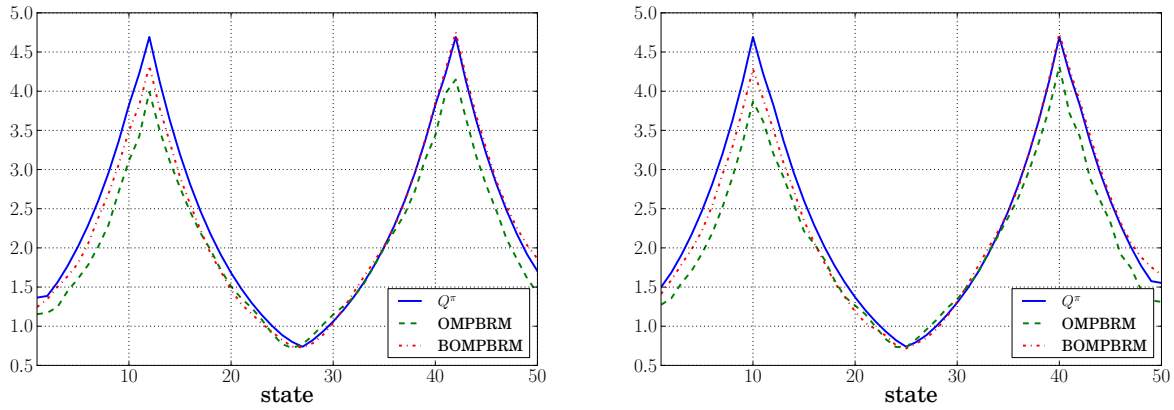
end while

output: x_j

$\widetilde{\Phi}^T$ as A^T, and $(\widetilde{\Phi} - \gamma \widetilde{\Phi}')$ as A. From an implementation point of view, it is important to note that the description of the innermost loop in Algorithm 16 follows from the mathematical derivation of the algorithms, and that it is not an efficient way to compute the terms a, b, and c. For instance, all the computations that depend on j' can be taken outside the loop. In addition, since the term $(\widetilde{\Phi}_j \widetilde{\Phi}_j^T - \widetilde{\Phi}_{j'} \widetilde{\Phi}_{j'}^T)$ used to compute b only depends on j and j', is advantageous to cache this result using a *memoization* technique [26, Sec. 15.3].

5.4 Experimental Results

This section describes some empirical results of the methods described previously. The experiments are based on the chain environment with $N = 50$ states (see Sec. 5.1.1), and



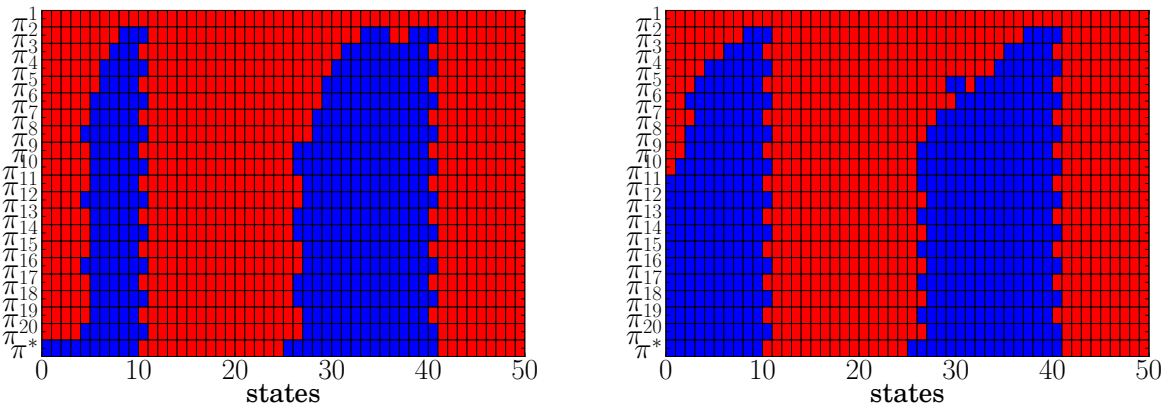
(a) $a =$ '`Left`' (b) $a =$ '`Right`'

Figure 5.11: Action-value function $\widehat{Q}(s, a)$ for the chain environment with $N = 50$ states, computed using OMP-BRM (in green) and BOMP-BRM (in red). Also shown is the exact action-value function $Q(s, a)$ (in blue).

in the four-rooms environments (see Sec. 5.1.2) with different configurations. For the chain environment we use multi-level RBF features with a total of 200 features and 5 levels. For the four-room environment we compute the feature vector by concatenating two-dimensional RBFs (see Sec 5.2.5) over 2-by-2, 4-by-4, 8-by-8, and 16-by-16 grids, for a total of 2728 features. In all the experiments the agent collects L samples by starting at a random state, executing a random policy for five steps, and repeating this $L/5$ times.

In the first experiment we find the approximation of the action-value function for the chain environment, using OMP-BRM, denoted by $\widehat{Q}_{OMPBRM}$, and BOMP-BRM, denoted by $\widehat{Q}_{BOMPBRM}$. The policy is fixed to the optimal policy (computed by hand). Figure 5.11 shows the results, together with the exact action-value function $Q(s, a)$ (left panel shows $Q(s, L)$ and right panel shows $Q(s, R)$). Note that for all the states $\widehat{Q}_{BOMPBRM}$ is closer to $Q(s, a)$ than $\widehat{Q}_{OMPBRM}$. We quantify this difference by comparing the error[14] between both

[14]To compute the errors the action-value functions are transformed to a vector by concatenating the function horizontally across each action, i.e., $Q(s, a)$ is transformed to $[Q(s, L)\ Q(s, R)]$.



(a) OMP-BRM

(b) BOMP-BRM

Figure 5.12: Optimal policy computed using LSPI and (a) OMP-BRM, (b) BOMP-BRM. '`Left`' and '`Right`' actions are represented by cells in red and blue, respectively. For comparison, an optimal policy π^* is shown in the last row. The approximation errors are $\left|\left|Q - \widehat{Q}_{OMPBRM}\right|\right|_2 = 3.6$ and $\left|\left|Q - \widehat{Q}_{BOMPBRM}\right|\right|_2 = 1.7$.

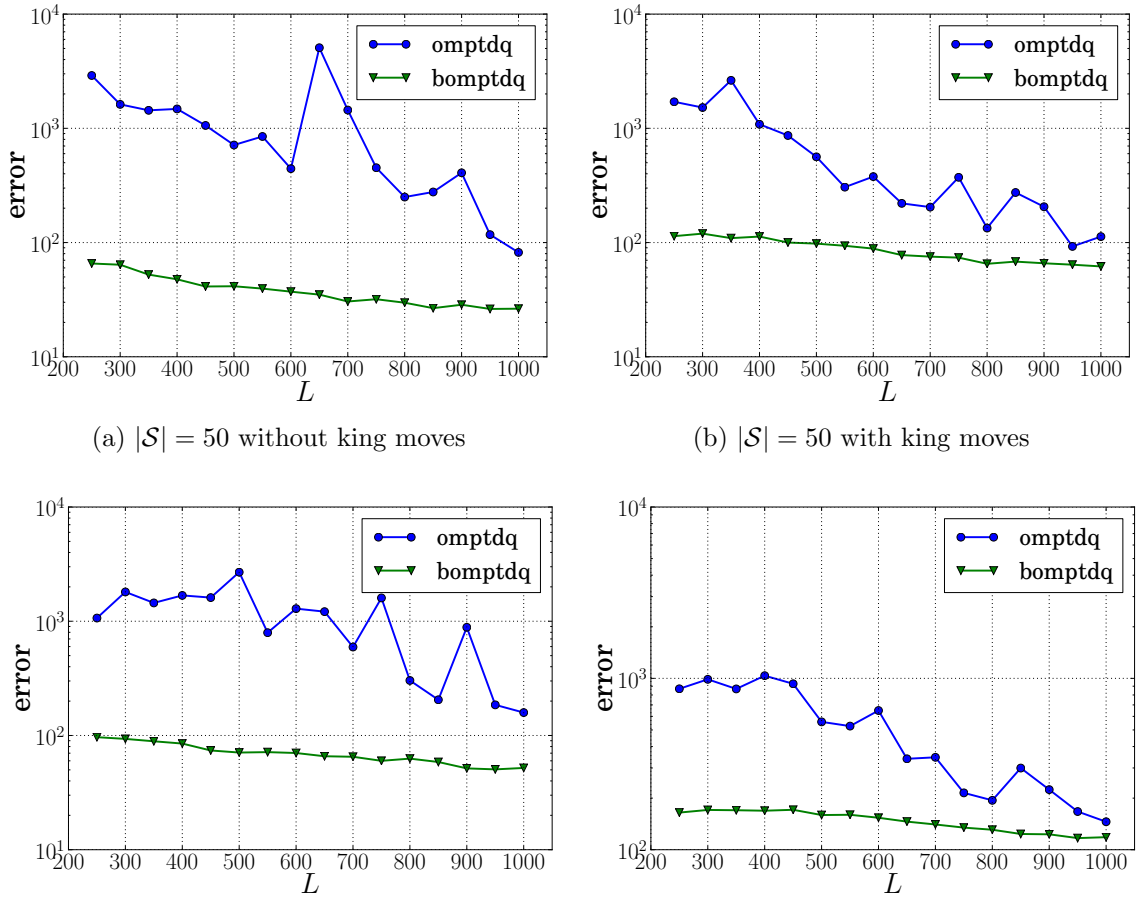
approximations:

$$\left|\left|Q - \widehat{Q}_{OMPBRM}\right|\right|_2 = 3.6,$$
$$\left|\left|Q - \widehat{Q}_{BOMPBRM}\right|\right|_2 = 1.7.$$

In the second experiment we use LSPI to find an optimal policy, using both OMP-BRM and BOMP-BRM. As observed in Fig. 5.12, for both methods the policy oscillates between two options.[15] However, while LSPI with OMP-BRM oscillates between two non-optimal policies, LSPI with BOMP-BRM oscillates between two optimal policies (recall that optimal policies are not unique).

In the next experiment we use the four-rooms environment to quantify the effect of the number of samples on the approximation error. We consider four different environments, with numbers of states set to $|\mathcal{S}| = 50$ and $|\mathcal{S}| = 100$ and with and without king moves. A single trial consists of approximating the action-value function using L samples with OMP-TDQ and BOMP-TDQ, both methods using 240 nonzero features. For each value

[15] This behavior is predicted by the LSPI theory [61].



(a) $|\mathcal{S}| = 50$ without king moves

(b) $|\mathcal{S}| = 50$ with king moves

(c) $|\mathcal{S}| = 100$ without king moves

(d) $|\mathcal{S}| = 100$ with king moves

Figure 5.13: Approximation error of action-value function using OMP-TDQ and BOMP-TDQ for the four-rooms environment. Both methods use 240 nonzero features. Average error computed over 200 trials. Note that the scale is logarithmic.

of $L \in \{250, 300, \ldots, 1000\}$ the experiment is repeated 200 times and average errors are reported. Figure 5.13 shows the results (note that the ordinate is in logarithmic scale). We observe how BOMP-TDQ is significantly better than OMP-TDQ. The main advantage of BOMP-TDQ is that it is much more robust. Our conjecture is that by selecting a group of features per each iteration, BOMP-TDQ is much better at generalizing from a small number of samples.

BOMP-TDQ is not only able to compute better approximations, but it also faster. Table 5.1 shows the time[16] required to approximate the action-value function using OMP-

[16] Algorithms executed in a machine with an Intel Core i5 processor operating at 2.6 GHz. The execution

Table 5.1: Time required to approximate the action-value function using OMP-TDQ and BOMP-TDQ in a four-rooms environment with $|\mathcal{S}| = 100$ states and king moves. The number of nonzero features is set to 240 for both methods.

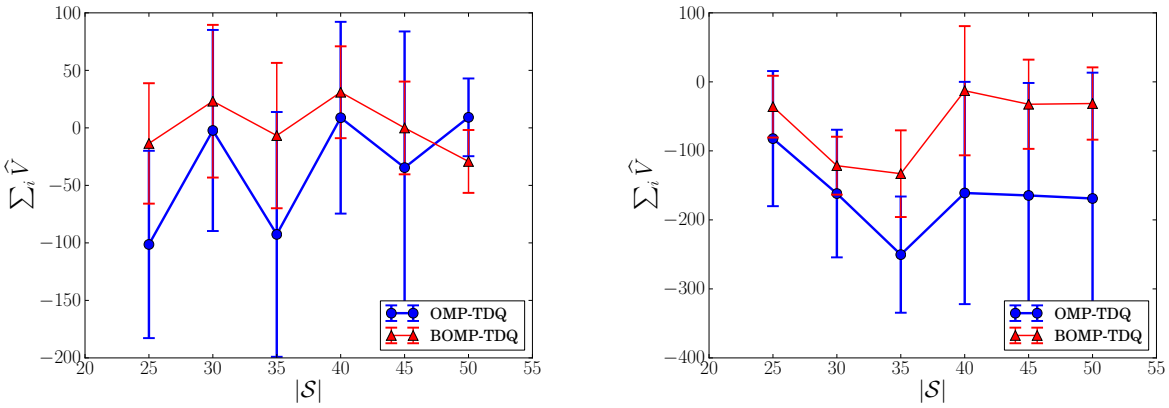
L	OMP-TDQ	BOMP-TDQ	Ratio
250	2.9	0.63	4.6
500	4.7	0.96	4.9
1000	8.1	1.52	5.3
2000	18.8	3.12	6.0

TDQ and BOMP-TDQ in a four-rooms environment with $|\mathcal{S}| = 100$ states and king moves, and the number of nonzero features is set to 240 for both methods for different values of the number of samples L. We observe that BOMP-TDQ is faster by a factor of 6 for $L = 2000$. This is not surprising, since per each BOMP-TDQ iteration, OMP-TDQ executes $|\mathcal{A}|$—in this example $|\mathcal{A}|$ is 8. Each BOMP-TDQ iteration is more expensive than an OMP-TDQ iteration, since it involves a norm computation. This explains why the ratio is not exactly $|\mathcal{A}|$.

Next, we use OMP-TDQ and BOMP-TDQ combined with LSPI (see Sec 5.2.4) to approximate an optimal policy for the four-rooms environment. In this experiment we learn policies for environments with different numbers of states. For each value of $|\mathcal{S}|$, an agent approximates a policy using both methods. To compare the policies we compute the state-value function[17] corresponding to each policy. For each value function, we compute its sum over all the states $\sum_{s\in\mathcal{S}} \widehat{V}^{\pi}(s)$, and report the average and standard deviation of the sum over 50 trials. Note that a larger value is indicative of a better policy. Figure 5.14 shows the results. We note that, with the exception of one case, the agent is able to learn better policies using BOMP-TDQ than OMP-TDQ. As expected the difference between both methods is more evident for environments with king moves, since under this condition the cardinality

time is measured using the `timeit` IPython magic function [81].

[17]This computation is done using the known environment model. Note that we use the model only to evaluate the performance of the learned policies. The policies were learned using only samples from the environment.



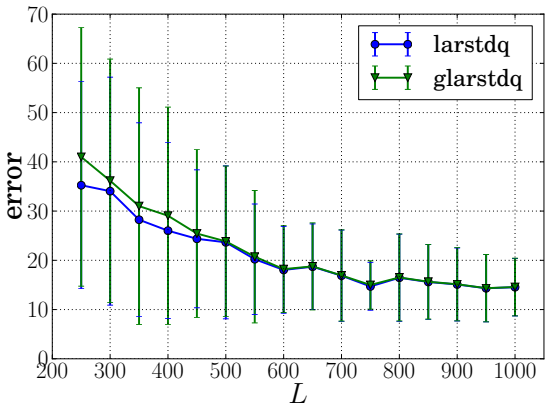
(a) Without king moves

(b) With king moves

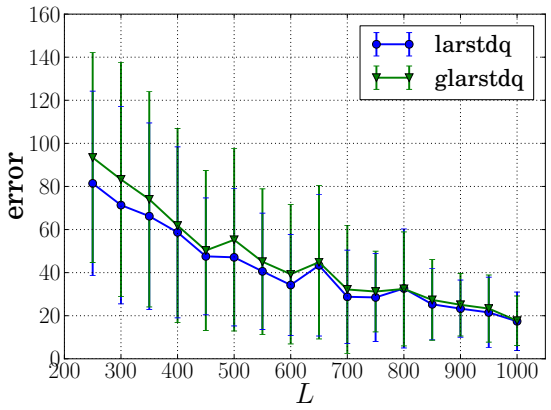
Figure 5.14: Policy iteration using OMP-TDQ and BOMP-TDQ. The plots show the average over 50 trails of the sum of the state-value function of policies learned with OMP-TDQ and BOMP-TDQ for the four-room environment, with and without king moves, for different values of the number of states $|\mathcal{S}|$. The error bars show the standard deviation.

of the action space $|\mathcal{A}|$ is 8 rather than 4.

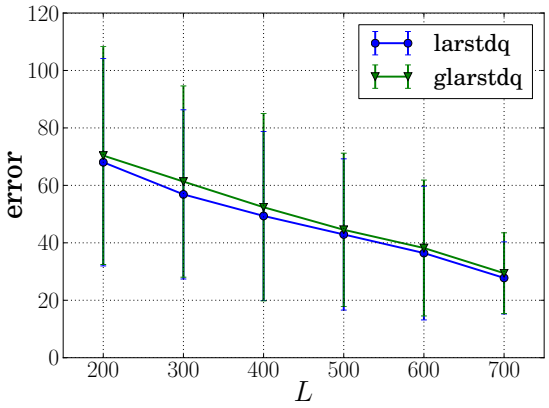
Finally, we repeat the approximation error experiment, this time comparing LARS-TDQ and GLARS-TDQ. As before, we use the four-room environment to quantify the effect of samples on the approximation error. We consider four different environments, with numbers of states set to $|\mathcal{S}| = 25$ and $|\mathcal{S}| = 50$ and with and without king moves. Figure 5.15 shows the result. Unfortunately, this time group sparsity addition does not provide any benefit, GLARS-TDQ being in fact slightly worse than LARS-TDQ. It is possible that the reason for this behavior is that, although called LARS-TDQ, this methods is in fact mimicking the LASSO estimator—note that LARS-TDQ adds and remove items to the set of nonzero features. On the other hand, GLARS-TDQ is mimicking GLARS. It is known that in general LARS is not always as effective as the LASSO estimator, making in turn GLARS-TDQ less effective. The reason why we implemented a GLARS-like method rather than a GLASSO-like method, is that while the GLARS estimate is piecewise linear, making it amenable to a homotopy implementation, the GLASSO estimate is not [104].



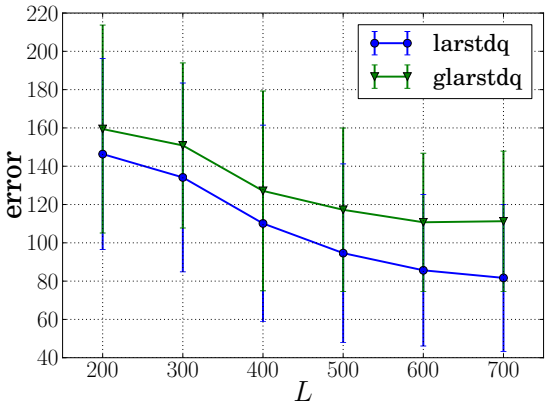
(a) $|\mathcal{S}| = 25$ without king moves



(b) $|\mathcal{S}| = 25$ with king moves



(c) $|\mathcal{S}| = 50$ without king moves



(d) $|\mathcal{S}| = 50$ with king moves

Figure 5.15: Approximation error of action-value function using LARS-TDQ and GLARS-TDQ for the four-rooms environment. Both methods use 240 nonzero features. Average error computed over 200 trials. Error bars indicate one standard deviation.

5.5 The Stationary Distribution of the Random Walk of the State-Action Graph for Deterministic 1-step Invertible Environments

The concept of *state-action graphs* can be used to find feature vectors that approximate the action-value function efficiently [77]. In this section we present a result that potentially can help to develop this approach.

Let $G = (V, E)$ be an undirected graph, and $G_{sa} = (V_{sa}, E_{sa})$ its corresponding state-

action graph [77]. For each $s \in V$ let

$$\mathcal{N}_s = (\text{neighbors of } s)$$

be the arbitrarily ordered set of neighbors of s. For a given $s \in V$ and $1 \leq a \leq d_s$, where d_s is the degree of s, the next state after taking action a when current state is s is given by the function

$$\text{nextState}_G(s, a) = \mathcal{N}_s(a).$$

Lemma 5.3

Let $(s, a) \in V_{sa}$ be a vertex of G_{sa}. The out-degree of (s, a) is equal to the degree of $\text{nextState}_G(s, a)$.

Proof. Vertex (s, a) has out-neighbors $(s', a_1), (s', a_2), \ldots, (s', a_n)$, where $s' = \text{nextState}_G(s, a)$ and $a_1, \ldots, a_n$ are the actions available at state s'. The number of available actions is equal to the degree of s'.

□

Lemma 5.4

For $s \in V$ and $(s, a) \in V_{sa}$, let $\Lambda_s = \{(s', a') \in V_{sa} : \text{nextState}_G(s', a') = s\}$. The cardinality of Λ_s is $|\Lambda_s| = d_s$.

Proof. The elements of Λ_s are all the state-action pairs with nextState_G equal to s. Thus, the number of elements in Λ_s is equal to the number of states that can transition to s, which is equal to the degree of s. □

Theorem 5.5

Let P be the random walk matrix of the state-action graph G_{sa}, and ψ the (left) Perron vector of P, i.e., it satisfies the relationship $\psi P = \psi$, and $\sum_i \psi(i) = 1$. Then, $\psi(i) = 1/|V_{sa}|$ for all i, i.e., the stationary distribution of the random walk is uniform.

Proof. Since ψ is a left eigenvector of P, $\psi = 1/|V_{sa}|$ if and only if P is a column-stochastic matrix[18]. In other words, we need to prove that $\sum_u P(u,v) = 1$ for all $v \in V_{sa}$. In general for a directed graph, P is defined as [23]

$$P(u,v) = \begin{cases} \frac{1}{d_u} & \text{if } (u,v) \in E_{sa} \\ 0 & \text{otherwise,} \end{cases} \tag{5.19}$$

where d_u is the out-degree of u. Note that since G_{sa} is the state-action graph of G, all the positive entries of the column of P corresponding to $v = (s_v, a_v)$ are the vertices $u = (s_u, a_u)$ such that $\text{nextState}_G(s_u, a_u) = s_v$. Let $\Lambda_{s_v} = \{u = (s_u, a_u) : \text{nextState}_G(s_u, a_u) = s_v\}$. It follows that for all $v \in V_{sa}$

$$\begin{aligned} \sum_{u \in V_{sa}} P(u,v) &= \sum_{\Lambda_{s_v}} P(u,v) \\ &= \sum_{\Lambda_{s_v}} \frac{1}{d_{s_u,a_u}} && \text{by (5.19)} \\ &= \sum_{\Lambda_{s_v}} \frac{1}{d_{\text{nextState}_G(s_u,a_u)}} && \text{by Lemma 5.3} \\ &= \sum_{\Lambda_{s_v}} \frac{1}{d_v} && \text{by } \text{nextState}_G(s_u,a_u) = s_v \\ &= 1 && \text{by Lemma 5.4.} \end{aligned}$$

□

[18]Note that P is always row-stochastic, but in general it is not column-stochastic.

5.5.1 Comments

Theorem 5.5 is true only for undirected graphs (if G is directed, it is easy to find a counter example that does not satisfy the theorem). This implies that this result is only useful when the environment is 1-step invertible and deterministic. Although not as general as we wish, this is still a class of problems worth studying. In particular, the Information Gathering [101] problem (at least in its simplest formulation) belongs to this class. See also [58, 59] for other examples.

At least in principle, this result suggests that the Perron vector can be used to evaluate the status of the exploration stage used to construct G_{sa}, since a G_{sa} graph with a corresponding Perron vector that is not uniform is an indication that the exploration is incomplete. It is important to note that the Perron vector is meaningful, in the sense that it represents the stationary distribution of a random walk over the graph, only if the graph is strongly connected and aperiodic [23]. We can overcome this issue by using the *PageRank* vector instead of the Perron vector (see Appendix B for more details about PageRank). Preliminary simulation results indicates that if the exploration is complete, i.e., that the estimated graph is equal to G_{sa}, then the Perron vector is equal to the *PageRank*. It might be interesting to prove this result.

CHAPTER 6

ONLINE SEARCH ORTHOGONAL MATCHING PURSUIT

> "It is not unscientific to make a guess, although many people who are not in science think it is."
>
> Richard Feynman [46, Ch. 7].

Many areas of signal processing, including Compressive Sensing, image inpainting and others, involve solving a sparse approximation problem. This corresponds to solving a system of equations $y = \Phi x$ where the matrix Φ has more columns than rows and x is a sparse vector. An important class of methods for solving this problem are the so called greedy algorithms, for which Orthogonal Matching Pursuit (OMP) is one of the classic representatives [98].

It is possible to think of greedy algorithms as instances of *search problems*. Karahanoğlu and Erdoğan [57] used the A* search method, a well known heuristic search algorithm for finding the shortest path between two nodes in a graph, to design a new greedy solver called A*OMP. This method stores the solution as a tree, where each node represents an index of the estimated support. At each iteration it selects, using a heuristic based on the evolution of the norm of the residue, which leaf node to expand. To avoid an exponential growing of the candidate solutions, this tree is pruned by keeping a relatively small number of leaves.

In this chapter we present a new greedy algorithm for solving sparse approximation problems. Like A*OMP, it frames the recovery of a sparse signal as a search instance. However, instead of using A* search which involves a monolithic planning stage, we formulate the problem as an *online* search, where the planning and execution stages are interleaved. This allows us to achieve a performance significantly better than OMP and similar to A*OMP while maintaining a reasonable computational load. Our simulations confirm this recovery performance with a computational speed 20× faster than A*OMP and less than 2× slower than OMP. The work in this chapter was published in [102].

6.1 Preliminaries

We start this section by introducing some notation and defining the problem we wish to solve. Let x be a real K-sparse vector of dimension N. That is, $x \in \mathbb{R}^N$ with $\|x\|_0 = K$, where $\|\cdot\|_0$ denotes the number of non-zero elements of a vector. Let the support of x be the index set $\text{supp}(x) = \{i \mid x(i) \neq 0\}$. Let $y = \Phi x$, with $\Phi \in \mathbb{R}^{M \times N}$ an $M \times N$ full-rank matrix with $M < N$ and $y \in \mathbb{R}^M$ a measurement or observation vector. Let ϕ_j denote column j of Φ, and we assume that $\|\phi_j\|_2 = 1$ for all j. For any index set Γ, let Φ_Γ be the $M \times |\Gamma|$ matrix corresponding to the columns of Φ indexed by Γ, where $|\Gamma|$ is the cardinality of the index set. We are interested in finding x given y and Φ. Formally, we wish to solve the non-convex optimization problem

$$\widehat{x} = \underset{x \in \mathbb{R}^N}{\text{argmin}} \|x\|_0 \quad \text{s.t.} \quad \Phi x = y. \tag{P_0}$$

Although this problem is NP-hard, it is possible under appropriate conditions (which depend on the particular values of N, M, K and Φ) to solve it using greedy methods [98].

6.1.1 Orthogonal Matching Pursuit

One of the classic greedy algorithms for solving this problem is OMP, described in Algorithm 17. OMP is an iterative algorithm that builds an estimate of the support of x by adding one index to this set at a time. The algorithm starts with an empty estimate $\Gamma^{(0)}$. It keeps a residue vector r, initially equal to y, which corresponds to the component of y perpendicular to the column span of Φ_Γ. At each iteration, OMP computes the correlation between the current residue and the columns of Φ. The index of the column with the highest correlation is added to the current estimate of the support. Using this new support estimate a new residue is computed. The loop exits when the stopping criterion is met, typically when the norm of the residue is small enough.

We will later exploit the fact that in OMP the norm of the residue vector $r^{(\ell)}$ decays

Algorithm 17 Orthogonal Matching Pursuit

input: Φ, Ψ, x_c, y, ϵ
  initialize: $r = y$, $\Lambda^{(1)} = \emptyset$, $\ell = 1$
  match: $h = \text{DFT}\{x_c\}$ (indexed from 0 to $N-1$)
  while $\|r\|_2 > \epsilon$ **do**
    identify: $k = \text{argmax}_{0 \le j \le \frac{N}{2}} |h(j)|$
    $\Lambda^{(\ell+1)} = \Lambda^{(\ell)} \cup \{k, (N-k) \mod N\}$
    $h(k) = 0$
    update: $\alpha = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Lambda^{(\ell+1)}} \|y - \Phi\Psi z\|_2$
    $r = y - \Phi\Psi\alpha$
    $\ell = \ell + 1$
  end while
output: $\widehat{x} = \Psi\alpha = \Psi\,\text{argmin}_{z:\ \text{supp}(z) \subseteq \Lambda^{(\ell)}} \|y - \Phi\Psi z\|_2$

exponentially [43, Sec. 3.1.2]. In particular,

$$\|r^{(\ell)}\|_2^2 \le (1 - \delta(\Phi))^{\ell} \|y\|_2^2, \tag{6.1}$$

where $\delta(\Phi) \in (0,1)$ is the universal decay-factor of Φ defined as $\delta(\Phi) = \inf_v \max_{1 \le j \le M} \frac{|\phi_j^T v|}{\|v\|_2^2}$. Thus, the norm of the residue converges to 0. In fact, it must reach 0 in M or fewer iterations, although a residue of 0 does not necessarily imply that the solution is correct.

6.1.2 Online Search

Search algorithms are methods that solve the problem of finding a minimal cost path between a given pair of start and goal states belonging to a state space [86]. The state space can be described by a graph where nodes represent states and weighted edges represent potential transitions between states.

Search algorithms can be broadly classified into *offline* and *online* algorithms. A way to understand the differences between these two classes is by thinking in terms of an agent that lives in the state space; the agent begins at the initial state and wants to go to the goal state. In the offline approach the agent finds a complete solution to reach the goal in what is called a planning stage, and then executes the corresponding actions. Dijkstra and A*

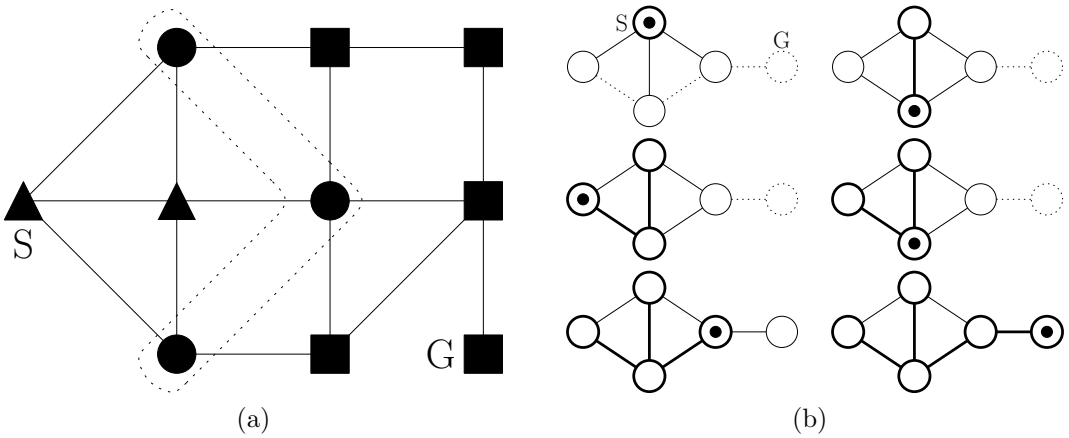


Figure 6.1: Examples of searching in a state space. S and G are the start and goal state, respectively. (a) Offline search during an intermediate stage of execution: explored, unexplored, and fringe states are represented by triangles, squares, and circles, respectively. (b) Evolution of online search. Execution depicted from left to right. A dot indicates the current state. Bold circles and edges represent visited states and transitions, respectively. Dotted lines represent unexplored regions.

are two classic offline search algorithms. Online approaches, on the other hand, interleave planning and execution of actions.

In offline searching the agent classifies the states into three different classes: *explored states*, *unexplored states*, and the *fringe*. At each iteration, the agent selects one node of the fringe (the criteria used to select the node from the fringe are what differentiate the different offline search algorithms). This node becomes an explored node, and all its successors are added to the fringe. The planning stops when the goal state is removed from the fringe. Figure 6.1(a) shows an example of a state space during an intermediate step of the planning.

As explained above, offline search algorithms must keep a list of the fringe states. When the branching factor (the number of successors for each state) is too large, this approach is unfeasible; colloquially, it said that "you run out of space before you run out of time". Online search algorithms are able to overcome situations like this one since they have memory requirements that do not depend on the number of states or the branching factor.

In online search [58] an agent starts by setting its current state s equal to the start state. Then it performs a local search among its neighbors and it moves to the state that

looks most promising. It repeats these two steps, planning and execution, until the current state is equal to the goal state. The agent uses a *value-function* $u(s)$ to store its knowledge about the state space. This function represents the current estimate of the distance to the goal for each state. The first time the value-function is evaluated it is set using a *heuristic function* $h(s)$ that returns an initial estimate of the distance to the goal. When the agent moves from the state s to the state s' it updates $u(s)$ using the value of $u(s')$. Note that the heuristic function is fixed, while the value-function changes as the agent learns about the structure of the state space. Figure 6.1(b) shows an instance of online search for a simple state space with five states.

6.2 Online Search OMP

Inspired by OMP and search methods in state spaces, we propose a new algorithm to solve (P_0). One way to think about OMP is that it searches for a support that is able to "explain" the observation vector y. This search proceeds by adding one element to the support at a time, and it is not possible to remove an element once it is added. In other words, OMP does not have the ability to backtrack. On the other hand, online search methods are backtracking algorithms that provide effective mechanisms to search for a solution in a state space. Our algorithm, dubbed "Online Search OMP" (OS-OMP), merges the above mentioned approaches. It combines the greedy addition of indices to the support based on the value of the residue used in OMP, with the use of a value-function to represent the accumulated knowledge.[1]

We adopt the "agent perspective," typically used in Artificial Intelligence [86], to explain OS-OMP. Consider an agent whose state is an index set Γ. The agent can move to any state corresponding to an index set with one extra element. It can also move to its predecessor state (it may be useful to consider this as an "undo" movement). Thus, the successor function

[1]Note that although methods such CoSaMP [73] and IHT [10] are also able to remove items from the support, they do this by estimating the complete support at each iteration, rather than by adding one element at a time.

can be written as

$$\text{succs}(\Gamma) = \{\Gamma' \mid \Gamma' \supset \Gamma, |\Gamma'| = |\Gamma| + 1\} \cup \ predecessors(\Gamma),$$

where *predecessors* is a table that keeps track of the state transitions.

Under this setup, solving (P_0) corresponds to finding, starting at the state $\Gamma = \emptyset$, the Γ with residue 0 and smallest cardinality. OS-OMP is specified entirely in Algorithm 18. It starts with an empty table to store the value function u and another empty table to store the state predecessors. The current support is set equal to the empty set. At each iteration it checks to see whether the current Γ exists in u. If it does not, this Γ is added to the table with a value equal to the heuristic function $h(\Gamma)$ (lines 3 to 5). As the heuristic function we use the norm of the residue corresponding to that support:

$$h(\Gamma) = \min_{z:\ \text{supp}(z) \subseteq \Gamma} \|y - \Phi z\|_2. \tag{6.2}$$

Note that this is the norm of the residue vector r computed in the *update* step of OMP. Also note that since this function is computed only when a support set does not have an entry in the value-function table, the computational cost of using the norm of the residue as a heuristic is reasonable. Then, OS-OMP computes and stores in the table u_{succs} the value-functions for all the successors of Γ (lines 7 to 12). As before, if there is no entry for a given successor, the value-function table is initialized using the heuristic function h.

The next steps in the algorithm are based on the following observation. As stated by (6.1), when the new elements added to the support are selected greedily, the norm of the residue decays exponentially. This implies that when the support is not sparse enough, i.e., the estimate is the wrong one, the reduction in the norm of the residue is very small. This behavior of the norm of the residue helps us to identify two search regimes: one during which the norm of the residue decays quickly, and one during which there is little change of the

norm of the residue. We consider this last situation as an indication that the current Γ is in the wrong region of the state space and that the algorithm should start backtracking. OS-OMP computes the difference between the maximum and the minimum of the value-function evaluated at the successors of Γ, and compares this difference with a threshold η. If the difference is above this threshold, Γ_{new} is set equal to the $\Gamma' \in \text{succs}(\Gamma)$ with the smallest $u(\Gamma')$; otherwise, Γ_{new} is set equal to the support in the successors with the smallest cardinality (lines 13 to 17).

After OS-OMP selects Γ_{new}, it checks to see if this new support is a backtrack move. If that is the case, the value-function of the current Γ is updated to $+\infty$. This guarantees that the path corresponding to this Γ will not be expanded in future iterations (lines 18 to 20). Finally, Γ_{new} is added to the table of predecessors if necessary (lines 21 to 23), and Γ is updated to its new value.

We continue with an example where OS-OMP works successfully but OMP fails. We set the length and sparsity of x to $N = 128$ and $K = 5$, respectively, and the length of y to $M = 19$. The matrix Φ has i.i.d. entries drawn from a standard Gaussian distribution. We compare the evolution of the norm of the residue between OS-OMP and OMP in Fig. 6.2. Note that, as explained in Sec. 6.1.1, finding a solution with a residue of norm 0 does not guarantee successful recovery of x. We observe the predicted exponential decay for OMP. For OS-OMP, on the other hand, the algorithm is able to detect that it is going in the wrong direction and backtracks several times until it finds the correct solution.

6.3 Experimental Results

In this section we empirically evaluate OS-OMP.[2] We also compare with OMP and A*OMP. For all experiments that follow we generate, for each value of the sparsity level K, signals of length $N = 128$ having K non-zero coefficients at locations selected uniformly

[2]Code available at https://github.com/aweinstein/osomp.

Algorithm 18 Online Search OMP

input: Φ, y, $\eta > 0$, stopping criterion

initialize: $\Gamma = \emptyset$, u: empty table, $predecessors$: empty table
while not converged **do**
  if $\Gamma \notin u$ **then**
    $u(\Gamma) = h(\Gamma)$
  end if
  u_{succs}: empty table
  for $\Gamma' \in succs(\Gamma)$ **do**
    if $\Gamma' \notin u$ **then**
      $u(\Gamma') = h(\Gamma')$
    end if
    $u_{succs}(\Gamma') = u(\Gamma')$
  end for
  if $\frac{\max(u_{succs}) - \min(u_{succs})}{\|y\|} > \eta$ **then**
    $\Gamma_{new} = \operatorname{argmin}(u_{succs})$
  else
    $\Gamma_{new} = \operatorname{argmin}(\{|\Gamma'| \mid \Gamma' \in succs(\Gamma)\})$
  end if
  if $|\Gamma_{new}| < |\Gamma|$ **then**
    $u(\Gamma) = +\infty$
  end if
  if $\Gamma_{new} \notin predecessors$ **then**
    $predecessors(\Gamma_{new}) = \Gamma$
  end if
  $\Gamma = \Gamma_{new}$
end while

output: $\widehat{x} = \operatorname{argmin}_{z:\ \operatorname{supp}(z) \subseteq \Gamma} \|y - \Phi z\|_2$

at random. We fix the value of M and plot the rate of perfect recovery (declared when $\|x - \widehat{x}\| \leq 10^{-4}$ and estimated using 500 trials) as a function of the sparsity level K. For each trial the matrix Φ has i.i.d. entries drawn from a standard Gaussian distribution. Since greedy methods are sensitive to the distribution of the non-zero signal coefficients, we consider three cases, each one with a different distribution.

In the first experiment we select the amplitude of the non-zero signal coefficients uniformly at random from the interval $[-2, -1] \cup [1, 2]$ and fix the dimension of y to $M = 30$. Figure 6.3(a) shows the results. Both OS-OMP and A*OMP perform significantly better than OMP, with OS-OMP performing slightly better than A*OMP for most values of K.

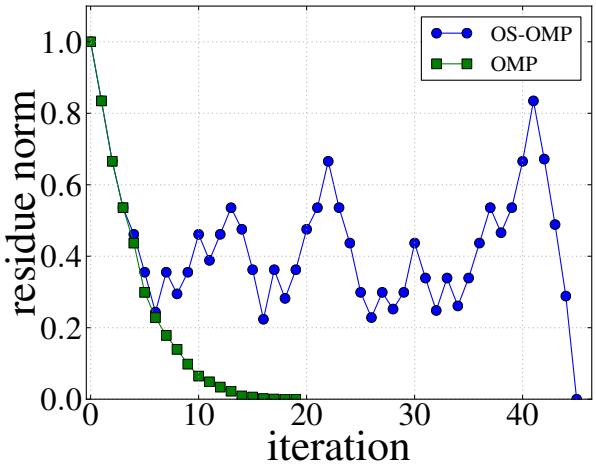
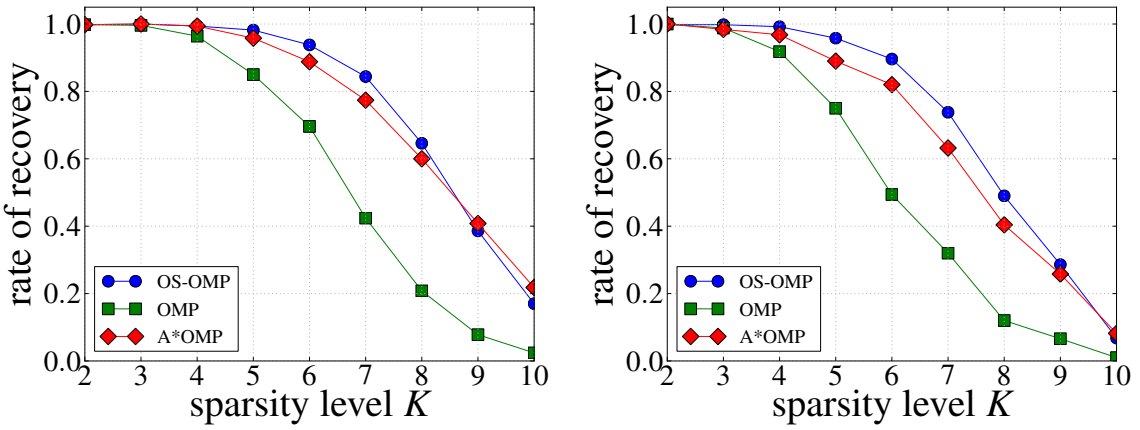


Figure 6.2: Comparison of the norm of the residue of OMP and OS-OMP for an instance with $N = 128$, $M = 19$ and $K = 5$. In this example OMP (green squares) fails to recover x, while OS-OMP (blue circles) successes.

In the second experiment we fix the magnitudes of the non-zero signal coefficients to 1 and set their signs uniformly at random. We fix the dimension of y to $M = 30$. Figure 6.3(b) shows the results. As in the previous experiment, both OS-OMP and A*OMP perform significantly better than OMP. This time the improvement of OS-OMP over A*OMP is more significant.

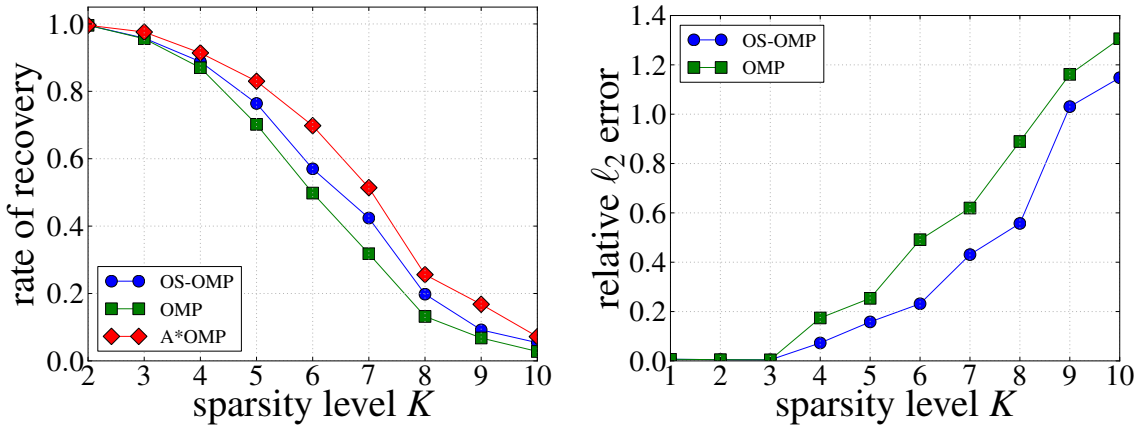
In the third experiment the amplitude of the non-zero signal coefficients are i.i.d. drawn at random from a standard Gaussian distribution. We fix the dimension of y to $M = 25$. Figure 6.3(c) shows the results. This time the difference between the three methods is less significant. OMP is still the method with the poorest performance. This time A*OMP is slightly better than OS-OMP.

Finally, we test OS-OMP in a scenario where the observations are corrupted by additive noise. We set $y = \Phi x + e$, where e is a vector with i.i.d. entries drawn from a zero-mean Gaussian distribution with standard deviation set to $\sigma = 0.1$. To handle this situation, we only need to modify the algorithm to stop when the norm of the residual is smaller than the noise level. We select the amplitude of the non-zero signal coefficients uniformly at random from the interval $[-2, -1] \cup [1, 2]$ and fix the dimension of y to $M = 30$. Figure 6.3(d) shows



(a) Uniform distribution

(b) Binary distribution



(c) Gaussian distribution

(d) Noisy observations

Figure 6.3: Experimental results. (a, b, c) Rate of perfect recovery as a function of the sparsity level K using OS-OMP, A*OMP, and OMP for three different distributions of the non-zero coefficients. (d) Relative ℓ_2 error for the recovery of x from noisy observations.

the results. We observe that OS-OMP performs better than OMP.

Although OS-OMP and A*OMP exhibit similar performance in terms of rate of recovery, we must stress the fact that the execution time of OS-OMP is significantly faster, roughly by a factor of 20. For instance, to recover a signal of length $N = 128$ and sparsity $K = 5$ from a vector y of length $M = 35$ using OS-OMP takes 5 ms, while recovering the same signal using A*OMP takes 140 ms. OMP takes 3 ms to recover the same signal.

CHAPTER 7

CONCLUSIONS

> "Probably they will fail and die, but if they flourish, it should make Randy's life a little more interesting. Not that it's devoid of interest now, but it is easier to introduce new complications than to resolve the old ones."
>
> Cryptonomicon, Neal Stephenson

In this thesis we have investigated several high-dimensional problems. Our main contribution is the design and analysis of a cornucopia of methods that, by exploiting the problem structure, allow one to solve these problems in a more effective way.

The third chapter of this thesis deals with the joint denoising problem, where we considered the situation where one needs to estimate a signal ensemble from noisy observations, under the assumptions that all the signals in the ensemble have the same support. We proposed two methods. The first one, a veto scheme, exhibited good performance, but its asymptotic behavior was not as good as desired. The second one, a voting scheme, exhibited a behavior uniformly better than the veto scheme, including a good asymptotic behavior.

There are several aspects of the joint denoising problem that deserve further considerations. Firstly, we would like to extend the proposed methods to other joint sparse models [5], such as the JSM-1 model—where each signal in the ensemble can be decomposed in a common sparse component plus a sparse innovation component—and the JSM-3 model—where each signal in the ensemble can be decomposed in a dense common component plus a sparse innovation component. Secondly, we would like to study an alternative approach to joint denoising, based on the framework of hypothesis testing. At this point it is not clear what kind of performance such methods would exhibit with respect to the ones proposed so far.

For this reason, we think it would be interesting to explore the use of this formulation.

In the fourth chapter of this thesis we have presented two methods for restoring a clipped signal using the model assumption of sparsity in the frequency domain. One of our methods, Trivial Pursuit with Clipping Constraints (TPCC), is particularly simple to implement; its running time is dominated by the computation of the Discrete Fourier Transform (DFT) and the solution of the least-squares problem, and it is significantly faster than Reweighted ℓ_1 with Clipping Constraints (Rℓ_1CC).

Our algorithms are inspired by existing techniques from Compressive Sensing (CS), and the performance we achieve (where the requisite number of non-clipped samples M scales with K) is fully in line with the state-of-the-art performance in CS. This is in spite of the fact that standard RIP analysis does not apply to the de-clipping problem and that the measurement operator in our problem is non-random and signal-dependent. Insight from CS would suggest that this signal dependence could be catastrophic for standard sparse approximation algorithms. Thus, we believe that further work is warranted to understand (i) why Rℓ_1CC offers such a substantial improvement over BPCC in the de-clipping problem and (ii) why a simple algorithm such as TPCC can succeed for de-clipping when much more complicated algorithms are required in CS.

In the fifth chapter we have presented several algorithms related to the Reinforcement Learning (RL) problem. These algorithms exploit the fact that in addition to admitting a sparse representation, action-value functions also admit a group sparse representation. The proposed method BOMP-TDQ not only exhibits a significantly better performance than its counterpart OMP-TDQ, but it is also faster. The other proposed method, GLARS-TDQ, unfortunately did not exhibit any benefit over its counterpart LARS-TDQ. Although unclear at the moment, we believe that a possible reason for this behavior is that, although called LARS-TDQ, this methods is in fact mimicking the LASSO estimator—note that LARS-TDQ adds and remove items to the set of nonzero features. On the other hand, GLARS-

TDQ is mimicking GLARS. It is known that in general LARS is not always as effective as the LASSO estimator, making in turn GLARS-TDQ less effective. The reason why we implemented a GLARS-like method rather than a GLASSO-like method, is that while the GLARS estimate is piecewise linear, making it amenable to a homotopy implementation, the GLASSO estimate is not [104]. Further work in necessary to understand this problem fully, and, ideally, to find the appropriate modifications required to extend the LARS-TDQ algorithm effectively.

At the end of the RL chapter we presented an interesting theorem related to state-action graphs. At the moment we have not been able find how to use this result, but we think that future work should also consider this problem.

In the sixth chapter we have presented a new method, called OS-OMP, for recovering a sparse vector x. The new algorithm merges ideas from greedy techniques and from online search methods. OS-OMP performs significantly better than OMP without incurring a significant extra computational load. It has a similar performance to A*OMP, a method also inspired by search algorithms, but it has a much faster execution time. Future work will include theoretical analysis of OS-OMP. We will also study the possibility of adjusting the parameter η automatically.

REFERENCES CITED

[1] J. S. Abel and J. O. Smith. Restoring a clipped signal. In *1991 International Conference on Acoustics, Speech, and Signal Processing (ICASSP-91)*, pages 1745–1748, 1991.

[2] A. Adler, V. Emiya, M. G. Jafari, M. Elad, R. Gribonval, and M. D. Plumbley. A Constrained Matching Pursuit Approach to Audio Declipping. In *2011 International Conference on Acustic, Speech and Signal Processing*, 2011.

[3] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Convex optimization with sparsity-inducing norms. *Optimization for Machine Learning*, pages 19–53, 2011.

[4] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 28(3):253–263, 2008.

[5] D. Baron, M. F. Duarte, M. B. Wakin, S. Sarvotham, and R. G. Baraniuk. Distributed Compressive Sensing. *arXiv:0901.3403 preprint*, Jan. 2009.

[6] R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[7] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Athena Scientific, 1996.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[9] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction To Probability*. Athena Scientific, 2002.

[10] T. Blumensath and M. Davies. A simple, efficient and near optimal algorithm for compressed sensing. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2009.

[11] J. M. Borwein and A. S. Lewis. *Convex Analysis and Nonlinear Optimization: Theory and Examples*. Springer Verlag, 2006.

[12] P. T. Boufounos and R. G. Baraniuk. 1-bit Compressive Sensing. In *42nd Annual Conference on Information Sciences and Systems (CISS)*, 2008.

[13] P. T. Boufounos and R. G. Baraniuk. Reconstructing sparse signals from their zero crossings. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3361–3364. Ieee, Mar. 2008.

[14] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[15] A. M. Bruckstein, D. L. Donoho, and M. Elad. From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images. *SIAM Review*, 51(1):34, 2009.

[16] L. Buşoniu, R. Babuška, D. Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Automation and Control Engineering. Taylor & Francis, 2010.

[17] E. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[18] E. Candès and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, 2005.

[19] E. J. Candès. Compressive sampling. In *Proceedings of the International Congress of Mathematicians*, pages 295–303, Spain, Jan. 2006.

[20] E. J. Candès and M. B. Wakin. An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, Mar. 2008.

[21] E. J. Candès, M. B. Wakin, and S. P. Boyd. Enhancing Sparsity by Reweighted ℓ_1 Minimization. *Journal of Fourier Analysis and Applications*, 14(5):877–905, Nov. 2008.

[22] G. Casella and R. L. Berger. *Statistical Inference*. Thomson Learning, 2002.

[23] F. Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–21, 2005.

[24] A. Cohen, W. Dahmen, and R. A. DeVore. Compressed sensing and best k-term approximation. *Journal of the American Mathematical Society*, 22(1):211–231, July 2009.

[25] A. Cohen, R. A. DeVore, S. Foucart, and H. Rauhut. Recovery of functions of many variables via compressive sensing. In *Proc. SampTA 2011, Singapore*, 2011.

[26] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.

[27] S. Cotter, B. D. Rao, K. Engan, and K. Kreutz-Delgado. Sparse solutions to linear inverse problems with multiple measurement vectors. *IEEE Transactions on Signal Processing*, 53(7), 2005.

[28] A. D'Aspremont and L. El Ghaoui. Testing the nullspace property using semidefinite programming. *arXiv:0807.3520 preprint*, 2009.

[29] J. Dattorro. *Convex Optimization and Euclidean Distance Geometry*. Meboo Publishing, 2005.

[30] M. A. Davenport and M. B. Wakin. Analysis of Orthogonal Matching Pursuit using the Restricted Isometry Property. *IEEE Transactions on Information Theory*, 56(9):4395–4401, Sept. 2010.

[31] D. Donoho and I. Johnstone. Ideal denoising in an orthonormal basis chosen from a library of bases. *Comptes rendus de l'Académie des sciences. Série I, Mathématique*, 319(12):1317–1322, 1994.

[32] D. Donoho and I. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90(432):1200–1224, 1995.

[33] D. L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627, 1995.

[34] D. L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, 2000.

[35] D. L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ^1 minimization. *Proceedings of the National Academy of Sciences of the United States of America*, 100(5):2197–202, Mar. 2003.

[36] D. L. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. *IEEE Transactions on Information Theory*, 47(7):2845–2862, 2001.

[37] D. L. Donoho and I. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[38] D. L. Donoho and Y. Tsaig. Fast Solution of ℓ_1-norm Minimization Problems When the Solution may be Sparse. Technical report, Stanford University Department of Statistics Technical Report 2006-18, 2006.

[39] S. Dreyfus. Richard Bellman on the birth of dynamic programming. *Operations Research*, pages 48–51, 2002.

[40] M. F. Duarte, M. B. Wakin, D. Baron, and R. G. Baraniuk. Universal distributed sensing via random projections. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 177–185, 2006.

[41] B. Dushnik and E. W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63(3):pp. 600–610, 1941.

[42] B. Efron, T. J. Hastie, I. M. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[43] M. Elad. *Sparse and Redundant Representations*. Springer New York, New York, NY, 2010.

[44] Y. Eldar and H. Bolcskei. Block-sparsity: Coherence and efficient recovery. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pages 2885–2888, 2009.

[45] Y. C. Eldar and G. Kutyniok. *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2012.

[46] R. P. Feynman. *The Character of Physical Law*. The MIT Press, 1965.

[47] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 2nd edition, 2000.

[48] M. Fornasier and H. Rauhut. Compressive Sensing. In O. Scherzer, editor, *Handbook of Mathematical Methods in Imaging*, chapter 6, pages 189–228. Springer New York, Jan. 2011.

[49] J. F. Gemmeke, H. Van Hamme, B. B. Cranen, and L. Boves. Compressive Sensing for Missing Data Imputation in Noise Robust Speech Recognition. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):272–287, Apr. 2010.

[50] S. Godsill and P. Rayner. *Digital audio restoration: a statistical model based approach*. Springer, 1998.

[51] S. J. Godsill, P. J. Wolfe, and W. N. Fong. Statistical Model-Based Approaches to Audio Restoration and Analysis. *Journal of New Music Research*, 30(4):323–338, Dec. 2001.

[52] R. Gonzalez and R. Woods. *Digital Image Processing*. Pearson/Prentice Hall, 2008.

[53] C. M. Grinstead and J. L. Snell. *Introduction to probability*. University Press of Florida, 2009.

[54] B. E. Hansen. *Econometrics*. Bruce Hansen, 2012.

[55] L. Jacques, J. Laska, P. Boufounos, and R. Baraniuk. Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *arXiv:1104.3160 preprint*, 2011.

[56] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions, Vol. 1*. Wiley, second edition, 1994.

[57] N. B. Karahanoğlu and H. Erdoğan. A* Orthogonal Matching Pursuit: Best-First Search for Compressed Sensing Signal Recovery. *Digital Signal Processing*, 2010.

[58] S. Koenig. Exploring unknown environments with real-time search or reinforcement learning. In K, editor, *Advances in Neural Information Processing Systems*, pages 1003–1009, 1999.

[59] S. Koenig and R. G. Simmons. Exploration with and without a map. In *Proceedings of the AAAI Workshop on Learning Action Models at the Eleventh National Conference on Artificial Intelligence (AAAI)*, pages 28–32, 1993.

[60] J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, 2009.

[61] M. G. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4(6):1107–1149, Jan. 2003.

[62] A. N. Langville and C. D. Meyer. *Google's PageRank and beyond: the science of search engine rankings*. Princeton University Press, Princeton, NJ, 2006.

[63] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer, 2007.

[64] M. Loth, M. Davy, and P. Preux. Sparse temporal difference learning using LASSO. In *Int. Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 352–359, 2007.

[65] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly. Compressed Sensing MRI. *IEEE Signal Processing Magazine*, 25(2):72–82, Mar. 2008.

[66] S. Mahadevan and M. Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[67] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, third edition, 2009.

[68] O. L. Mangasarian. Uniqueness of solution in linear programming. *Linear Algebra and its Applications*, 25(1):151–162, June 1979.

[69] H. Mansour, R. Saab, Nasiopoulos, and R. Ward. Color image desaturation using sparse reconstruction. In *Proceedings of the International Conference of Acoustics, Speech and Signal Processing (ICASSP)*, pages 4–7, 2010.

[70] J. Michels, A. Saxena, and A. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 593–600. ACM, 2005.

[71] T. Mogensen. *Basics of Compiler Design*. Torben Ægidius Mogensen, 2010.

[72] T. K. Moon and W. C. Stirling. *Mathematical Methods and Algorithms for Signal Processing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[73] D. Needell and J. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3), 2009.

[74] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX*, pages 363–372, 2006.

[75] D. J. Olive. *Applied Robust Statistics*. Southern Illinois University, 2008.

[76] T. Olofsson. Deconvolution and Model-Based Restoration of Clipped Ultrasonic Signals. *IEEE Transactions on Instrumentation and Measurement*, 54(3):1235–1240, June 2005.

[77] S. Osentoski and S. Mahadevan. Learning state-action basis functions for hierarchical MDPs. In *Proceedings of the 24th international conference on Machine learning*, pages 705–712, New York, New York, USA, 2007. ACM.

[78] C. Painter-Wakefield and R. Parr. Greedy Algorithms for Sparse Reinforcement Learning. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[79] C. Painter-Wakefield and R. Parr. L_1 Regularized Linear Temporal Difference Learning. Technical report, Duke University, Durham, NC, 2012.

[80] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 40–44. IEEE Comput. Soc. Press, 1993.

[81] F. Pérez and B. E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.

[82] Y. Plan and R. Vershynin. One-bit compressed sensing by linear programming. *arXiv:1109.4299 preprint*, 2011.

[83] W. B. Powell. *Approximate dynamic programming: solving the curses of dimensionality*. Wiley-Interscience, 2007.

[84] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, 2007.

[85] J. Proakis and D. Manolakis. *Digital Signal Processing*. Pearson Prentice Hall, 2009.

[86] S. J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, Englewood Cliffs, 2009.

[87] V. Saligrama and M. Zhao. Thresholded Basis Pursuit: An LP Algorithm for Achieving Optimal Support Recovery for Sparse and Approximately Sparse Signals from Noisy Random Measurements. *arXiv:0809.4883 preprint*, Sept. 2010.

[88] P. Smaragdis. Dynamic Range Extension Using Interleaved Gains. *IEEE Transactions on Audio, Speech and Language Processing*, 17(5):966–973, 2009.

[89] S. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub., 1997.

[90] P. Stoica, P. Babu, and J. Li. New Method of Sparse Parameter Estimation in Separable Models and its Use for Spectral Analysis of Irregularly Sampled Data. *IEEE Transactions on Signal Processing*, 59(1):35–47, Jan. 2011.

[91] C. Studer and R. G. Baraniuk. Stable Restoration and Separation of Approximately Sparse Signals. *Applied and Computational Harmonic Analysis*, Submitted, July 2012.

[92] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT press, Cambridge, Massachusetts, 1998.

[93] C. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, San Rafael, CA, Jan. 2010.

[94] V. N. Temlyakov. A remark on simultaneous sparse approximation. *East J. Approx*, 100:17–25, 2004.

[95] V. N. Temlyakov and P. Zheltov. On performance of greedy algorithms. *Journal of Approximation Theory*, 163(9):1134–1145, 2011.

[96] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[97] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[98] J. A. Tropp. Greed is Good: Algorithmic Results for Sparse Approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, Oct. 2004.

[99] J. A. Tropp and A. C. Gilbert. Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, Dec. 2007.

[100] J. A. Tropp, A. C. Gilbert, and M. Strauss. Simultaneous sparse approximation via greedy pursuit. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 721–724, 2005.

[101] A. J. Weinstein and M. B. Wakin. A Tale of Four Libraries. In *Proceedings of the 11th Python in Science Conference*, Austin, Texas, 2012.

[102] A. J. Weinstein and M. B. Wakin. Online search orthogonal matching pursuit. In *IEEE Statistical Signal Processing Workshop*, 2012.

[103] A. J. Weinstein and M. B. Wakin. Recovering a Clipped Signal in Sparseland. *Sampling Theory in Signal and Image Processing*, Oct. 2013.

[104] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B*, 68(1):49–67, 2006.

APPENDIX A

SANITY CHECK FOR THE UNIQUENESS TEST

In order to check that our implementation of the Mangasarian test is correct, we try our program with a simple example where we know the uniqueness of the solution. The linear program is given by

$$\begin{aligned} \underset{x_1,x_2}{\text{minimize}} \quad & z_i \\ \text{subject to} \quad & x_1 + x_2 \geq 1, x_1 + x_2 \leq 2 \\ & x_1 - x_2 \geq -1, x_1 - x_2 \leq 1. \end{aligned} \tag{A.1}$$

We analyze the uniqueness of the solution for three different objective functions:

$$z_1 = -x_2, \qquad z_2 = x_1 - x_2 \qquad \text{and} \qquad z_3 = x_1 + x_2.$$

As can be seen in Figure A.1, minimizing z_1 has a unique solution, while minimizing z_2 and z_3 have non-unique solutions. Our implementation of the Mangasarian test successfully computed these same results.

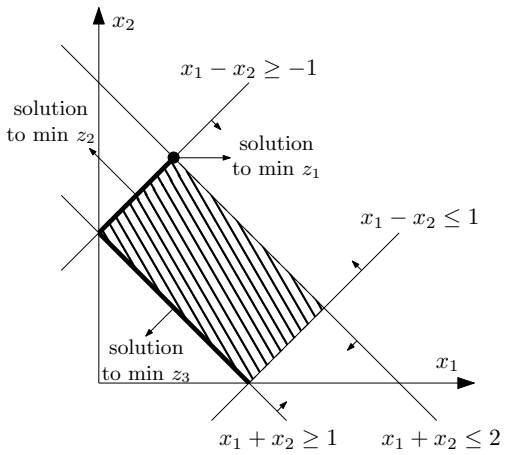


Figure A.1: Graphic representation of the Linear Program (A.1).

APPENDIX B

PAGERANK

The following details about the PageRank algorithm are based on Ch. 4 of [62].

Given a directed graph $G = (V, E)$, the hyperlink matrix (also known as the random-walk matrix, denoted by P) is given by

$$H = D^{-1}A,$$

where D is the degree matrix and A is the adjacency matrix. Let $\boldsymbol{\pi}^{(k)T}$ be the PageRank vector (or Perron vector) at the kth iteration. The iteration is given by[1]

$$\boldsymbol{\pi}^{(k+1)T} = \boldsymbol{\pi}^{(k)T}H. \tag{B.1}$$

Equation (B.1) will converge to the stationary distribution if H is stochastic, irreducible[2], and aperiodic (a matrix that is both irreducible and aperiodic is called primitive). Note that in general H does not satisfy these conditions.

The matrix H is not stochastic if there are *dangling* nodes, i.e., nodes with out-degree equal to zero. For each dangling node, H will have row equal to $\mathbf{0}^T$; and the existence of such rows implies that H is not stochastic. This problem is fixed by the "stochasticity adjustment," where each zero row in H is replaced by $\frac{1}{n}\mathbf{1}^T$. This is equivalent to letting the random-walker to jump to any node once it enters a dangling node. We can write this modified H as

$$S = H + \boldsymbol{a}\frac{1}{n}\mathbf{1}^T,$$

[1] This is a "linear stationary method". In fact, is the *power method* applied to H.

[2] G is strongly connected iff H is irreducible.

where

$$\boldsymbol{a}(i) = \begin{cases} 1 & \text{if } i \in V \text{ is a dangling node,} \\ 0 & \text{otherwise} \end{cases}$$

is the "dangling vector". Note that $\boldsymbol{a}\frac{1}{n}\mathbf{1}^T$ is a rank-one matrix. This adjustment guarantees that S is stochastic, but it does not imply that it is primitive, i.e., irreducible and aperiodic. For this reason, we also need to consider a "primitivity adjustment," which corresponds to add the teleporting capability to the random-walker. It is equivalent to constructing the *Google matrix*

$$G = \alpha S + (1-\alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^T, \qquad \text{for } 0 < \alpha < 1.$$

Since G is the convex combination of the two stochastic matrices S and $E = \frac{1}{n}\mathbf{1}\mathbf{1}^T$ it is also stochastic. Note that although G is dense, we can write it as a rank-one update to H:

$$\begin{aligned} G &= \alpha S + (1-\alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^T \\ &= \alpha\left(H\frac{1}{n}\boldsymbol{a}\mathbf{1}^T\right) + (1-\alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^T \\ &= \alpha H + (\alpha\boldsymbol{a} + (1-\alpha)\mathbf{1})\,\frac{1}{n}\mathbf{1}^T. \end{aligned} \tag{B.2}$$

Using Eq. (B.2) we can write the power method iteration as

$$\begin{aligned} \boldsymbol{\pi}^{(k+1)T} &= \boldsymbol{\pi}^{(k)T}G \\ &= \alpha\boldsymbol{\pi}^{(k)T}H + \boldsymbol{\pi}^{(k)T}(\alpha\boldsymbol{a} + (1-\alpha)\mathbf{1})\mathbf{1}^T\frac{1}{n} \\ &= \alpha\boldsymbol{\pi}^{(k)T}H + \left(\alpha\boldsymbol{\pi}^{(k)T}\boldsymbol{a} + (1-\alpha)\boldsymbol{\pi}^{(k)T}\mathbf{1}\right)\mathbf{1}^T\frac{1}{n} \\ &= \alpha\boldsymbol{\pi}^{(k)T}H + (\alpha\boldsymbol{\pi}^{(k)T}\boldsymbol{a} + 1 - \alpha)\mathbf{1}^T\frac{1}{n}. \end{aligned}$$

Since H is typically sparse, the only vector-matrix multiplication can be executed efficiently.