Chapter 2
Sparse Signal Models

This book is about modeling and exploiting *simple structure* in signals, images, and data. In this chapter, we take our first steps in this direction. We study a class of models known as *sparse models*, in which the signal of interest is a superposition of a few basic signals (called "atoms") selected from a large "dictionary." This basic model arises in a surprisingly large number of applications. It also illustrates fundamental tradeoffs in modeling and computation that will recur throughout the book.

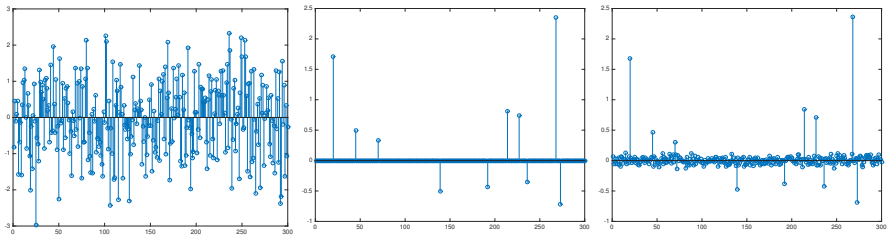
2.1 Applications of Sparse Signal Modeling

Why do we need signal models at all? We give a pragmatic answer. Many problems arising in modern signal processing and data analysis are intrinsically *ill-posed*. Often, the number of unknowns vastly exceeds the number of observations. In this situation, prior knowledge is absolutely essential to solving the problem correctly.

To describe this phenomenon mathematically, consider the simple equation

$$\underset{\text{Observation}}{\boldsymbol{y}} = \boldsymbol{A} \underset{\text{Unknown}}{\boldsymbol{x}}. \tag{2.1.1}$$

Here, $\boldsymbol{y} \in \mathbb{R}^m$ is our observation, while $\boldsymbol{x} \in \mathbb{R}^n$ is unknown. The matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ represents the data generation process: the observed data $\boldsymbol{y}$ is a linear function of the unknown (or hidden) signal $\boldsymbol{x}$. This is a simple



Dense vector **Sparse vector** **Compressible vector**

Figure 2.1. **Dense vs. sparse vectors.** Left: a *generic* vector $\boldsymbol{x} \in \mathbb{R}^n$, with entries independent standard normal random variables. Center: a *sparse* vector, with only a few nonzero entries. Right: a *compressible* vector, with only a few significant entries.

models; however, we will see that it is rich enough to bear on a vast array of practical applications.

Recovering the unknown $\boldsymbol{x}$ from observation $\boldsymbol{y}$ may appear trivial: we simply have to solve a linear system of equations! However, many practical applications raise a substantial challenge: the number of observations, m, can be significantly smaller than the number of elements n in the signal to be recovered. From linear algebra,[1] we know that when $m < n$, the system of equations $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ does not necessarily have any solution, but if it has any solution at all, then the solution space has dimension $n - m$. Hence, either there is no solution, or there are infinitely many solutions. Only one of them is the one we wish to recover! To make progress, we need to leverage some additional properties of the target solution.

Sparsity is one such property, which has strong implications on our ability to solve underdetermined systems. A vector $\boldsymbol{x} \in \mathbb{R}^n$ is considered *sparse* if only a few of its elements are nonzero. Figure 2.1 (center) shows an example of such a vector. Some form of sparsity arises naturally in almost every type of high-dimensional signal or data that we encounter in practical applications. Below, we illustrate with a few representative examples.

2.1.1 An Example from Medical Imaging

Figure 2.2 shows a *magnetic resonance* (MR) image of the brain. This is a digital image $I \in \mathbb{R}^{N \times N}$. Each entry $I(\boldsymbol{v})$ (here, $\boldsymbol{v} \in \mathbb{R}^2$) corresponds to the density of protons at a given spatial location inside the brain. This essentially indicates where water is in the brain, and can reveal many biological structures that are important for disease monitoring and diagnosis.

[1] Appendix A provides a detailed review of linear algebra and matrix analysis. In particular, Appendix A.6 reviews the existence and uniqueness of solutions to linear systems, which we use here to motivate our study of sparse approximation.

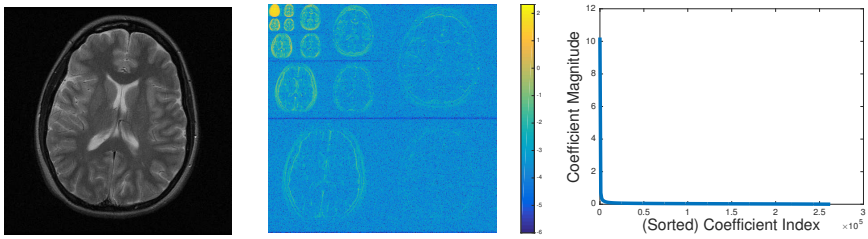


Image $I(\boldsymbol{v})$ **Wavelet coefficients $\boldsymbol{x}$: $I = \Psi[\boldsymbol{x}]$.**

Figure 2.2. **A Magnetic Resonance image.** Left: target image of a human brain. Right: coefficients in the wavelet decomposition $I = \sum_i \psi_i x_i$, and their magnitudes, sorted in descending order. The large wavelet coefficients concentrate around sharp edges in the image; wavelet coefficients corresponding to smooth regions are much smaller. The wavelet coefficients are highly compressible: their magnitude decays rapidly. Data courtesy M. Lustig [Lustig, 2013].

To caricature the MRI problem a bit, our goal is to estimate I, without opening up the brain! This is possible, if we subject the patient to a large, spatially and temporally varying magnetic field. The magnetic field causes the protons to oscillate at a frequency that depends on position their locations and energy states. Each proton essentially acts as its own radio transmitter, and in aggregate they create a signal we can measure.

As we will see from a more detailed derivation of the physical model for MRI in Chapter 11, it turns out that the signal we observe is simply a sample of the two-dimensional Fourier transform of I:

$$y = \int_{\boldsymbol{v}} I(\boldsymbol{v}) \exp(-\mathrm{i}\, 2\pi\, \boldsymbol{u}^* \boldsymbol{v})\, d\boldsymbol{v}. \tag{2.1.2}$$

Here, i is the imaginary unit, and $\cdot^*$ denotes the (complex conjugate) transpose of a vector. The two-dimensional frequency vector $\boldsymbol{u} = (u_1, u_2) \in \mathbb{R}^2$ depends how the magnetic field we applied varies over space. Here, letting $\mathcal{F}$ denote the 2D Fourier transform, the above expression is

$$y = \mathcal{F}[I](\boldsymbol{u}). \tag{2.1.3}$$

By changing the applied magnetic field, we can vary $\boldsymbol{u}$, and collect m samples of the Fourier transform, corresponding to different applied magnetic fields, parameterized by $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m\}$. We can concatenate all of our observations into a vector $\boldsymbol{y} \in \mathbb{C}^m$, given by

$$\boldsymbol{y} = \begin{bmatrix} \mathcal{F}[I](\boldsymbol{u}_1) \\ \vdots \\ \mathcal{F}[I](\boldsymbol{u}_m) \end{bmatrix} \doteq \mathcal{F}_U[I]. \tag{2.1.4}$$

Here, $\mathcal{F}_U$ is simply the operator that obtains from I the Fourier samples indexed by U. If you imagine the Fourier transform as acting by matrix

multiplication, $\mathcal{F}_U$ is simply the matrix we get if we discard all the rows of $\mathcal{F}$ that are not indexed by U.

One very basic property of the integral (2.1.2), and hence of the operator $\mathcal{F}_U$, is that it is *linear* in its input I. This means that for any pair of inputs I and J and complex scalars α, β,

$$\mathcal{F}_U[\alpha I + \beta J] = \alpha \mathcal{F}_U[I] + \beta \mathcal{F}_U[J]. \tag{2.1.5}$$

Because $\mathcal{F}_U$ is a linear operator, the problem of finding I from $\boldsymbol{y}$ using the observation equation (2.1.4) "just" consists of solving a large linear system of equations.

There is a substantial catch, though. In this system of equations, there are typically far more unknowns (here $n = N^2$) than observations m. This is necessary: it is generally too time and energy intensive to simply measure all N^2 Fourier coefficients. This is even more pressing of a concern in *dynamic MRI*, where the object being imaged is changing over time, and so acquisition needs to be time-efficient. So, in general, we need m to be as small as is just necessary to guarantee accurate reconstruction – and certainly significantly smaller than n.

This leaves us with a seemingly impossible situation: we have n unknowns and $m \ll n$ equations. Unless we can make some additional assumptions on the structure of I, the problem is ill-posed. Fortunately, real signals are not completely unstructured.[2] Figure 2.2 (right) shows a *wavelet transform* of I. The wavelet transform expresses I as a superposition of a collection of basis functions $\Psi = \{\psi_1, \ldots, \psi_{N^2}\}$:

$$\underset{\text{image}}{I} = \sum_{i=1}^{N^2} \underset{i\text{-th basis signal}}{\psi_i} \times \underset{i\text{-th coefficient}}{x_i}. \tag{2.1.6}$$

Here, $x_1, \ldots, x_{N^2} \in \mathbb{R}$ are coefficients of the image I with respect to the basis Ψ. The entries in Figure 2.2 (right) are the magnitudes $|x_i|$ for the N^2 coefficients x_i. The important point is that many of these coefficients are extremely small. If let $J = \{i_1, \ldots, i_k\}$ denote the k largest coefficients, we can approximate I as

$$\underset{\text{Target image}}{I} \quad \approx \quad \underset{\text{Superposition of } k \text{ basis functions}}{\widetilde{I}_k = \sum_{i \in J} \psi_i x_i}. \tag{2.1.7}$$

Figure 2.3 visualizes the reconstruction and reconstruction error $I - \widetilde{I}_k$. It seems that even if we retain only a relatively small fraction of the coefficients, we still obtain an accurate approximation, and most of what remains

[2] Indeed, we can construct a "generic" element I_{generic} of $\mathbb{R}^{N \times N}$, by choosing its entries at random – say from a standard Gaussian distribution $\mathcal{N}(0, 1)$. With very high probability, I_{generic} will simply look like noise. The target magnetic resonance image in Figure 2.2 certainly does not look like noise!

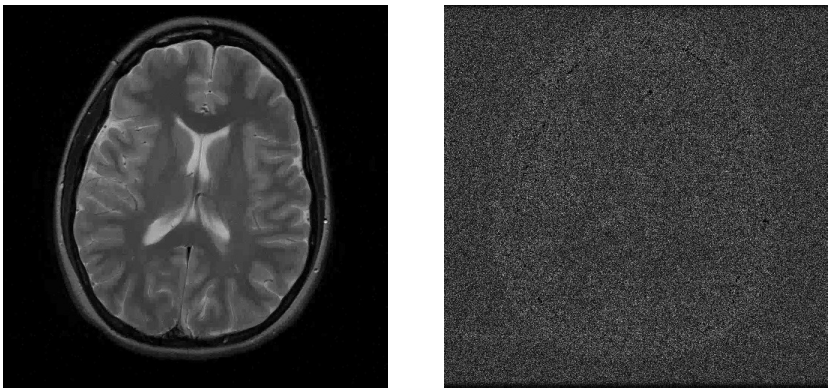


Figure 2.3. **Wavelet approximation $\hat{I}$ to I, and approximation error**. Left: approximation to the image in Figure 2.2 using the most significant 7% of the wavelet coefficients. Right: approximation error $|I - \hat{I}|$. The error contains mostly noise, suggesting that most of the important structure of the image is captured in the wavelet approximation $\hat{I}$.

is noise. This suggests that the sequence $\boldsymbol{x}$ is *compressible* – it is very close to a sparse vector.

In order to recover I, we can first try to reconstruct the sparse vector $\boldsymbol{x}$, using the observation equation

$$\begin{aligned} \underset{\text{Observed Fourier coefficients}}{\boldsymbol{y}} &= \mathcal{F}_U[I], \\ &= \mathcal{F}_U\Big[\, \psi_1 x_1 + \cdots + \psi_{N^2} x_{N^2} \,\Big], \\ &= \mathcal{F}_U[\psi_1]\, x_1 \;+\; \ldots \;+\; \mathcal{F}_U[\psi_{N^2}]\, x_{N^2}, \\ &= \Big[\mathcal{F}_U[\psi_1] \mid \cdots \mid \mathcal{F}_U[\psi_{N^2}]\Big]\, \boldsymbol{x}, \\ &\quad \text{Matrix } \boldsymbol{A} \in \mathbb{R}^{m \times N^2},\ m \ll N^2. \\ &= \boldsymbol{A}\boldsymbol{x}. \end{aligned} \tag{2.1.8}$$

After these manipulations, we end up with a system of equations $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. The vector $\boldsymbol{x}$ contains the coefficients of the target image I in the wavelet basis. The i-th column of the matrix $\boldsymbol{A}$ contains a subset U of the Fourier coefficients of the i-th basis signal ψ_i. To reconstruct I, we can look for a solution $\hat{\boldsymbol{x}}$ to this system, and then set

$$\hat{I} \;=\; \sum_{i=1}^{N^2} \psi_i \hat{x}_i. \tag{2.1.9}$$

Because $\boldsymbol{x}$ has N^2 entries, but we only have $m \ll N^2$ observations, the system $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ is underdetermined. Nevertheless, because the wavelet coefficients of I are (nearly) sparse – say only its k largest coefficients are

significant and others are negligible, the desired solution $\boldsymbol{x}$ to this system is sparse. To reconstruct I we need to find a sparse solution to an underdetermined system! In Chapter 11, we will illustrate how to actually apply such a "compressive sampling" scheme to real MRI images under more realistic conditions.

2.1.2 An Example from Image Processing

In the previous example, we used the fact that the image I had a good sparse approximation in terms of a "dictionary" of basic elements $\psi_1, \ldots, \psi_{N^2}$:

$$I \approx \sum_{i \in J} \psi_i x_i = \underset{N^2 \times N^2 \text{ matrix}}{\boldsymbol{\Psi}} \underset{\text{sparse vector}}{\boldsymbol{x}}, \tag{2.1.10}$$

where $x_i = 0$ for $i \notin J$, and $k = |J| \ll N^2$. Expressions of this form play a central role in lossy data compression. Image compression standards such as JPEG [Wallace, 1991] and JPEG 2000 [Taubman and Marcellin, 2001] leverage sparse approximations (in the discrete cosine transform (DCT) [Ahmed et al., 1974] and wavelet bases, respectively). Generally speaking, the sparser the representation is, the more input image can be compressed. However, sparse representations of images are not *just* useful for compression: they can be used for solving inverse problems, in which we try to reconstruct I from noisy, corrupted or incomplete observations. We already saw an example of this in the previous section, in which we used sparsity in the wavelet domain to reconstruct MR images. To facilitate all of these tasks, we can seek representations of I that are as sparse as possible, by replacing $\boldsymbol{\Psi}$ with more general dictionaries $\boldsymbol{A}$. For example, we might consider *overcomplete dictionaries* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $n > m$, which consist of several orthobases (e.g., DCT and wavelets together). The idea is that each individual representation may capture a particular type of signal well – say, DCT for smooth variations and wavelets for signals with sharp edges. Together, they can represent a broader class of signals.

An even more aggressive idea is to simply *learn* $\boldsymbol{A}$ from data, rather than designing it by hand. This approach tends to produce better sparsity-accuracy tradeoffs for representing images I, and is also useful for a wealth of other problems, including denoising, inpainting, and superresolution that involve reconstructing I from incomplete or corrupted observations. Each of these problems leads to an underdetermined linear system of equations; the goal is to use the prior knowledge that the target signal I has a compact representation in some dictionary $\boldsymbol{A}$ to make the problem well-posed. Figure 2.4 shows an example of this for the problem of color image denoising, from [Mairal et al., 2008a]. We observe a noisy image

$$I_{\text{noisy}} = \underset{\text{target image}}{I_{\text{clean}}} + \underset{\text{noise}}{\boldsymbol{z}}. \tag{2.1.11}$$

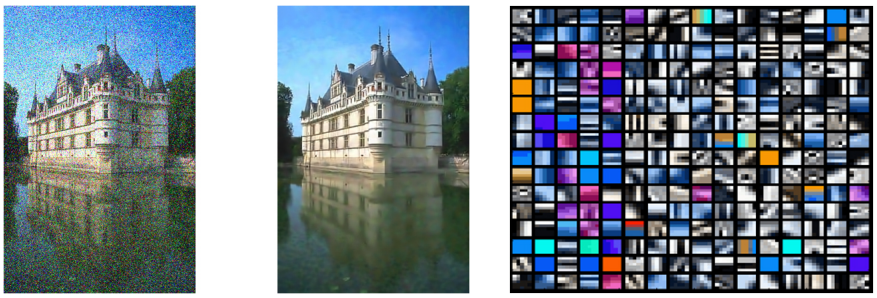


Figure 2.4. **Image denoising by sparse approximation.** Left: A noisy input image. The image is broken into patches $\boldsymbol{y}_1, \ldots \boldsymbol{y}_p$. A dictionary $\boldsymbol{A} = [\boldsymbol{a}_1 \mid \cdots \mid \boldsymbol{a}_n]$ is learned such that each input patch can be approximated as $\boldsymbol{y}_i \approx \boldsymbol{A}\boldsymbol{x}_i$, with $\boldsymbol{x}_i$ sparse. Right: dictionary patches $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$. Center: denoised image, reconstructed from the approximations $\hat{\boldsymbol{y}}_i = \boldsymbol{A}\boldsymbol{x}_i$. Results of Mairal, Saprio and Elad [Mairal et al., 2008a]. Figure from [Wright et al., 2010].

We assume[3] that patches of the clean image have an accurate sparse approximation in some dictionary $\boldsymbol{A}$: if we break I_{clean} into patches $\boldsymbol{y}_{1\text{clean}}, \ldots, \boldsymbol{y}_{p_{\text{clean}}}$

$$\underset{}{\boldsymbol{y}_{i\text{clean}}} \approx \underset{\text{patch dictionary}}{\boldsymbol{A}} \times \underset{\text{sparse coefficient vector}}{\boldsymbol{x}_i}. \tag{2.1.12}$$

In denoising, we do not actually observe $\boldsymbol{y}_{i\text{clean}}$. Rather, we observe *noisy* patches $\boldsymbol{y}_i = \boldsymbol{y}_{i\text{clean}} + \boldsymbol{z}_i$. Based on these patches $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p$, we learn[4] a dictionary $\hat{\boldsymbol{A}}$ such that

$$\underset{i\text{-th image patch}}{\boldsymbol{y}_i} \approx \underset{\text{learned dictionary}}{\hat{\boldsymbol{A}}} \times \underset{\text{sparse coefficient vector}}{\hat{\boldsymbol{x}}_i} = \underset{\text{denoised patch}}{\hat{\boldsymbol{y}}_i}.$$

We take $\hat{\boldsymbol{y}}_i = \hat{\boldsymbol{A}}\hat{\boldsymbol{x}}_i$ as an estimate of $\boldsymbol{y}_{i\text{clean}}$.

Figure 2.4 (left) shows the noisy input image; Figure 2.4 (center) shows a denoised image constructed from $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_p$. Figure 2.4 (right) shows the dictionary $\hat{\boldsymbol{A}}$ learned from the noisy patches. Although the sparse dictionary prior is relatively simple, and does not capture all of the global geometric structure of the image, it leads to surprisingly good performance on many low-level image processing tasks. We discuss modeling and computational aspects of this area in detail in Chapter **??**. For now, the key point is that the problem of reconstructing the clean image from noisy

[3] Of course, this assumption needs to be justified! See Exercise 2.14 and the notes and references to this chapter.

[4] We learn the dictionary $\hat{\boldsymbol{A}}$ by solving an optimization problem, which attempts to strike an optimal balance between sparsity of the coefficients $\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_p$ and the accuracy of the approximation $\boldsymbol{y}_i \approx \hat{\boldsymbol{A}}\hat{\boldsymbol{x}}_i$. We will explain this in more detail in Chapter **??**.

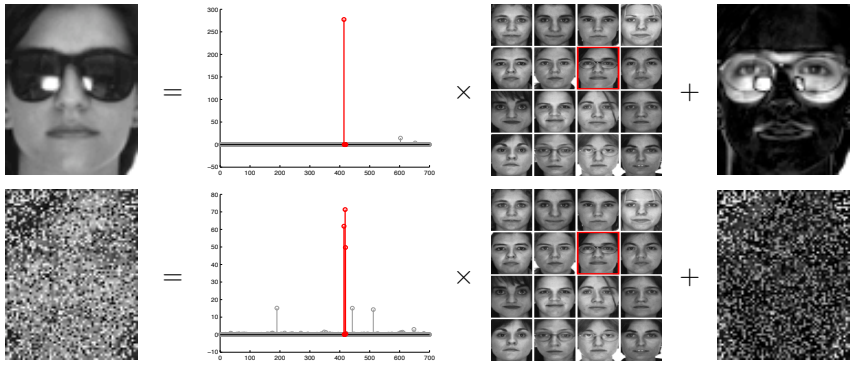


Figure 2.5. **Face recognition via sparse representation**. Top: input face image $\boldsymbol{y}$ is wearing a sunglasses; Bottom: input face image $\boldsymbol{y}$ is with 50% pixels arbitrarily corrupted. Each test image $\boldsymbol{y}$ is approximated as a sparse combination $\boldsymbol{Bx}$ of the training images, plus a sparse error $\boldsymbol{e}$ due to occlusion. In this example, red coefficients correspond to images of the correct subject.

patches again leads us to an underdetermined linear systems of equations, $\boldsymbol{y}_i \approx \boldsymbol{A}\boldsymbol{x}_i$.

2.1.3 An Example from Face Recognition

Sparsity also arises naturally in problems in which we wish to perform reliable inference from unreliable measurements. For example, due to sensor errors or malicious tampering, a vector-valued observation $\boldsymbol{y} \in \mathbb{R}^m$ might be grossly corrupted in a few of its entries:

$$\underset{\text{observation}}{\boldsymbol{y}} = \underset{\text{clean data}}{\boldsymbol{y}_o} + \underset{\text{sparse error}}{\boldsymbol{e}}. \tag{2.1.13}$$

We illustrate this more concretely using an example from automatic face recognition. Imagine that we have a database consisting of a number of subjects. For each subject i, we collect grayscale training images $I_{i,1}, \ldots, I_{i,n_i} \in \mathbb{R}^{W \times H}$, and vectorize them to form a base matrix $\boldsymbol{B}_i \in \mathbb{R}^{m \times n_i}$, with $m = W \times H$. We can further concatenate these matrices to form a large training "dictionary"

$$\boldsymbol{B} = \underset{\text{all training images}}{[\boldsymbol{B}_1 \mid \boldsymbol{B}_2 \mid \cdots \mid \boldsymbol{B}_n]} \in \mathbb{R}^{m \times n}, \qquad n = \sum_i n_i. \tag{2.1.14}$$

Suppose our system is confronted with a new image $\boldsymbol{y} \in \mathbb{R}^m$, taken under some new lighting condition, and possibly occluded – see Figure 2.5. For now, we can assume that the input $\boldsymbol{y}$ is well-aligned to the training images (i.e., the faces occur at the same position in the training and test

images).[5] There is a beautiful physical argument [Basri and Jacobs, 2003a] that shows that in an average case sense, images of "nice" objects taken under varying lighting conditions lie very close to low-dimensional linear subspaces of the high-dimensional image space $\mathbb{R}^m$. This suggests that if we have seen enough training examples, we can approximate the input sample $\boldsymbol{y}$ as a linear combination of the training samples from the same class:

$$\underset{\text{observed image}}{\boldsymbol{y}} \approx \underset{\text{linear combination of training images from } i_\star\text{-th class}}{\boldsymbol{B}_{i_\star}\boldsymbol{x}_{i_\star}}. \tag{2.1.15}$$

Unfortunately, in practice, this equation is violated in at least two ways: first, we don't know the true identity $i_\star$ ahead of time. Second, nuisance factors such as occlusion cause the equation to be badly violated on a portion of the image pixels (those that are occluded). For the first problem, we note that we can *still* write down an expression for $\boldsymbol{y}$ as a linear combination of elements of the database $\boldsymbol{B}$ as a whole: $\boldsymbol{y} \approx \boldsymbol{B}\boldsymbol{x}$. To deal with occlusion, we need to introduce an additional term $\boldsymbol{e}$, giving

$$\boldsymbol{y} \approx \boldsymbol{B}\boldsymbol{x} + \boldsymbol{e}. \tag{2.1.16}$$

Because the errors caused by occlusion are large in magnitude, this error $\boldsymbol{e}$ cannot simply be ignored, or treated with techniques designed for small noise. Unfortunately, this means that the system is underdetermined: we have m equations, but $m + n$ unknowns $\bar{\boldsymbol{x}} = (\boldsymbol{x}, \boldsymbol{e})$. Writing $\boldsymbol{A} = [\boldsymbol{B} \mid \boldsymbol{I}]$, we again have a very large underdetermined system

$$\boldsymbol{y} = \boldsymbol{A}\bar{\boldsymbol{x}}. \tag{2.1.17}$$

If we did not have prior information about $\bar{\boldsymbol{x}}$, there would be no hope of recovering it from this observation. Fortunately, both $\boldsymbol{x}$ and $\boldsymbol{e}$ are very structured. The nonzero values of $\boldsymbol{x}$ should be concentrated only on those images of the true subject, $i_\star$, and so it should be a *sparse vector*. The nonzero values of the error $\boldsymbol{e}$ should be concentrated only on those pixels that are occluded or corrupted, and so it should also be sparse.[6]

Figure 2.5 shows two examples of a sparse solution to this system of equations for a given input image $\boldsymbol{y}$. Notice that the coefficients in the estimated $\hat{\boldsymbol{x}}$ are concentrated on images of the correct subject (red) and that the error indeed corresponds to the physical occlusion. The setting

[5] Relaxing this assumption is essential to building systems that work with unconstrained input images. We will talk about how to relax this assumption in Chapter 10.

[6] Of course, the goal is to correct as many errors as possible. One of the surprises of high dimensions is it is indeed possible to correct large fractions of errors using simple, efficient algorithms. Understanding precisely how many errors we can correct (and how dense the vector $\bar{\boldsymbol{x}}$ can be before our methods break down) will be a major theoretical thrust of this book. In Chapter 10, we will give a more precise characterization about how large a fraction of errors can be corrected for a system of linear equations similar to those arise in the robust face recognition setting.

we have described so far is somewhat idealized – we will discuss both the modeling and system building aspects of this problem in more detail later in the book. For our purposes here, it is enough to note that *if* we can somehow obtain a sparse $(\boldsymbol{x}, \boldsymbol{e})$, it should suffice to identify the subject, despite nuisances such as illumination, occlusion, and corruption.

2.2 Recovering a Sparse Solution

Suppose, as in the above examples, that we know the ground truth signal $\boldsymbol{x}_o$ is sparse. How powerful is this knowledge? Can it render ill-posed problems such as MR image acquisition or occluded face recognition well-posed? To answer these questions, we need a formal notion of sparsity. In the next two subsections, we begin by introducing the concept of a norm of a vector, which generalizes the concept of *length*. We then introducing an "ℓ^0 norm", which counts the number of nonzero entries in a vector, a basic measure of how dense (or sparse) that vector is.

2.2.1 Norms on Vector Spaces

A *vector space* $\mathbb{V}$ consists of a collection of elements (vectors), field such as the real numbers $\mathbb{R}$ or complex numbers $\mathbb{C}$ (scalars) and operations (adding vectors and multiplying vectors with scalars) that work in ways that conform to our intuitions from $\mathbb{R}^3$. Appendix A reviews the formal definition of a vector space, and gives examples. In the above application examples, our signals of interest consisted of collections of real or complex numbers – e.g., in MR imaging, the target image I was an element of $\mathbb{R}^{N\times N}$. We can view $\mathbb{R}^{N\times N}$ as a vector space, with scalar field $\mathbb{R}$ (write $\mathbb{V} = (\mathbb{R}^{N\times N}, \mathbb{R})$). In the other examples as well, the signals of interest reside in vector spaces.

A *norm* on a vector space $\mathbb{V}$ gives *a way of measuring lengths of vectors, that conforms in important ways to our intuition* from lengths in $\mathbb{R}^3$. Formally:

Definition 2.2.1. *A* **norm** *on a vector space* $\mathbb{V}$ *over* $\mathbb{R}$ *is a function* $\|\cdot\| : \mathbb{V} \to \mathbb{R}$ *that is*

1. *Nonnegatively homogeneous:* $\|\alpha\boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\|$ *for all vectors* $\boldsymbol{x} \in \mathbb{V}$, *scalars* $\alpha \in \mathbb{R}$,
2. *Positive definite:* $\|\boldsymbol{x}\| \geq 0$, *and* $\|\boldsymbol{x}\| = 0$ *if and only if* $\boldsymbol{x} = \boldsymbol{0}$,
3. *Subadditive:* $\|\cdot\|$ *satisfies the triangle inequality* $\|\boldsymbol{x}+\boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ *for all* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$.

For our purposes, the most important family of norms are the ℓ^p norms. We will use norms from this family to derive practical algorithms for find-

ing sparse solutions to linear systems of equations, and for studying their properties. If we take $\mathbb{V} = (\mathbb{R}^n, \mathbb{R})$, and $p \in (0, \infty)$, we can write

$$\|\boldsymbol{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}. \tag{2.2.1}$$

The function $\|\boldsymbol{x}\|_p$ is a norm for any $p \geq 1$. The most familiar example is the ℓ^2 norm or "Euclidean norm"

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_i |x_i|^2} = \sqrt{\boldsymbol{x}^* \boldsymbol{x}}$$

which coincides with our usually way of measuring lengths. Two other cases are of almost equal importance: $p = 1$, and $p \to \infty$. Setting $p = 1$ in (A.9.1), we obtain

$$\|\boldsymbol{x}\|_1 = \sum_i |x_i|, \tag{2.2.2}$$

which will play a very large role in this book.[7] Finally, as p becomes larger, the expression in (A.9.1) accentuates large $|x_i|$. As $p \to \infty$, $\|\boldsymbol{x}\|_p \to \max_i |x_i|$. We extend the definition of the ℓ^p norm to $p = \infty$ by defining

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i|. \tag{2.2.3}$$

To appreciate the distinction between the various ℓ^p norms, we can visualize their unit balls $\mathsf{B}_{\|\cdot\|_p}$, which consist of all vectors $\boldsymbol{x}$ whose norm is at most one:

$$\mathsf{B}_{\|\cdot\|_p} = \left\{ \boldsymbol{x} \mid \|\boldsymbol{x}\|_p \leq 1 \right\}. \tag{2.2.4}$$

The ℓ^2 ball is a (solid) sphere, the ℓ^∞ ball is a cube, and the ℓ^1 ball a kind of diamond shape – see Figure 2.6. [8]

Notice that for $p \leq p'$, $\mathsf{B}_{\|\cdot\|_p} \subseteq \mathsf{B}_{\|\cdot\|_{p'}}$. This is because when $p \leq p'$, $\|\boldsymbol{x}\|_p \geq \|\boldsymbol{x}\|_{p'}$ for all $\boldsymbol{x}$.

Remark 2.2.2. *This containment becomes even more striking in higher dimensions: in* $\mathbb{R}^n$, $\mathrm{vol}(\mathsf{B}_\infty) = 2^n$, *while* $\mathrm{vol}(\mathsf{B}_1) = 2^n/n!$ *(see e.g., [Matousek, 2002a]). So, in* $n = 2$ *dimensions* $\mathrm{vol}(\mathsf{B}_1) = (1/2) \times \mathrm{vol}(\mathsf{B}_\infty)$, *while in* $n = 1,000$ *dimensions* $\mathrm{vol}(\mathsf{B}_1) \approx 10^{-2,568} \times \mathrm{vol}(\mathsf{B}_\infty)$ – *a truly negligible fraction!*

[7] Anyone who has traveled in Manhattan should have good appreciation for the distinction between ℓ^1 and ℓ^2 – in fact, the ℓ^1 norm is sometimes called the Manhattan norm! This example illustrates a simple, but important point – the proper choice of norm depends quite a bit on the properties of the problem and design goals. Unless you can leap tall buildings in a single bound, measuring distance using the ℓ^2 norm would underestimate how much travel you need to reach your destination.

[8] To see this in action, you can run `Chapter_2_Illustrate_Lp_Balls.m`.

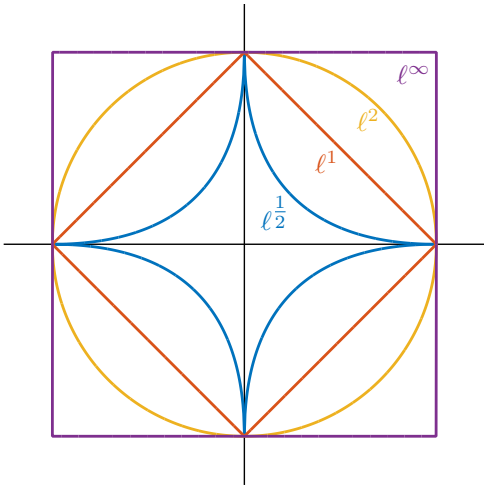


Figure 2.6. **The ℓ^p balls $\mathsf{B}_p = \left\{ \boldsymbol{x} \mid \|\boldsymbol{x}\|_p \leq 1 \right\}$** for $0 < p \leq \infty$. For $p \geq 1$, B_p is a convex set, and $\|\cdot\|_p$ is a norm. For $p < 1$, $\|\cdot\|_p$ is not a norm, in the formal sense.

Remark 2.2.3. *This may seem to be in contrast to the mathematical fact that "in finite dimensions, all norms are equivalent" (see, e.g., Appendix A). Formally, this statement means that in a finite dimensional vector space $\mathbb{V}$, such as $\mathbb{R}^n$, for any pair of norms $\|\cdot\|_\diamond$ and $\|\cdot\|_\square$ there exist numbers $0 < \alpha, \beta < \infty$ such that for every $\boldsymbol{x} \in \mathbb{V}$,*

$$\alpha \|\boldsymbol{x}\|_\square \;\leq\; \|\boldsymbol{x}\|_\diamond \;\leq\; \beta \|\boldsymbol{x}\|_\square \,. \tag{2.2.5}$$

So, the norms $\|\cdot\|_\square$ and $\|\cdot\|_\diamond$ can be compared in size. However, as the example in Remark 2.2.2 shows, in high dimensions, the unit balls of the various ℓ^p norms can be very different. In applications involving high-dimensional signals, the different choices of norm can lead to radically different solutions.

2.2.2 The ℓ^0 Norm

With the notion of a norm in hand, we are prepared to define a formal notion of sparsity. For this, we introduce a function, called the "ℓ^0 norm" (read "ell zero norm"), which is simply the number of nonzero entries in a vector $\boldsymbol{x}$:

$$\|\boldsymbol{x}\|_0 = \#\left\{ i \mid \boldsymbol{x}(i) \neq 0 \right\}. \tag{2.2.6}$$

Loosely speaking, $\boldsymbol{x}$ is sparse whenever $\|\boldsymbol{x}\|_0$ is small.

The ℓ^0 norm $\|\cdot\|_0$ is *not* a norm, in the formal sense of Definition A.9.1: since for $\alpha \neq 0$, $\|\alpha\boldsymbol{x}\|_0 = \|\boldsymbol{x}\|_0$, it does not have the property of nonnegative

homogeneity. It *does* have the other two properties, however. In particular, $\|\cdot\|_0$ is subadditive:

$$\forall\, \boldsymbol{x},\ \boldsymbol{x}', \qquad \|\boldsymbol{x}+\boldsymbol{x}'\|_0 \ \leq\ \|\boldsymbol{x}\|_0 + \|\boldsymbol{x}'\|_0\,. \tag{2.2.7}$$

This is easily checked by noting that the set of nonzero entries for $\boldsymbol{x}+\boldsymbol{x}'$ is contained union of the set of nonzero entries of $\boldsymbol{x}$ and the set of nonzero entries of $\boldsymbol{x}'$.

Although the ℓ^0 norm is not a norm in the mathematical sense, it is closely related to the ℓ^p norms. To understand this relationship, note that for every $\boldsymbol{x} \in \mathbb{R}^n$,

$$\begin{aligned} \lim_{p\searrow 0} \|\boldsymbol{x}\|_p^p &= \sum_{i=1}^{n} \lim_{p\searrow 0} |\boldsymbol{x}(i)|^p \\ &= \sum_{i=1}^{n} \mathbb{1}_{\boldsymbol{x}(i)\neq 0} \\ &= \|\boldsymbol{x}\|_0. \end{aligned} \tag{2.2.8}$$

In this sense, the ℓ^0 norm can be considered to be generated from the ℓ^p norms, by taking p (infinitesimally) small. In the context of Figure 2.6, this can be understood as follows: in $\mathbb{R}^2$, the sparse vectors correspond to the coordinate axes. As p drops towards zero, the unit ball of the ℓ^p norm becomes more concentrated around the coordinate axes, i.e., around the sparse vectors.

The geometric relationship between the ℓ^0 and ℓ^p norms is useful for deriving algorithms, and for understanding why small p tends to favor sparse solutions. With this said, the formal notation $\|\boldsymbol{x}\|_0$ has a very simple meaning: *it counts the number of nonzero entries in* $\boldsymbol{x}$. In all of the applications discussed above, our goal is to recover a vector $\boldsymbol{x}_{\text{true}}$ with $\|\boldsymbol{x}_{\text{true}}\|_0$ small. In this book, we often use $\boldsymbol{x}_o$ as a shorthand for $\boldsymbol{x}_{\text{true}} = \boldsymbol{x}_o$.

2.2.3 The Sparsest Solution: Minimizing the ℓ^0 Norm

Suppose we observe $\boldsymbol{y} \in \mathbb{R}^m$, with $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$, and that our goal is to recover $\boldsymbol{x}_o$. If we know that $\boldsymbol{x}_o$ is sparse, it seems reasonable to form an estimate $\hat{\boldsymbol{x}}$ by choosing the *sparsest* vector $\boldsymbol{x}$ that satisfies the equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. That is, we choose the sparsest $\boldsymbol{x}$ that could have generated our observation. We can write this as an optimization problem

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{x}\|_0 \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \end{array} \tag{2.2.9}$$

How might we solve this problem numerically? Call

$$\text{supp}\,(\boldsymbol{x}) = \{i \mid \boldsymbol{x}_i \neq 0\} \subset \{1 \ldots n\} \tag{2.2.10}$$

the *support* of the vector $\boldsymbol{x}$ – this set contains the indices of the nonzero entries. The ℓ^0 minimization problem (2.2.9) asks us to find a vector $\boldsymbol{x}$

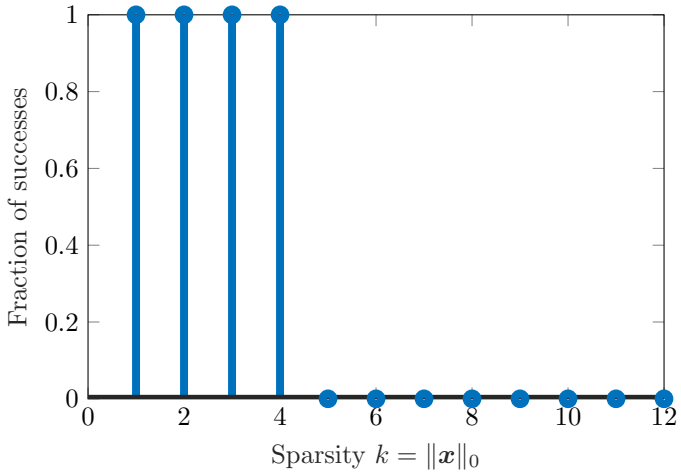


Figure 2.7. **Transitions in ℓ^0 recovery.** Fraction of correct recoveries across 100 trials, as a function of sparsity of the target solution $\boldsymbol{x}_o$. The system is of size 5×12. In this experiment, ℓ^0 minimization successfully recovers all $\boldsymbol{x}_o$ with $k \leq 4$ nonzeros.

of smallest support that agrees with the observation $\boldsymbol{y}$. One approach to finding such an $\boldsymbol{x}$ is to simply try every possible support set $I \subseteq \{1, \ldots, n\}$ For each such set I, we can form a system of equations

$$\boldsymbol{A}_I \boldsymbol{x}_I = \boldsymbol{y}, \tag{2.2.11}$$

where $\boldsymbol{A}_I \in \mathbb{R}^{m \times |I|}$ is the column submatrix of $\boldsymbol{A}$ formed by keeping only those columns indexed by I, and $\boldsymbol{x}_I \in \mathbb{R}^{|I|}$. We can attempt to solve (2.2.11) for $\boldsymbol{x}_I$. If such an $\boldsymbol{x}_I$ exists, we can obtain a solution $\boldsymbol{x}$ to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ by filling in the remaining entries of $\boldsymbol{x}$ with zeros. This exhaustive search procedure is spelled out formally as Algorithm 2.1.

Algorithm 2.1: ℓ^0-minimization by exhaustive search

Input: a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and a vector $\boldsymbol{y} \in \mathbb{R}^m$.
for $k = 0, 1, 2, \ldots, n$,
  for each $I \subseteq \{1, \ldots, n\}$ of size k,
    if the system of equations $\boldsymbol{A}_I \boldsymbol{z} = \boldsymbol{y}$ has a solution $\boldsymbol{z}$,
      set $\boldsymbol{x}(I) = \boldsymbol{z}$, $\boldsymbol{x}(I^c) = \boldsymbol{0}$.
      return $\boldsymbol{x}$.
    end if
  end for
end for

Example 2.2.4. *Let us examine how the algorithm behaves numerically, using* `Chapter_2_L0_recovery.m` *and* `Chapter_2_L0_transition.m`*. These examples generate random underdetermined linear systems* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$*, with* $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$*, and* $\boldsymbol{x}_o$ *sparse. They then apply Algorithm 2.1 (*`minimize_L0.m`*) to recover a vector* $\hat{\boldsymbol{x}}$*, and ask whether* $\hat{\boldsymbol{x}}$ *is equal to* $\boldsymbol{x}_o$ *up to machine precision. Fixing the system parameters* (m, n)*, varying the sparsity* $k = 0, 1, \ldots,$ *and performing many random trials, we produce Figure 2.7, which shows that as long as the* k *is not too large, the algorithm almost always succeeds.*

Is there any mathematical explanation for this phenomenon? To understand why ℓ^0 minimization succeeds, it is worth first thinking about when it would fail. Suppose that there is a k-sparse vector $\boldsymbol{x}_o \in \text{null}\,(\boldsymbol{A})$. Then

$$\boldsymbol{A}\boldsymbol{x}_o \;=\; \boldsymbol{0} \;=\; \boldsymbol{A}\boldsymbol{0}. \tag{2.2.12}$$

Hence, for this $\boldsymbol{x}_o$, when $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$, the ℓ^0 minimizer is simply $\hat{\boldsymbol{x}} = \boldsymbol{0}$, and $\boldsymbol{x}_o$ is not recovered. Put simply: if the null space of $\boldsymbol{A}$ contains sparse vectors (aside from $\boldsymbol{0}$), ℓ^0 minimization may fail to recover the desired sparse vector $\boldsymbol{x}_o$.

In fact, the converse statement is also true: when the null space of $\boldsymbol{A}$ *does not* contain sparse vectors (aside from $\boldsymbol{0}$), ℓ^0 minimization *does* recover any sufficiently sparse vector $\boldsymbol{x}_o$. To state the argument simply, let us suppose that $\|\boldsymbol{x}_o\|_0 \leq k$, and

$(\star)$ *The only* $\boldsymbol{\delta} \in \text{null}\,(\boldsymbol{A})$ *with* $\|\boldsymbol{\delta}\|_0 \leq 2k$ *is* $\boldsymbol{\delta} = \boldsymbol{0}$.

Let $\hat{\boldsymbol{x}}$ denote the solution to the ℓ^0 minimization problem, so $\|\hat{\boldsymbol{x}}\|_0 \leq \|\boldsymbol{x}_o\|_0 \leq k$. If we define the *estimation error*

$$\boldsymbol{\delta} = \hat{\boldsymbol{x}} - \boldsymbol{x}_o, \tag{2.2.13}$$

then

$$\begin{aligned} \|\boldsymbol{\delta}\|_0 &= \|\hat{\boldsymbol{x}} - \boldsymbol{x}_o\|_0 \\ &\leq \|\hat{\boldsymbol{x}}\|_0 + \|\boldsymbol{x}_o\|_0 \\ &\leq 2k. \end{aligned} \tag{2.2.14}$$

So, $\boldsymbol{\delta}$ is a sparse vector. Moreover,

$$\begin{aligned} \boldsymbol{A}\boldsymbol{\delta} &= \boldsymbol{A}\,(\hat{\boldsymbol{x}} - \boldsymbol{x}_o) \\ &= \boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{A}\boldsymbol{x}_o \\ &= \boldsymbol{y} - \boldsymbol{y} \;=\; \boldsymbol{0}. \end{aligned} \tag{2.2.15}$$

So, $\boldsymbol{\delta}$ is a sparse vector *in the null space of* $\boldsymbol{A}$. Property $(\star)$ states that the only sparse vector in $\text{null}(\boldsymbol{A})$ is $\boldsymbol{0}$. So, if $(\star)$ holds, $\boldsymbol{\delta} = \boldsymbol{0}$, and so $\hat{\boldsymbol{x}} = \boldsymbol{x}_o$: ℓ^0 minimization indeed recovers $\boldsymbol{x}_o$.

Property $(\star)$ is a property of the matrix $\boldsymbol{A}$. The above reasoning suggests a slogan: *the "good"* $\boldsymbol{A}$ *for recovering sparse vectors* $\boldsymbol{x}_o$ *are those* $\boldsymbol{A}$ *that have no sparse vectors in their null space.* We can restate property $(\star)$ more

conveniently in terms of the columns of $\boldsymbol{A}$: property ($\star$) holds if and only if every set of $2k$ columns of $\boldsymbol{A}$ is linearly independent.

Definition 2.2.5 (Kruskal rank [Kruskal, 1977])**.** *The* Kruskal rank *of a matrix $\boldsymbol{A}$, written* $\operatorname{krank}(\boldsymbol{A})$ *is the largest number r such that every subset of r columns of $\boldsymbol{A}$ is linearly independent.*

From the above reasoning, if $\|\boldsymbol{x}_o\|_0$ is at most half of $\operatorname{krank}(\boldsymbol{A})$, ℓ^0 minimization will recover $\boldsymbol{x}_o$:

Theorem 2.2.6 (ℓ^0 recovery)**.** *Suppose that $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$, with*

$$\|\boldsymbol{x}_o\|_0 \;\leq\; \tfrac{1}{2}\operatorname{krank}(\boldsymbol{A}). \tag{2.2.16}$$

Then $\boldsymbol{x}_o$ is the unique optimal solution to the ℓ^0 minimization problem

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{x}\|_0 \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \end{array} \tag{2.2.17}$$

Notice that Theorem 2.2.6 agrees with the behavior in Figure 2.7.[9] Theorem 2.2.6 predicts that as long as $\boldsymbol{x}_o$ is *sufficiently sparse*, it will be recovered by ℓ^0 minimization. The level of allowable sparsity depends on the Kruskal rank of the matrix $\boldsymbol{A}$. It is not hard to see that in general,

$$0 \leq \operatorname{krank}(\boldsymbol{A}) \leq \operatorname{rank}(\boldsymbol{A}). \tag{2.2.18}$$

For "generic" $\boldsymbol{A}$, the Kruskal rank is quite large:

Proposition 2.2.7. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $n \geq m$, with $\boldsymbol{A}_{ij}$ independent identically distributed $\mathcal{N}(0,1)$ random variables. Then, with probability one,* $\operatorname{krank}(\boldsymbol{A}) = m$.

Proof. Exercise 2.6 guides the interested reader through the proof. □

The intuition is that to have $\operatorname{krank}(\boldsymbol{A}) < m$, there must be some subset of m columns of $\boldsymbol{A}$ which are linearly dependent, i.e., there is some subset $\boldsymbol{a}_{i_1}, \boldsymbol{a}_{i_2}, \ldots, \boldsymbol{a}_{i_m}$ which lie on a linear subspace of dimension $m-1$. For a Gaussian random matrix $\boldsymbol{A}$, the probability that this happens is zero. This is true of many other random matrices.[10] We can interpret this as saying that under generic circumstances, knowing that the target $\boldsymbol{x}_o$ is sparse turns an ill-posed problem into a well-posed one. The ℓ^0 minimization problem recovers vectors $\boldsymbol{x}_o$ whose number of nonzeros is as large as $\frac{m}{2}$. This level of sparsity is well beyond what is needed for most applications.

[9] Actually, the behavior in Figure 2.7 is slightly better than what Theorem 2.2.6 predicts – with probability one the Kruskal rank of $\boldsymbol{A}$ is m, and so the theorem shows that ℓ^0 minimization succeeds when $k \leq \frac{m}{2} = 2$. However, in the experiment, success always occurs when $k \leq 4$. Exercise 2.7 asks you to explain this discrepancy, by proving a modified version of Theorem 2.2.6.

[10] For example, $\operatorname{krank}(\boldsymbol{A}) = m$ with probability one whenever $\boldsymbol{A}$ is distributed according to any absolutely continuous measure, i.e., there is a probability density function.

2.2.4 Computational Complexity of ℓ^0 Minimization

The theoretical results in the previous section show the power of sparsity: knowing that the target solution $\boldsymbol{x}_o$ is even moderately sparse can render the problem of recovering $\boldsymbol{x}_o$ well-posed. Unfortunately, Algorithm 2.1 is not very useful in practice. Its worst-case running time is on the order of n^k, where $k = \|\boldsymbol{x}_o\|_0$ is the number of nonzero entries we wish to recover. For example, at the time of writing this book, to solve a problem with $m = 50$, $n = 200$, and $k = 10$, on a standard laptop, Algorithm 2.1 would require ≈ 140 *centuries*. This is still a very small problem by the standard of most modern-day applications!

Exhaustively searching all possible supports I may not seem like a particularly intelligent strategy for solving (2.2.9). However, no significantly better algorithm is currently known. There is some reason to believe that none will ever be found:

Theorem 2.2.8 (Hardness of ℓ^0 minimization)**.** *The ℓ^0-minimization problem* (2.2.9) *is* NP*-hard.*

NP-hard problems are considered highly unlikely to be efficiently solvable (i.e., solvable on standard (model) computers in time polynomial in the size of the problem). This class of problems includes notoriously difficult examples, such as the Traveling Salesman Problem. Fully appreciating the mathematical content of Theorem 2.2.8 requires formal modeling of computation (Turing machines, complexity theory for problem classes P/NP, etc.) that is beyond our scope. For readers unfamiliar with these details, Appendix 2.A gives a brief, high-level description of the issues that arise. For practical purposes, we should interpret the result as indicating that we cannot expect to give an efficient algorithm that guarantees to solve all instances of (2.2.9).

Hardness results such as Theorem 2.2.8 are typically proved by reduction: we show that if we can solve the problem of interest efficiently, this would allow us to also efficiently solve some other problem, which is already known to be hard. For the ℓ^0 minimization problem, we do this by showing that ℓ^0 minimization can be used to solve certain (hard) set covering problems:

Proof of Theorem 2.2.8. Consider the following problem:

> *Exact 3-Set Cover (E3C)*: Given a set $S = \{1, 2, \ldots, m\}$ and a collection $\mathcal{C} = \{U_1, \ldots, U_n\}$ of subsets $U_j \subseteq S$ each of which has size $|U_j| = 3$, does there exist a subcollection $\mathcal{C}' \subseteq \mathcal{C}$ that exactly covers S, i.e., $\forall i \in S$ there is exactly one $U \in \mathcal{C}'$ with $i \in U$.

This problem is known to be NP-hard [Garey and Johnson, 1979, Karp, 1972]. To reduce it to ℓ^0 minimization, suppose that we are given an instance of E3C: Form a $m \times n$ matrix $\boldsymbol{A} \in \{0, 1\}^{m \times n}$ by letting $\boldsymbol{A}_{ij} = 1$ if

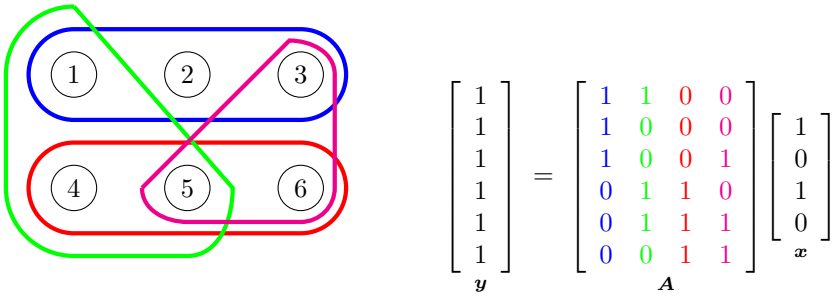


Figure 2.8. **Exact 3-Set Cover as a Sparse Representation problem.** Left: a universe $S = \{1, \ldots, 6\}$ and four subsets $U_1, \ldots, U_4 \subseteq S$. $\{U_1, U_3\}$ is an exact 3-set cover. Right: the same problem as a linear system of equations. The columns of $\boldsymbol{A}$ are the incidence vectors for the sets U_1, U_2, U_3, U_4. The Exact 3-Cover $\{U_1, U_3\}$ corresponds to a solution $\boldsymbol{x}$ to the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ with only $m/3 = 2$ nonzero entries.

$i \in U_j$, and $\boldsymbol{A}_{ij} = 0$ otherwise. Set $\boldsymbol{y} = \mathbf{1} \in \mathbb{R}^m$ (i.e., an m-dimensional vector of ones). Figure 2.8 illustrates this construction. We show:

> *Claim:* The system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ has a solution $\boldsymbol{x}_o$ with $\|\boldsymbol{x}_o\|_0 \leq m/3$ if and only if there exists an exact 3-set cover.

($\Leftarrow$) Suppose there exists an exact 3-set cover $\mathcal{C}'$. Clearly, $|\mathcal{C}'| = m/3$. Set

$$x_j = \begin{cases} 1 & U_j \in \mathcal{C}' \\ 0 & \text{else} \end{cases}.$$

Then $\|\boldsymbol{x}\|_0 = m/3$, and $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$.

($\Rightarrow$) Let $\boldsymbol{x}_o$ be a solution to $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ with at most $m/3$ nonzero entries. Set $\mathcal{C}' = \{U_j \mid x_o(j) \neq 0\}$. We claim $\mathcal{C}'$ is the desired cover. Let $I = \operatorname{supp}(\boldsymbol{x}_o)$. Since each column of $\boldsymbol{A}$ has exactly 3 nonzero entries, and $\boldsymbol{A}_I$ has at most $m/3$ columns, the matrix $\boldsymbol{A}_I$ has at most m nonzero entries. Since $\boldsymbol{A}_I \boldsymbol{x}_o(I) = \boldsymbol{y}$, each row of $\boldsymbol{A}_I$ has at least one nonzero entry. Hence, each row of $\boldsymbol{A}_I$ has *exactly* one nonzero entry, and the set $\mathcal{C}'$ gives an exact cover. □

In fact, the truth is even worse than Theorem 2.2.8 suggests: The ℓ^0 minimization problem remains NP-hard even if we only demand that $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{y}$, in an appropriate sense. It is also NP-hard to find an $\boldsymbol{x}$ whose number of nonzero entries is within a constant factor of the smallest possible! Based on our current understanding of complexity theory, it is extraordinarily unlikely that anyone will ever discover an efficient algorithm that solves any interesting variant of the ℓ^0 minimization problem for all possible inputs $(\boldsymbol{A}, \boldsymbol{y})$.

2.3 Relaxing the Sparse Recovery Problem

The rather bleak worst-case picture for ℓ^0-minimization has not stopped engineers from searching for efficient heuristics for finding sparse solutions to linear systems. There is always some possibility for optimism:

> "Although the *worst* sparse recovery problem may be impossible to solve efficiently, perhaps my *particular* instance of interest is not so hard."

This optimism is occasionally rewarded in a rather striking fashion. In the next few chapters, we will see that many sparse recovery problems that matter for engineering practice *are* solvable efficiently. Our first step is to find a proper replacement for the ℓ^0 norm which still encourages sparsity, but can be optimized efficiently.

2.3.1 Convex Functions

If our goal is efficient optimization, perhaps the most natural class of objective functions to consider is the *convex* functions. Smooth convex functions often appear "bowl shaped" – as in Figure 2.9 (left). Indeed, a necessary and sufficient condition for a smooth function $f : \mathbb{R} \to \mathbb{R}$ to be convex is that it exhibit nonnegative curvature – its second derivative $\frac{d^2 f}{dx^2}(x) \geq 0$ at every point x.

Iterative methods for optimization seek a minimizer of an objective function $f(\boldsymbol{x})$, by starting from some initial point $\boldsymbol{x}_0$,[11] and then generating a new point $\boldsymbol{x}_1$ based on the local shape of the objective function in the vicinity of $\boldsymbol{x}_0$. For a smooth function $f(\boldsymbol{x})$, the negative gradient $-\nabla f(\boldsymbol{x})$ defines the direction in which the objective function decreases most rapidly. A natural strategy for choosing $\boldsymbol{x}_1$ is to set

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - t\nabla f(\boldsymbol{x}_0). \tag{2.3.1}$$

where t is a step size. Continuing in this manner to produce points $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, we obtain the *gradient descent* method, a natural and intuitive algorithm for minimizing a smooth function $f(\boldsymbol{x})$. For the function f in Figure 2.9 (left), assuming we choose the step size t appropriately, the iterates $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ will converge to the global minimizer $\boldsymbol{x}_\star$. For the nonconvex function to the right, this strategy only guarantees a local minimizer.

Convex functions such as Figure 2.9 (left) have the property that every local minimizer is a global minimizer.[12] Moreover, many convex functions

[11] In this book, we will use $\boldsymbol{x}_0$ to indicate the initial point of an iterative algorithm, which is not to be confused with $\boldsymbol{x}_o$, the desired ground truth.

[12] It is worth noting that for many of the problems we will later discuss (e.g., MRI, spectrum sensing, face recognition), global optimality is very important – there is a *true*

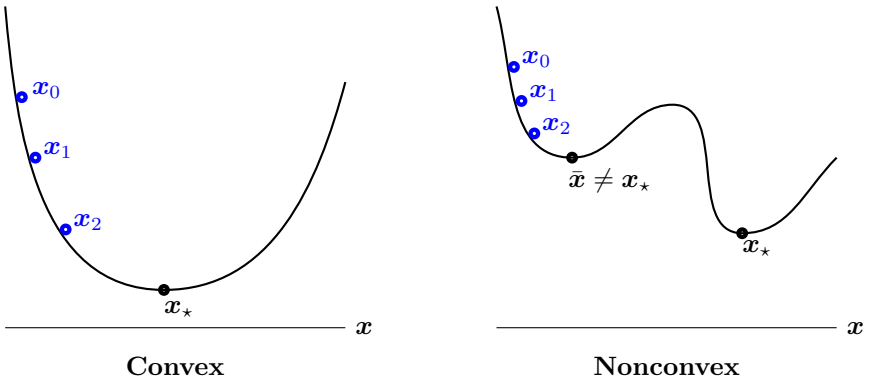


Figure 2.9. **Convex and nonconvex functions.** Left: a convex function. Local descent methods such as gradient descent produce a sequence of points $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ which approach the global minimizer $\boldsymbol{x}_\star$. Right: A nonconvex function. For this particular function, depending on the initial point $\boldsymbol{x}_0$, local descent methods may produce the suboptimal local minimum $\bar{\boldsymbol{x}}$. Motivated by their good properties for optimization, in the first part of this book, we will seek convex formulations for recovering sparse (and otherwise structured) signals.

arising in practice can be optimized efficiently using variants of gradient descent. Indeed, in Chapter 8, we will see that the particular convex functions that we encounter in computing with sparse signals (and their generalizations) *can* be efficiently optimized, even on a large scale and in high dimensions.

We review the properties of convex functions more formally in Appendix C. Here, we briefly remind the reader of the general definition of convex function:[13]

Definition 2.3.1 (Convex function on $\mathbb{R}^n$)**.** *A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for every pair of points $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$ and $\alpha \in [0, 1]$,*

$$f\Big(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{x}'\Big) \;\leq\; \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{x}'). \tag{2.3.2}$$

signal that we are trying to recover, and it is important to build algorithms that can do this reliably. In our simulated example of ℓ^0 minimization, we declared the solution $\hat{\boldsymbol{x}}$ correct, because it coincided with the true $\boldsymbol{x}_o$ that generated the observation $\boldsymbol{y}$. This is in contrast to some applications of optimization (e.g., in finance) where the objective function measures the goodness of the solution (say the expected rate of return on an investment), and locally improving the solution is meaningful, or even desirable, if the objective corresponds to dollars earned/lost!

[13] On the surface, this definition appears much more complicated than simply asking the second derivative to be positive. The reason for this complication is that we will need to work with convex functions that are not smooth; the general definition in Definition 2.3.1 handles this situation as well.

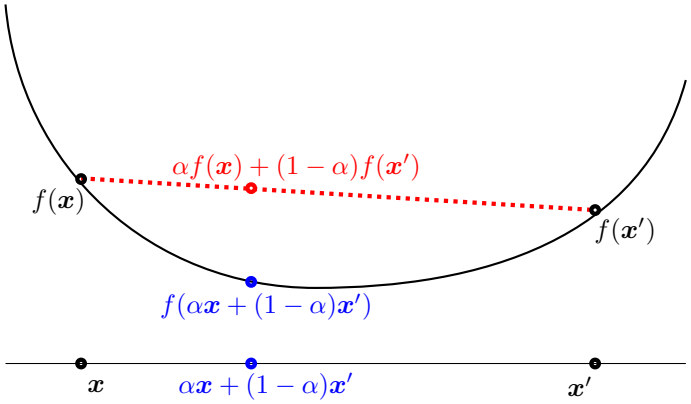


Figure 2.10. **Definition of convexity.** A convex function $f : \mathbb{R}^n \to \mathbb{R}$ is one which satisfies the inequality $f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{x}') \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{x}')$ for all $\alpha \in [0,1]$ and $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$. Geometrically, this means that if we take the points $(\boldsymbol{x}, f(\boldsymbol{x}))$ and $(\boldsymbol{x}', f(\boldsymbol{x}')$ on the graph of f, and then draw a **line** joining them, the graph of the function falls below this line segment.

This inequality can be visualized as follows. Consider two points $(\boldsymbol{x}, f(\boldsymbol{x}))$ and $(\boldsymbol{x}', f(\boldsymbol{x}'))$ on the graph of f. If we form the line segment joining these two points, this line segment lies above the graph of f. Figure 2.10 visualizes this inequality for several different convex functions.

A *convex combination* of a collection of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ is an expression of the form $\sum_{i=1}^k \lambda_i \boldsymbol{x}_i$, where the weights λ_i are nonnegative and $\sum_{i=1}^k \lambda_i = 1$. For example, for $\alpha \in [0,1]$, the expression $\boldsymbol{z} = \alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{x}'$ is a convex combination of the points $\boldsymbol{x}$ and $\boldsymbol{x}'$. The definition (2.3.2) states that at the point $\boldsymbol{z}$, the function f is no larger than the corresponding combination $\alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{x}')$ of the function values at the points $\boldsymbol{x}$ and $\boldsymbol{x}'$.

This property of convex functions generalizes, to give Jensen's inequality, which states that the value of a convex function f at a convex combination of points is no greater than the corresponding convex combination of the function values:

Proposition 2.3.2 (Jensen's inequality)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then for any k, any collection of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^n$ and any nonnegative scalars $\lambda_1, \ldots, \lambda_k$ satisfying $\sum_{i=1}^k \lambda_i = 1$,*

$$f\left(\sum_{i=1}^k \lambda_i \boldsymbol{x}_i\right) \leq \sum_{i=1}^k \lambda_i f(\boldsymbol{x}_i). \tag{2.3.3}$$

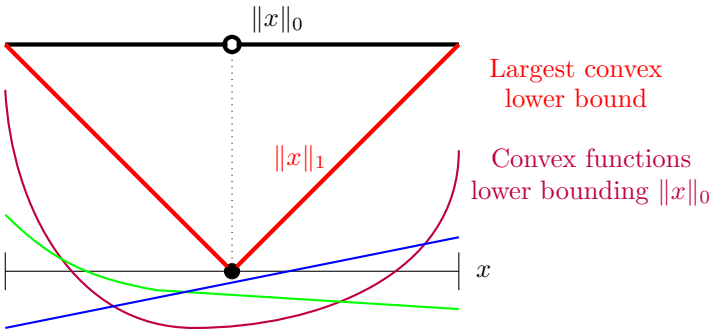


Figure 2.11. **A convex surrogate for the ℓ^0 norm.** In **black**, we plot the graph of the ℓ^0 norm of a scalar x, over the interval $x \in [-1, 1]$. This function takes on value 0 at $x = 0$, and $+1$ everywhere else. In **purple**, **green** and **blue**, we plot various convex function examples $f(x)$ which underestimate $\|x\|_0$ on $[-1, 1]$, in the sense that $f(x) \leq \|x\|_0$ for all $x \in [-1, 1]$. In **red**, we plot the function $f(x) = |x|$. This is the largest convex function which underestimates $\|x\|_0$ on $[-1, 1]$. We call $|x|$ the *convex envelope* of $\|x\|_0$ on $[-1, 1]$.

2.3.2 A Convex Surrogate for the ℓ^0 Norm: the ℓ^1 Norm

With the good properties of convex functions in mind, let us try to find a convex "surrogate" for the ℓ^0 norm. In one dimension, x is a scalar, and $\|x\|_0 = \mathbb{1}_{x \neq 0}$ is simply the indicator function for nonzero x. From Figure 2.11, it is clear that if we restrict our attention to the interval $x \in [-1, 1]$, the largest convex function which does not exceed $\|\cdot\|_0$ on this interval is simply the absolute value $|x|$. In the language of convex analysis, $|x|$ is the *convex envelope* of the function $\|x\|_0$ over the set $[-1, 1]$. This means that $|x|$ is the largest convex function f which satisfies $f(x) \leq \|x\|_0$ for every $x \in [-1, 1]$, i.e., it is the largest convex underestimator of $\|x\|_0$ over this set. Thus, in one dimension, we might consider the absolute value of x as a plausible replacement for $\|x\|_0$.

For higher-dimensional $\boldsymbol{x}$ (i.e., $\boldsymbol{x} \in \mathbb{R}^n$), the ℓ^0 norm is

$$\|\boldsymbol{x}\|_0 = \sum_{i=1}^{n} \mathbb{1}_{\boldsymbol{x}(i) \neq 0}. \tag{2.3.4}$$

Applying our above reasoning to each of the coordinates $\boldsymbol{x}(i)$, we obtain the ℓ^1 norm

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |\boldsymbol{x}(i)|. \tag{2.3.5}$$

As in the scalar case, this function is the tightest convex underestimator of $\|\cdot\|_0$, over an appropriate set of vectors $\boldsymbol{x}$:

Theorem 2.3.3. *The function $\|\cdot\|_1$ is the convex envelope of $\|\cdot\|_0$, over the set $\mathfrak{B}_\infty = \{\boldsymbol{x} \mid \|\boldsymbol{x}\|_\infty \leq 1\}$ of vectors whose elements all have magnitude at most one.*

Proof. Let f be a convex function satisfying $f \leq \|\cdot\|_0$ on $\mathfrak{B}_\infty$. We prove that $f \leq \|\cdot\|_1$ on $\mathfrak{B}_\infty$ as well. Consider the cube $C = [0,1]^n$. Its vertices are the vectors $\boldsymbol{\sigma} \in \{0,1\}^n$. Any $\boldsymbol{x} \in C$ can be written as a convex combination of these vertices:

$$\boldsymbol{x} = \sum_i \lambda_i \boldsymbol{\sigma}_i \tag{2.3.6}$$

Because $f \leq \|\cdot\|_0$, $f(\boldsymbol{\sigma}_i) \leq \|\boldsymbol{\sigma}_i\|_0 = \|\boldsymbol{\sigma}_i\|_1$. Because f is convex,

$$\begin{aligned} f(\boldsymbol{x}) &= f\left(\textstyle\sum_i \lambda_i \boldsymbol{\sigma}_i\right) \leq \sum_i \lambda_i f(\boldsymbol{\sigma}_i) && \text{[Jensen's inequality]} \\ &\leq \sum_i \lambda_i \|\boldsymbol{\sigma}_i\|_0 = \sum_i \lambda_i \|\boldsymbol{\sigma}_i\|_1 && [\boldsymbol{\sigma}_i \text{ are binary}] \\ &= \|\boldsymbol{x}\|_1 . \end{aligned} \tag{2.3.7}$$

Hence, $f \leq \|\cdot\|_1$ on the intersection of $\mathfrak{B}_\infty$ with the nonnegative orthant. Repeating the argument for each of the orthants, we obtain that $f \leq \|\cdot\|_1$ on $\mathfrak{B}_\infty$, and hence $\|\cdot\|_1$ is the convex envelope of $\|\cdot\|_0$ over $\mathfrak{B}_\infty$. □

So, at least in the sense of convex envelops, the ℓ^1 norm provides a good replacement for the ℓ^0 norm. Replacing the ℓ^0 norm in (2.2.9) with the ℓ^1 norm, we obtain a convex optimization problem,

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{x}\|_1 \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}. \end{array} \tag{2.3.8}$$

In contrast to the ℓ^0 problem, this problem *can* be solved efficiently.

2.3.3 A Simple Test of ℓ^1 Minimization

Theorem 2.3.3 is a strong initial motivation for considering (2.3.8) – it says that in a certain sense, the ℓ^1 norm is the canonical convex surrogate for the ℓ^0 norm. Some care is in order, though. Theorem 2.3.3 does not say anything at all about the *correctness* of (2.3.8) – whether the solution to (2.3.8) is actually the desired sparse vector $\boldsymbol{x}_o$.

The easiest way to get some insight into this question is to do an experiment! For this, we will need to solve the problem (2.3.8) computationally and see how well it works. How do we solve the optimization problem (2.3.8)? Appendix C gives a quick introduction to some general optimization techniques that may help us solve problems of this kind. More specifically, since the objective function is convex, the geometry of a convex function in Figure 2.12 (left) suggests that we should do quite well just using local information about the slope of the objective function. Indeed, if

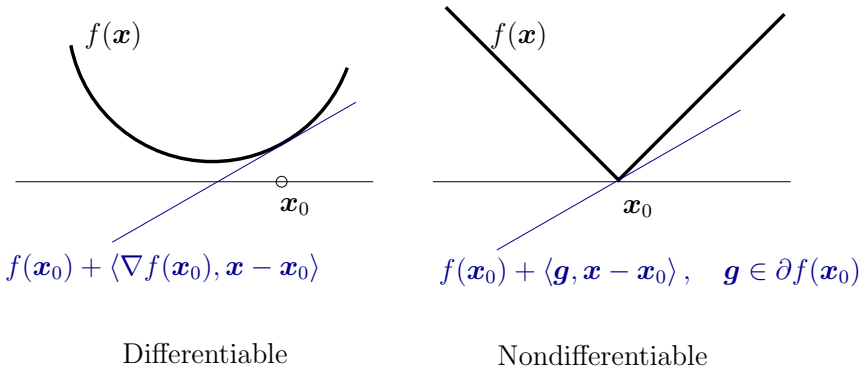


Figure 2.12. **Subgradients of convex functions.** Left: for a differentiable convex function, the best linear approximation at any point $\boldsymbol{x}_0$ is a *global* lower bound on the function. Right: for a nondifferentiable function, we say that $\boldsymbol{g}$ is a subgradient of f at $\boldsymbol{x}_0$ (and write $\boldsymbol{g} \in \partial f(\boldsymbol{x}_0)$ if $\boldsymbol{g}$ is the slope of a linear function that takes on value $f(\boldsymbol{x}_0)$ at $\boldsymbol{x}_0$, and globally lower bounds f.

our objective function were differentiable, this would very naturally suggest the classical *gradient descent* method for solving problems of the form

$$\text{minimize} \quad f(\boldsymbol{x}). \tag{2.3.9}$$

This algorithm starts at some initial point $\boldsymbol{x}_0$, and then generates a sequence of points $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \ldots)$ by iteratively moving in the direction of greatest decrease of $f(\cdot)$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t_k \nabla f(\boldsymbol{x}_k). \tag{2.3.10}$$

Here, $t_k \geq 0$ is a properly chosen step size.

There are two main difficulties that prevent us from applying the gradient descent iteration (2.3.10) to the ℓ^1 minimization problem (2.3.8):

- **Nontrivial constraints:** Unlike the general unconstrained problem (2.3.9), in the problem (2.3.8) we are only interested in $\boldsymbol{x}$ that satisfy $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$.
- **Nondifferentiable objective:** The objective function in (2.3.8) is not differentiable, and so at certain points the gradient does not exist. Figure 2.12 (right) shows this: the function is pointed at zero! Since zero is sparse, this is precisely one of the points we are most interested in.

Constraints.

One approach to handling the first problem is to replace the gradient descent iteration with *projected gradient descent*. This algorithm aims at

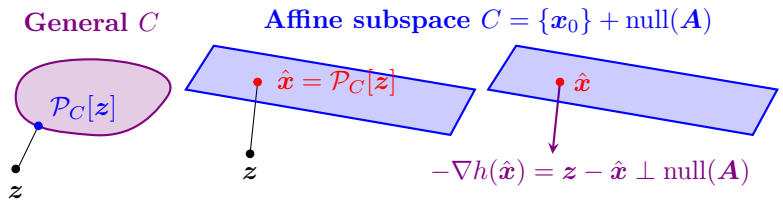


Figure 2.13. **Projection onto convex sets.** Left: projection onto a general convex set. Middle: projection onto an affine subspace. Right: the projection onto the affine subspace can be characterized as the point $\hat{\boldsymbol{x}}$ at which the gradient $\nabla h(\hat{\boldsymbol{x}})$ is orthogonal to $\text{null}(\boldsymbol{A})$.

general problems of the form

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in C, \end{array} \tag{2.3.11}$$

where C is some constraint set. This algorithm is exactly the same as gradient descent, except that at each iteration it *projects* the result $\boldsymbol{x}_k - t_k \nabla f(\boldsymbol{x}_k)$ onto the set C. The projection of a point $\boldsymbol{z}$ onto the set C is simply the nearest point to $\boldsymbol{z}$ in C:

$$\mathcal{P}_C[\boldsymbol{z}] = \arg\min_{\boldsymbol{x} \in C} \tfrac{1}{2} \|\boldsymbol{z} - \boldsymbol{x}\|_2^2 \equiv h(\boldsymbol{x}). \tag{2.3.12}$$

For general C, the projection may not exist, or may not be unique (think about how this could happen). However, for closed, convex sets, the projection is well-defined, and satisfies a wealth of useful properties. If $\boldsymbol{A}$ has full row rank, the projection onto the convex set $C = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}\}$ has an especially simple form:

$$\mathcal{P}_{\{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}\}}[\boldsymbol{z}] = \boldsymbol{z} - \boldsymbol{A}^* \left(\boldsymbol{A}\boldsymbol{A}^*\right)^{-1} \left[\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}\right]. \tag{2.3.13}$$

Figure 2.13 visualizes the projection onto this particular C. This formula can be derived by noting two properties of the projection $\hat{\boldsymbol{x}} = \mathcal{P}_C[\boldsymbol{z}]$:

- **(i) Feasibility**: $\hat{\boldsymbol{x}} \in C$, i.e., $\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{y}$.
- **(ii) Error is orthogonal**: $\boldsymbol{z} - \hat{\boldsymbol{x}} \perp \text{null}(\boldsymbol{A})$. Since $\boldsymbol{z} - \hat{\boldsymbol{x}} = -\nabla h\left(\hat{\boldsymbol{x}}\right)$, this condition can be stated as

  $$-\nabla h(\hat{\boldsymbol{x}}) \text{ is normal to } C \text{ at } \hat{\boldsymbol{x}}.$$

Exercise 2.9 guides the interested reader through the derivation of this expression. For the general problem (2.3.11), with differentiable objective f, the projected gradient algorithm simply repeats the iteration

$$\boldsymbol{x}_{k+1} = \mathcal{P}_C\left[\boldsymbol{x}_k - t_k \nabla f(\boldsymbol{x}_k)\right]. \tag{2.3.14}$$

Nondifferentiability.

The problem of nondifferentiability is slightly trickier. To handle it properly, we need to generalize the notion of derivative to include functions that are not differentiable. For this, we draw inspiration from geometry. Consider Figure 2.12 (left). It displays a convex, differentiable function $f(\boldsymbol{x})$, as well as a linear approximation $\hat{f}(\boldsymbol{x})$, taken at a point $\boldsymbol{x}_0$:

$$\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \langle \nabla f(\boldsymbol{x}_0), \boldsymbol{x} - \boldsymbol{x}_0 \rangle \,. \tag{2.3.15}$$

The salient point here is that the graph of f lies entirely above the graph of the approximation $\hat{f}$:

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \langle \nabla f(\boldsymbol{x}_0), \boldsymbol{x} - \boldsymbol{x}_0 \rangle \quad \forall\, \boldsymbol{x} \in \mathbb{R}^n. \tag{2.3.16}$$

It is not too difficult to prove that this property holds for *every* convex differentiable function and every point $\boldsymbol{x}_0$, simply by using calculus and the definition of convexity.

This geometry opens the door for generalizing the notion of the gradient to nonsmooth functions. For nonsmooth functions such as $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$, at a point of nonsmoothness $\boldsymbol{x}_0$, the gradient does not exist, but we can still make a linear under-estimator

$$\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \langle \boldsymbol{u}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle \,, \tag{2.3.17}$$

as in Figure 2.12 (right). Here, $\boldsymbol{u}$ replaces ∇f in the previous expression, and plays the role of the "slope" of the approximation. We say that $\boldsymbol{u}$ is a *subgradient* of f at $\boldsymbol{x}_0$ if the linear approximation defined by $\boldsymbol{u}$ is indeed an under-estimator of f (i.e., it lower bounds $f(\boldsymbol{x})$ at all points $\boldsymbol{x}$):

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \langle \boldsymbol{u}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle \,, \qquad \forall\, \boldsymbol{x}. \tag{2.3.18}$$

Let us consider our function of interest – the ℓ^1 norm. For $\boldsymbol{x} \in \mathbb{R}$ (one dimension), $\|\boldsymbol{x}\|_1 = |x|$ is simply the absolute value. For $x < 0$, the slope of the graph of $|x|$ is -1, while for $|x| > 0$, it is $+1$. Convince yourself that if we take $\boldsymbol{x}_0 \neq 0$, then the only u satisfying the above definition is $u = \text{sign}(x)$.

However, at 0 the function $|x|$ is "pointy," namely, nondifferentiable, and something different happens: at $x_0 = 0$, every $u \in [-1, 1]$ defines a linear approximation that underestimates f. So, in fact, every $u \in [-1, 1]$ is a subgradient. Thus, at points of nondifferentiability there may exist multiple subgradients. We call the collection of all subgradients of f at a point $\boldsymbol{x}_0$ the *subdifferential* of f at $\boldsymbol{x}_0$, and denote it by $\partial f(\boldsymbol{x}_0)$. Formally:

Definition 2.3.4 (Subgradient and subdifferential)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. A* subgradient *of f at $\boldsymbol{x}_0$ is any $\boldsymbol{u}$ satisfying*

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \langle \boldsymbol{u}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle \,, \quad \forall\, \boldsymbol{x}. \tag{2.3.19}$$

The subdifferential *of f at $\boldsymbol{x}_0$ is the set of all subgradients of f at $\boldsymbol{x}_0$:*

$$\partial f(\boldsymbol{x}_0) = \{\boldsymbol{u} \mid \forall\, \boldsymbol{x} \in \mathbb{R}^n,\ f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + \langle \boldsymbol{u}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle\} \,. \tag{2.3.20}$$

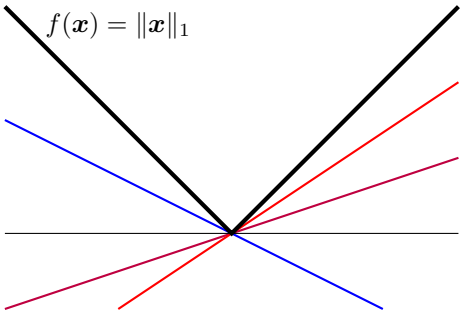


Figure 2.14. **Subdifferential of the ℓ^1 norm.** In **black**, $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$. In **blue**, **purple**, and **red**, three linear lower bounds of the form $g(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \langle \boldsymbol{u}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle$, taken at $\boldsymbol{x}_0 = \boldsymbol{0}$, with slope $\boldsymbol{u} = -\frac{1}{2}$, $\frac{1}{3}$, and $\frac{2}{3}$, respectively. It should be clear that any slope $\boldsymbol{u} \in [-1, 1]$ defines a linear lower bound on $f(\boldsymbol{x})$ around $\boldsymbol{x}_0 = \boldsymbol{0}$. So, $\partial|\cdot|(0) = [-1, 1]$. For $\boldsymbol{x}_0 > 0$, the only linear lower bound has slope $\boldsymbol{u} = 1$; for $\boldsymbol{x}_0 < 0$, the only linear lower bound has slope $\boldsymbol{u} = -1$. So, $\partial|\cdot|(x) = \{-1\}$ for $x < 0$ and $\partial|\cdot|(x) = \{1\}$ for $x > 0$. Lemma 2.3.5 proves this formally, and extends to higher-dimensional $\boldsymbol{x} \in \mathbb{R}^n$.

With these definitions in mind, we might imagine that in the nonsmooth case, a suitable replacement for the gradient algorithm might be the *subgradient method*, which chooses (somehow) $\boldsymbol{g}_k \in \partial f(\boldsymbol{x}_k)$, and then proceeds in the direction of $-\boldsymbol{g}_k$: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t_k \boldsymbol{g}_k$. Incorporating projection onto the feasible set, we arrive at the following *projected subgradient algorithm*:

$$\boldsymbol{x}_{k+1} = \mathcal{P}_{\mathcal{C}}[\boldsymbol{x}_k - t_k \boldsymbol{g}_k], \quad \boldsymbol{g}_k \in \partial f(\boldsymbol{x}_k). \tag{2.3.21}$$

To apply the projected subgradient method, we need an expression for the subdifferential of the ℓ^1 norm. Figure 2.14 visualizes this. In one dimension, $\|\boldsymbol{x}\|_1 = |x|$; this function is differentiable away from $x = 0$. For $x > 0$, $\partial|\cdot|(x) = \{1\}$, while for $x < 0$, $\partial|\cdot|(x) = \{-1\}$. At $x = 0$, $|x|$ is not differentiable, and there are multiple possible linear lower bounds. Figure 2.14 visualizes three of these lower bounds. It is not difficult to see that lower bounds at $x = 0$ can have any slope from -1 to 1; hence, $\partial|\cdot|(x) = [-1, 1]$. The following lemma extends this observation to higher-dimensional $\boldsymbol{x} \in \mathbb{R}^n$:

Lemma 2.3.5 (Subdifferential of $\|\cdot\|_1$)**.** *Let $\boldsymbol{x} \in \mathbb{R}^n$, with $I = \operatorname{supp}(\boldsymbol{x})$,*

$$\partial \|\cdot\|_1 (\boldsymbol{x}) = \{\boldsymbol{v} \in \mathbb{R}^n \mid \boldsymbol{P}_I \boldsymbol{v} = \operatorname{sign}(\boldsymbol{x}),\ \|\boldsymbol{v}\|_\infty \leq 1\}. \tag{2.3.22}$$

Here, $\boldsymbol{P}_I \in \mathbb{R}^{n \times n}$ is the orthoprojector onto coordinates I:

$$[\boldsymbol{P}_I \boldsymbol{v}](j) = \begin{cases} \boldsymbol{v}_j & j \in I \\ 0 & j \notin I \end{cases}. \tag{2.3.23}$$

Proof. The subdifferential $\partial \|\cdot\|_1 (\boldsymbol{x})$ consists of all vectors $\boldsymbol{v}$ that satisfy

$$\sum_{i=1}^{n} |\boldsymbol{x}'(i)| \;\geq\; \sum_{i=1}^{n} |\boldsymbol{x}(i)| + \boldsymbol{v}(i)\,(\boldsymbol{x}'(i) - \boldsymbol{x}(i)) \tag{2.3.24}$$

for every $\boldsymbol{x}$ and $\boldsymbol{x}'$. A sufficient condition is that for every index i and every scalar z,

$$|z| \;\geq\; |\boldsymbol{x}(i)| + \boldsymbol{v}(i)(z - \boldsymbol{x}(i)). \tag{2.3.25}$$

Taking $\boldsymbol{x}' = \boldsymbol{x} + (z - \boldsymbol{x}(i))\boldsymbol{e}_i$ in (2.3.24) shows that (2.3.25) is also necessary. If $\boldsymbol{x}(i) = 0$, (2.3.25) becomes $|z| \geq \boldsymbol{v}(i) z$, which holds for all z if and only if $|\boldsymbol{v}(i)| \leq 1$. If $\boldsymbol{x}(i) \neq 0$, the inequality is satisfied if and only if $\boldsymbol{v}(i) = \operatorname{sign}(\boldsymbol{x}(i))$. Hence, $\boldsymbol{v} \in \partial \|\cdot\|_1$ if and only if for all $i \in I$, $\boldsymbol{v}(i) = \operatorname{sign}(\boldsymbol{x}(i))$, and for all i, $|\boldsymbol{v}(i)| \leq 1$. This conclusion is summarized as (2.3.22). □

The projected subgradient method alternates between subgradient steps, which move in the direction of $-\operatorname{sign}(\boldsymbol{x})$, and orthogonal projections onto the feasible set $\{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}\}$ according to equation (2.3.13). We obtain a very simple algorithm that solves (2.3.8), which we spell out in detail as Algorithm 2.2.

Algorithm 2.2: ℓ^1-minimization by projected subgradient

Input: a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and a vector $\boldsymbol{y} \in \mathbb{R}^m$.
Compute $\boldsymbol{\Gamma} \leftarrow \boldsymbol{I} - \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{A}^*)^{-1}\boldsymbol{A}$, and $\tilde{\boldsymbol{x}} \leftarrow \boldsymbol{A}^\dagger \boldsymbol{y} = \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{A}^*)^{-1}\boldsymbol{y}$.
$\boldsymbol{x}_0 \leftarrow \boldsymbol{0}$.
$t \leftarrow 0$.
repeat many times
    $t \leftarrow t + 1$.
    $\boldsymbol{x}_t \leftarrow \tilde{\boldsymbol{x}} + \boldsymbol{\Gamma}\left(\, \boldsymbol{x}_{t-1} - \frac{1}{t}\operatorname{sign}(\boldsymbol{x}_{t-1}) \,\right)$.
end while

Remark 2.3.6 (Projected Subgradient and Better Alternatives)**.** *In many respects, this is a bad method for solving the ℓ^1 problem. It is correct, but it converges very slowly compared to methods that exploit a certain piece of problem-specific structure, which we will describe in later chapters. The main virtue of Algorithm 2.2 is that it is simple and intuitive, and also serves our exposition by introducing or reminding us of subgradients and projection operators.*[14] *The projected subgradient method for ℓ^1 minimization can be implemented in just a few lines of Matlab code. In the later*

[14] Also, we would like you to have a feel for at least one *very* simple way for implementing ℓ^1 minimization in code and play with it. Our experience is that this helps to think more concretely about the optimization problem and its applications, rather than leaving it as a mathematical abstraction.

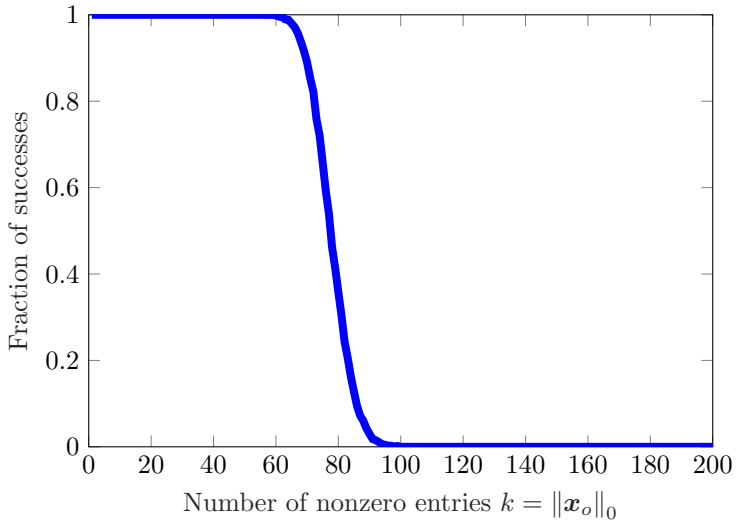


Figure 2.15. **Phase transition in ℓ^1 minimization.** We consider the problem of recovering a sparse vector $\boldsymbol{x}_o$ from measurements $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_o$, where $\boldsymbol{A} \in \mathbb{R}^{100\times 200}$ is a Gaussian matrix. We vary the number of nonzero entries $k = \|\boldsymbol{x}_o\|_0$ across $k = 0, 1, \ldots, 200$, and plot the fraction of instances where ℓ^1 minimization successfully recovers $\boldsymbol{x}_o$, over 50 independent experiments for each value of k. Notice that this probability of success exhibits a (rather sharp) transition from 1 (guaranteed success) to 0 (guaranteed failure) as k increases. Notice moreover, that *for sufficiently well-structured problems (k small), ℓ^1 minimization always succeeds.*

chapters, we will see a number of more sophisticated and better-performing approaches that better utilize the structure in this problem.

To see how well ℓ^1 minimization (as implemented through the projected subgradient method) performs, run `Chapter_2_L1_recovery.m`. You may see an interesting phenomenon! Although the method does not *always* succeed, it *does* succeed whenever the target solution $\boldsymbol{x}_o$ is *sufficiently sparse*! Figure 2.15 illustrates this in a more systematic way. In the figure, we generate random matrices $\boldsymbol{A}$ of size 200×400 and random vectors $\boldsymbol{x}_o$ with k nonzero entries. We vary k from 1 to 200. For each k, we run 50 experiments and plot the fraction of trials in which that ℓ^1 minimization correctly recovers $\boldsymbol{x}_o$, up to numerical error. Notice that indeed, ℓ^1 minimization succeeds whenever $\boldsymbol{x}_o$ is sufficiently sparse.

Given the examples we have seen thus far of how sparsity arises in application problems, this phenomenon is certainly intriguing. In the coming chapters, we will study it first from a mathematical perspective, to understand *why* it occurs and what its limitations are; we will then investigate its implications for practical applications.

2.4 Summary

Let us briefly recap what we have learned in this chapter. In many modern data analysis and signal processing applications, we need to solve very large, underdetermined systems of linear equations:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \quad \boldsymbol{A} \in \mathbb{R}^{m \times n},\ m < n.$$

Such problems are inherently ill-posed: they admit infinitely many solutions.

To make such problems well-posed, or to make the solution unique, we need to leverage additional properties of the solution that we wish to recover. One important property, which obtains in many practical applications, is sparsity (or compressibility). This is a powerful piece of information: although the signals themselves reside in a very high-dimensional space, they have only a few intrinsic degrees of freedom – they can be represented as a linear superposition of just a few atoms from a properly chosen dictionary. As Theorem 2.2.6 shows, under fairly general conditions, imposing sparsity on $\boldsymbol{x}$ can indeed make the problem of solving

$$\min \|\boldsymbol{x}\|_0 \quad \text{subject to} \quad \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

well conditioned: As long as the target solution $\boldsymbol{x}_o$ is sufficiently sparse w.r.t. the Kruskal rank of $\boldsymbol{A}$, the sparsest solution to $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ is unique and is the correct solution.

Computationally, however, finding the sparsest solution to a linear system is in general intractable (Theorem 2.2.8). To alleviate the computational difficulty, we relax the ℓ^0 minimization problem and replace the ℓ^0 norm of $\boldsymbol{x}$ with its convex envelope, the ℓ^1 norm:

$$\min \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}.$$

We have introduced a very basic subgradient descent algorithm (Algorithm 2.2) that solves the convex ℓ^1 minimization problem. From the results of the algorithm, we observe a striking phenomenon that ℓ^1 minimization can effectively recover the sparse solution under fairly broad conditions. We will explain why this is the case in the next chapter after we carefully characterize exact conditions under which ℓ^1 minimization gives the correct sparse solution.

2.5 Notes and References

Sparsity.

Historically, the notion of sparsity has arisen in many different guises, across different fields and applications.

Statistics – in statistics, sparsity arises in the context of *variable selection*. For example, in medical research, we may wish to select a few genes (predictors) whose expression levels predict the presence or absence of a certain type of disease. It is often important that we select a small (even minimal) set of predictors, so that the model we fit can be interpreted and further studied by humans. Classical variable selection methods, such as *stepwise regression* [Efroymson, 1960] are closely related to greedy methods for ℓ^0 minimization – see Chapter 8 for more on these methods. In recent years, ℓ^1 minimization (LASSO) has been widely used for variable selection, thanks to the seminal work of Tibshirani [Tibshirani, 1996]. We will examine the properties of this method in depth, beginning in the next chapter. Sparsity is also related to the principle of minimum description length [Rissanen, 1978], which argues that in choosing between various models, we should prefer models which can be encoded efficiently – see, e.g., [Hansen and Yu, 2001].

Error correction and robustness – the robust face recognition problem discussed in Section 2.1.3 illustrates the tight connection between sparsity and error correction. See Exercise 2.10 and the work of Candes and Tao [Candès and Tao, 2005a] for a more general view on this connection. Robustness to erroneous or outlying observations is an important issue for statistical inference [Huber, 1981, Hampel et al., 1986], and for communications, where we wish to ensure reliable communications over unreliable channels. The first uses of ℓ^1 minimization that we are aware of arise in the context of statistical estimation with errors, in the work of Boscovitch [Boscovich, 1750] and later Laplace [Laplace, 1774].

Neuroscience – the human brain is believed to encode sensory phenomena in a sparse fashion, in which only a small fraction of the available neurons are active at any given time (see, e.g., Field [Field, 1987]). Seminal work of Olshausen and Field [Olshausen and Field, 1997] observes that when applied to large families of natural images, optimization methods that pursue a dictionary which represents the images as *sparsely* as possible naturally produce dictionaries that reproduce qualitative properties of the receptive fields in the human visual cortex. The problem of producing a good dictionary for sparsely encoding a data set is known as *dictionary learning*. We discuss algorithmic aspects of this problem in Chapters 9 and 15. For a contemporary discussion of the neuroscientific implications of sparsity, we refer the interested reader to Ganguli and Sompolinsky [Ganguli and Sompolinsky, 2012] and the references therein.

Data compression – there is a natural connection between sparsity and data compression.

Geophysics – amongst the earliest applications of sparse recovery are problems in geophysical exploration.

Application vignettes.

The three applications described in this chapter illustrate various aspects of sparse modeling and sparse recovery. The medical imaging application is described in the work of Lustig et. al. [Lustig et al., 2007b, Lustig et al., 2008a]. The denoising results shown in Section 2.1.2 are due to Mairal et. al. [Mairal et al., 2008a]. The face recognition formulation in Section 2.1.3 is described in [Wright et al., 2009]. The discussion in this chapter only touches the surface of these problems; we will revisit medical imaging in Chapter 11, dictionaries and image processing in Chapter 15, and face recognition in Chapter 10. Please see these chapters and their references for broader context and related work on each of these problems. These are just a few of the vast array of applications of sparse methods; a few of these are highlighted in Chapters 12 - 16.

NP hardness of ℓ^0 minimization and related problems.

The hardness result for ℓ^0 minimization, Theorem 2.2.8, is due to Natarajan [Natarajan, 1995]; see also Davis, Mallat and Avellaneda [Davis et al., 1997]. Results of Amaldi and Kann [Amaldi and Kann, 1995, Amaldi and Kann, 1998] and Arora, Babai, Stern and Sweedyk [Arora et al., 1993] show that that ℓ^0 minimization problems are also NP-hard to approximate. Delineating the boundaries between tractable and intractable instances of sparse approximation remains an active topic of research: see, e.g., Zhang, Wainwright and Jordan [Zhang et al., 2014] and Foster, Karloff and Thaler [Foster et al., 2015] for more recent results. There are hardness results for a number of problems that relate closely to sparse approximation. The above results also have implications for sparse error correction. There are also hardness results around the problem of *matrix sparsification* in numerical analysis, which seeks to replace a given matrix $\boldsymbol{A}$ with a sparse matrix $\hat{\boldsymbol{A}}$ such that range($\boldsymbol{A}$) $\approx$ range($\hat{\boldsymbol{A}}$): see McCormick [McCormick, 1983], Coleman and Pothen [Coleman and Pothen, 1986], and Gottlieb and Neylon [Gottlieb and Neylon,] for discussions of the hardness of this and related problems.

Exercises

2.1 (Convexity of ℓ^p norms)**.** *Show that*

$$\|\boldsymbol{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p} \tag{2.5.1}$$

is convex for $p \geq 1$, and nonconvex for $0 < p < 1$.

2.2 (Relationship between ℓ^p norms)**.** *Show that for $p < p'$,*

$$\|\boldsymbol{x}\|_p \geq \|\boldsymbol{x}\|_{p'} \tag{2.5.2}$$

for every $\boldsymbol{x}$. For what $\boldsymbol{x}$ is equality obtained (i.e., $\|\boldsymbol{x}\|_p = \|\boldsymbol{x}\|_{p'}$)?

2.3 (Computing the Kruskal rank)**.** *Write a Matlab function that takes as an input a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, and outputs the Kruskal rank* $\operatorname{krank}(\boldsymbol{A})$*. There is no known way to efficiently compute the Kruskal rank. It is fine if your code takes time exponential in n. Corroborate the conclusion of Theorem 2.2.6, by generating a 4×8 Gaussian matrix $\boldsymbol{A}$, via* `A = randn(4,8);`*, and computing its Kruskal rank.*

2.4 (A structured matrix with small Kruskal rank)**.** *Consider a 4×8 dimensional complex matrix generated as*

$$\boldsymbol{A} = [\, \boldsymbol{I} \mid \boldsymbol{F} \,], \tag{2.5.3}$$

where $\boldsymbol{I}$ is the 4×4 identity matrix, and $\boldsymbol{F}$ is a 4×4 Discrete Fourier Transform (DFT) matrix: in Matlab, `A = [ eye(4), dftmtx(4) ]`*. Either using your code from Exercise 2.3, or hand calculations, determine the Kruskal rank of $\boldsymbol{A}$. You should find that it is smaller than 4! A general version of this phenomenon can be observed with the* Dirac comb*, which is sparse in both time and frequency.*

2.5 (The spark)**.** *Results on ℓ^0 uniqueness are sometimes described in terms of the* spark *of a matrix, which is the number of nonzero entries in the sparsest nonzero element of the null space of $\boldsymbol{A}$:*

$$\operatorname{spark}(\boldsymbol{A}) \quad = \quad \min_{\boldsymbol{d}\neq\boldsymbol{0},\ \boldsymbol{A}\boldsymbol{d}=\boldsymbol{0}} \|\boldsymbol{d}\|_0.$$

What is the relationship between $\operatorname{spark}(\boldsymbol{A})$ *and* $\operatorname{krank}(\boldsymbol{A})$*?*

2.6 (Kruskal rank of random matrices)**.** *In this exercise we prove that for a generic $m \times n$ matrix $\boldsymbol{A} \sim_{\text{iid}} \mathcal{N}(0,1)$,* $\operatorname{krank}(\boldsymbol{A}) = 1$ *with probability one.*

(i) Argue that for any *$m \times n$ matrix $\boldsymbol{A}$,* $\operatorname{krank}(\boldsymbol{A}) \leq m$*.*

(ii) Let $\boldsymbol{A} = [\boldsymbol{a}_1 \mid \ldots \boldsymbol{a}_n]$. Let span *denote the linear span of a collection of vectors. Argue that*

$$\mathbb{P}\left[\boldsymbol{a}_m \in \operatorname{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{m-1})\right] = 0. \tag{2.5.4}$$

(iii) Argue that $\operatorname{krank}(\boldsymbol{A}) < m$ *if and only if there exist some indices $i_1, \ldots, i_m$ such that*

$$\boldsymbol{a}_{i_m} \in \operatorname{span}(\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m-1}) \tag{2.5.5}$$

(iv) Conclude that $\operatorname{krank}(\boldsymbol{A}) = m$ *with probability one, by noting that*

$$\begin{aligned}
&\mathbb{P}\left[\exists i_1, \ldots, i_m \ : \ \boldsymbol{a}_{i_m} \in \operatorname{span}(\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_{m-1}})\right] \\
&\quad \leq \sum_{i_1, \ldots, i_m} \mathbb{P}\left[\boldsymbol{a}_{i_m} \in \operatorname{span}(\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_{m-1}})\right] \\
&\quad \leq m^n \times \underbrace{\mathbb{P}\left[\boldsymbol{a}_m \in \operatorname{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{m-1})\right]}_{=0} \\
&\quad = 0.
\end{aligned}$$

2.7 (ℓ^0 minimization and typical examples)**.** *We showed that there is a* worst case *phase transition in* ℓ^0 *minimization at* $\frac{\operatorname{krank}(\boldsymbol{A})}{2}$*. This means that* ℓ^0 *minimization recovers* every $\boldsymbol{x}_o$ *satisfying* $\|\boldsymbol{x}_o\|_0 < \frac{\operatorname{krank}(\boldsymbol{A})}{2}$*. We also know that for a Gaussian matrix* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\operatorname{krank}(\boldsymbol{A}) = m$*, with probability one.*

Using code for ℓ^0 *minimization provided via CourseWorks (or write your own!), please do the following: generate a* 5×12 *Gaussian matrix* `A = randn(5,12)`*. What is* $\operatorname{rank}(\boldsymbol{A})$*? Generate a sparse vector* $\boldsymbol{x}_o$*, with four nonzero entries, via* `xo = zeros(12,1); xo(1:4) = randn(4,1)`*. Now, set* `y = A xo;`*. Solve the* ℓ^0 *minimization problem, to find the sparsest vector* $\boldsymbol{x}$ *satisfying* $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$*. Is it the same as* $\boldsymbol{x}_o$*? Check that* `norm(x - xo)` *is small, where* `x` *is the solution produced by your code.*

Notice that *the worst case* theory for ℓ^0 predicts that we can only recover vectors with at most 2 nonzero entries. But we have observed ℓ^0 succeeding with 4 nonzero entries! This is an example of a *typical case* performance which is better than the worst case.

Please explain this! Argue that if $\boldsymbol{x}_o$ *is a fixed vector supported on some set* I *of size* $< m$*, then the probability that there exists a subset* $I' \neq I$ *of size* $< m$ *satisfying* $\boldsymbol{A}\boldsymbol{x}_o \in \operatorname{range}(\boldsymbol{A}_{I'})$ *is zero.*

Does your argument imply that the worst case theory based on rank *can be improved? Why or why not?*

2.8 (Subdifferentials)**.** *Compute the subdifferentials for the following functions:*

1. *The subdifferential for* $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_\infty$ *with* $\boldsymbol{x} \in \mathbb{R}^n$*.*
2. *The subdifferential for* $f(\boldsymbol{X}) = \sum_{j=1}^n \|\boldsymbol{X}\boldsymbol{e}_j\|_2$ *with* $\boldsymbol{X}$ *a matrix in* $\mathbb{R}^{n \times n}$*.*
3. *The subdifferential for* $f(\boldsymbol{x}) = \|\boldsymbol{X}\|_*$ *with* $\boldsymbol{X}$ *a matrix in* $\mathbb{R}^{n \times n}$*.*

2.9 (Projection onto an affine subspace)**.** *In deriving the projected subgradient method for* ℓ^1 *minimization, we used the fact that for an affine subspace*

$$C = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}\}, \tag{2.5.6}$$

where $\boldsymbol{A}$ *is a matrix with full row rank, and* $\boldsymbol{y} \in \mathrm{range}(\boldsymbol{A})$*, the Euclidean projection on* C *is given by*

$$\mathcal{P}_C[\boldsymbol{z}] = \arg\min_{\boldsymbol{A}\boldsymbol{x}=\boldsymbol{y}} \|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \tag{2.5.7}$$

$$= \boldsymbol{z} - \boldsymbol{A}^*\left(\boldsymbol{A}\boldsymbol{A}^*\right)^{-1}\left[\boldsymbol{A}\boldsymbol{z}-\boldsymbol{y}\right]. \tag{2.5.8}$$

Prove that this formula is correct. You may use the following geometric characterization of $\mathcal{P}_C[\boldsymbol{z}]$*:* $\boldsymbol{x} = \mathcal{P}_C[\boldsymbol{z}]$ *if and only if (i)* $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ *and (ii) for any* $\tilde{\boldsymbol{x}}$ *satisfying* $\boldsymbol{A}\tilde{\boldsymbol{x}} = \boldsymbol{y}$*, we have*

$$\langle \boldsymbol{z}-\boldsymbol{x}, \tilde{\boldsymbol{x}}-\boldsymbol{x}\rangle \le 0. \tag{2.5.9}$$

2.10 (Sparse error correction)**.** *In coding theory and statistics, we often encounter the following situation: we have an observation* $\boldsymbol{z}$*, which* should *be expressible as* $\boldsymbol{B}\boldsymbol{x}$*,* except that *some of the entries are corrupted. We can express our corrupted observation as*

$$\underset{\text{observation}}{\boldsymbol{z}} = \underset{\text{encoded message}}{\boldsymbol{B}\boldsymbol{x}} + \underset{\text{sparse corruption}}{\boldsymbol{e}} \tag{2.5.10}$$

Here $\boldsymbol{z} \in \mathbb{R}^n$ *is the observation* $\boldsymbol{x} \in \mathbb{R}^r$ *is a message of interest;* $\boldsymbol{B} \in \mathbb{R}^{n\times r}$ *(*$n > r$*) is a tall matrix with full column rank* r*, and* $\boldsymbol{e} \in \mathbb{R}^n$ *represents any corruption of the message. In many applications, the observation may be subject to corruption which is large in magnitude, but affects only a few of the observations, i.e.,* $\boldsymbol{e}$ *is sparse vector. Let* $\boldsymbol{A} \in \mathbb{R}^{n-r\times n}$ *be a matrix whose rows span the left null space of* $\boldsymbol{B}$*, i.e.,* $\mathrm{rank}(\boldsymbol{A}) = n-r$*, and* $\boldsymbol{A}\boldsymbol{B} = \boldsymbol{0}$*. Prove that for any* k*, (2.5.10) has a solution* $(\boldsymbol{x}, \boldsymbol{e})$ *with* $\|\boldsymbol{e}\|_0 = k$ *if and only if the* underdetermined system

$$\boldsymbol{A}\boldsymbol{e} = \boldsymbol{A}\boldsymbol{z} \tag{2.5.11}$$

has a solution $\boldsymbol{e}$ *with* $\|\boldsymbol{e}\|_0 = k$*. Argue that that the optimization problems*

$$\min_{\boldsymbol{x}} \|\boldsymbol{B}\boldsymbol{x}-\boldsymbol{z}\|_1 \tag{2.5.12}$$

and

$$\min_{\boldsymbol{e}} \|\boldsymbol{e}\|_1 \quad \textit{subject to} \quad \boldsymbol{A}\boldsymbol{e} = \boldsymbol{A}\boldsymbol{z} \tag{2.5.13}$$

are equivalent, in the sense that for every solution $\hat{\boldsymbol{x}}$ *of (2.5.12),* $\hat{\boldsymbol{e}} = \boldsymbol{B}\hat{\boldsymbol{x}}-\boldsymbol{z}$ *is a solution to (2.5.13); and for every solution* $\hat{\boldsymbol{e}}$ *of (2.5.13), there is a solution* $\hat{\boldsymbol{x}}$ *of (2.5.12) such that* $\hat{\boldsymbol{e}} = \boldsymbol{B}\hat{\boldsymbol{x}} - \boldsymbol{z}$*.*

It is sometimes observed that "sparse representation and sparse error correction are equivalent". In what sense is this true?

2.11 (ℓ^1 vs. ℓ^p minimization)**.** *We have studied the* ℓ^1 *minimization problem*

$$\min \|\boldsymbol{x}\|_1 \quad \textit{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \tag{2.5.14}$$

for recovering sparse $\boldsymbol{x}_o$*. We can obtain other convex optimization problems by replace* $\|\cdot\|_1$ *with* $\|\cdot\|_p$ *for* $p \in (1,\infty]$*. For what kind of* $\boldsymbol{x}_o$ *would you*

expect ℓ^∞ minimization to outperform ℓ^1 minimization (in the sense of recovering $\boldsymbol{x}_o$ more accurately)?

2.12 (Faces and linear subspaces)**.** *Download* `face_intro_demo.zip` *from the CourseWorks. Run* `load_eyb_recognition` *to load a collection of images under varying illumination into memory. The training images (under different lighting) will be stored in* `A_train`*, the identities of the subjects in* `label_train`*. Form a matrix* `B` *by selecting those columns of* `A_train` *that correspond to Subject 1. We will use the* singular value decomposition *to investigate how well-approximated the columns of* `B` *are by a linear subspace.*

Compute the singular values *of* `B` *using* `sigma = svd(B)`*. How many singular values r are needed to capture 95% of the energy of $\boldsymbol{B}$? I.e., to ensure that*

$$\sum_{i=1}^{r} \sigma_i^2 > .95 \times \sum_{i=1}^{n} \sigma_i^2 \ ? \tag{2.5.15}$$

What about 99% of the energy? Repeat this calculation for several subjects.

2.13 (Sparsity of MR images)**.**

2.14 (Sparsity of natural image patches)**.**

2.A Complexity Classes and NP-Hardness

If you don't have any background in complexity theory, you can loosely think of the situation as follows. The problem class P consists of the problems that we can solve in time polynomial in the size of the problem. The problem class NP consists of those problems for which, if we are given a "certificate" describing the optimal solution, we can check that it is correct in polynomial time. That is, P contains problems for which *finding* the right answer is "easy", while NP contains problems for which *checking* the right answer is easy. Anyone who has ever struggled with a problem for days, only to have a colleague or teacher easily demonstrate an obviously correct solution can appreciate the difference between finding the right answer and checking the right answer!

It turns out that amongst the NP problems, there are certain "NP complete" problems to which *every* problem in NP can be reduced, in polynomial time. So, solving one of these problems efficiently would enable you to solve every problem in NP efficiently! It is remarkable that this class of problems exists, and that it is quite large. It includes famous examples such as the Traveling Salesman Problem or the Multiway Cut Problem.

To understand the phrase "NP hard", we have to appreciate one technicality regarding the above definitions of P and NP: they pertain only to *decision* problems, in which the goal is to produce a YES/NO answer. For

example, the decision version of the Traveling Salesman Problem asks: "Is it possible to visit all of the nodes of a given graph (cities) while traveling a distance at most $d_\star$?" The decision version of the ℓ^0 problem asks: "Does the system $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ have a solution with at most k nonzero entries?"

Often in practice we care much more about *optimization problems* than *decision problems* – we do not just want to know whether a solution exists ... we want to know the way to find it! Optimization problems can't be "NP-complete" – in the formal definition of NP, we only include decision problems. We call an optimization problem NP-*hard* if an efficient solution to that problem can be used to efficiently solve NP-complete problems. For example, the optimization version of the Traveling Salesman Problem asks: "Find the shortest path that visits all of the nodes in a given graph." If I can solve this problem efficiently, I can clearly also solve the decision version efficiently, just by checking whether the optimal path that I found has length at most $d_\star$.