Lab 1

Compressed Sensing

Lab Objective: *Learn About Techniques in Compressed Sensing.*

One of the more important and fundamental problems in mathematics and science is solving a system of linear equations

$$Ax = b.$$

Depending on the properties of the matrix A (such as its dimensions and rank), there may be exactly one solution, infinitely many solutions, or no solution at all.

In the case where A is a square invertible matrix, there is of course one unique solution, given by $A^{-1}b$. There are various computational methods for inverting A, and we have studied many of them previously.

When b does not lie in the range of A, there is no exact solution. We can still hope to find approximate solutions, and techniques such as least squares and least absolute deviations provide ways to do this.

The final case is when there are infinitely many vectors x that satisfy $Ax = b$. How do we decide which vector to choose? A common approach is to choose a vector x of minimal norm satisfying $Ax = b$. This can be stated as an optimization problem:

$$\begin{aligned} \text{minimize} \quad & \|x\| \\ \text{subject to} \quad & Ax = b. \end{aligned}$$

When we use the standard Euclidean 2-norm, the problem is equivalent to the quadratic program

$$\begin{aligned} \text{minimize} \quad & x^T x \\ \text{subject to} \quad & Ax = b, \end{aligned}$$

which we can solve using an iterative procedure. Alternatively, the solution is given directly by $A^\dagger b$, where $A^\dagger$ is the Moore-Penrose pseudoinverse of A.

If instead of the 2-norm we use the 1-norm, our problem can be restated as a linear program, and solved efficiently using the Simplex Algorithm or an Interior

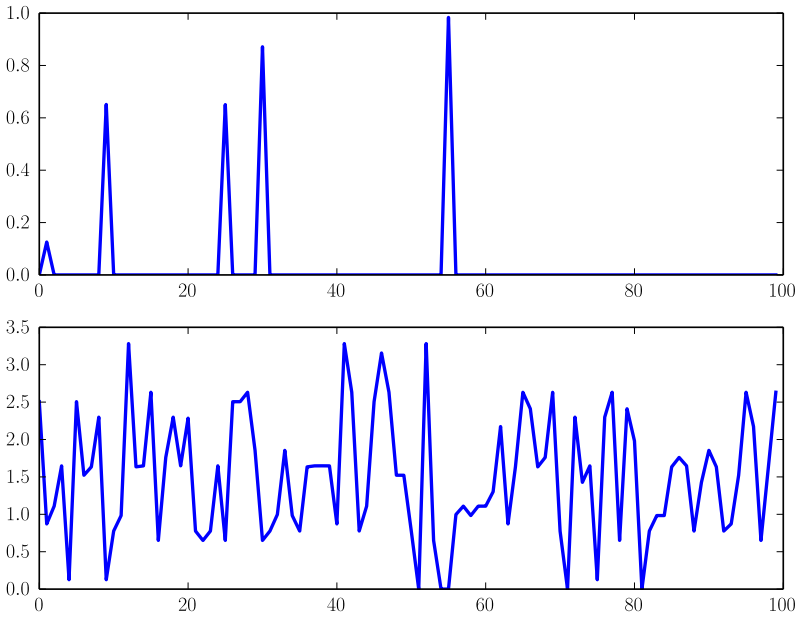


Figure 1.1: A sparse signal and a non-sparse signal.

Point method. Of course we can use any norm whatsoever, but finding the solution may be much more difficult.

The basic problem in the field of Compressed Sensing is to recover or reconstruct certain types of signals from a small set of measurements. For example, we might have measurements of the frequency spectrum of an audio signal, and we wish to recover the original audio signal as nearly as possible. Mathematically, this problem can be viewed as solving the under-determined system of equations $Ax = b$, where b is a vector of measurements and A is called the measurement matrix. The crucial idea that Compressed Sensing brings to the table is the concept of *sparsity*, which we address now.

Sparsity and the l_0 Pseudonorm

Sparsity is a property of vectors in $\mathbb{R}^n$ related to how compactly and concisely they can be represented in a given basis. Stated more concretely, the sparsity of a vector x (expressed in some given basis) refers to how many nonzero entries are in x. A vector having at most k nonzero entries is said to be k-sparse. This concept can be extended to time series (such as an audio signal) and to images. Figure 32.1 shows examples of both sparse and non-sparse signals.

As a convenient way of measuring sparsity, we define the so-called l_0 pseudonorm, notated $\|\cdot\|_0$, that simply counts the number of nonzero entries in a vector. For

example, if we have

$$x = \begin{bmatrix} 1 & 0 & 0 & -4 & 0 \end{bmatrix}^T$$

and

$$y = \begin{bmatrix} .5 & .2 & -.01 & 3 & 0 \end{bmatrix}^T,$$

we then have

$$\|x\|_0 = 2$$

and

$$\|y\|_0 = 4.$$

Despite our choice of notation, $\|\cdot\|_0$ is not truly a norm (which properties does it fail to satisfy?). Keep this in mind, even if we refer to it as the l_0 norm.

As mentioned earlier, sparsity is of central importance in Compressed Sensing, for it provides a way for us to select from the infinite possibilities a single vector x satisfying $Ax = b$. In particular, we require x to be as sparse as possible, i.e. to have minimal l_0 norm. Stated explicitly as an optimization problem, Compressed Sensing boils down

$$\begin{aligned} \text{minimize} \quad & \|x\|_0 \\ \text{subject to} \quad & Ax = b. \end{aligned}$$

Sparse Reconstruction

How does the Compressed Sensing framework laid out above help us recover a signal from a set of measurements? If we know nothing about the signal that we are trying to reconstruct, anything but a complete set of measurements (or *samples*) of the signal will be insufficient to fully recover the signal without any error.

However, we can often make certain assumptions about the unknown signal, such as setting an upper bound for its highest frequency. Given this prior knowledge about the frequency of the signal, we *are* able to perfectly or nearly perfectly reconstruct the signal from an incomplete set of measurements, provided that the sampling rate of the measurements is more than twice that of the largest frequency. This classic result in signal processing theory is known as the *Nyquist-Shannon sampling theorem*.

What if, instead of having prior knowledge about the frequency of the signal, we have prior knowledge about its sparsity? Recent research asserts that it is possible to recover sparse signals to great accuracy from just a few measurements. This matches intuition: sparse signals do not contain much information, and so it ought to be possible to recover that information from just a few samples. Stated more precisely, if $\hat{x} \in \mathbb{R}^n$ is sufficiently sparse and an $m \times n$ matrix A ($m < n$) satisfies certain properties (to be described later), with $A\hat{x} = b$, then the solution of the optimization problem

$$\begin{aligned} \text{minimize} \quad & \|x\|_0 \\ \text{subject to} \quad & Ax = b \end{aligned}$$

yields $\hat{x}$.

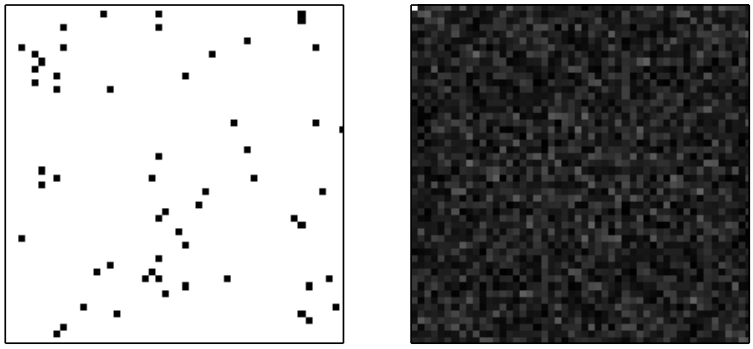


Figure 1.2: A sparse image (left) and its Fourier transform (right). The Fourier domain is incoherent with the standard domain, so the Fourier transform of the spare image is quite diffuse.

The matrix A above must satisfy a technical condition called the *Restricted Isometry Principle*. Most random matrices obtained from standard distributions (such as the Gaussian or Bernoulli distributions) satisfy this condition, as do transformation matrices into the Fourier and Wavelet domains. Generally speaking, the measurement matrix A represents a change of basis from the *sparse* basis to the *measurement* basis, followed by an under-sampling of the signal. The Restricted Isometry Principle guarantees that the measurement basis is incoherent with the sparse basis; that is, a sparse signal in the sparse basis is diffuse in the measurement basis, and vice versa (see Figure 1.2). This ensures that the information contained in the few nonzero coefficients in the sparse domain is spread out randomly and roughly evenly among the coefficients in the measurement domain. We can then obtain a small random subset of these measurement coefficients, solve the above optimization problem, and recover the original signal.

Fortunately, many types of signals that we wish to measure are sparse in some domain. For example, many images are sparse in the Wavelet domain. It turns out that the Fourier domain is incoherent with the Wavelet domain, so if we take measurements of the image in the Fourier domain, we are able to recover the image with high fidelity from relatively few measurements. This has been particularly useful in magnetic resonance imaging (MRI), a medical process that obtains pictures of tissues and organs in the body by taking measurements in the Fourier domain.

Collecting these measurements can in some cases be harmful. Compressed sensing allows for fewer measurements and therefore a shorter, safer MRI experience.

Solving the l_0 Minimization Problem

Now that we have become familiar with the mathematical underpinnings of compressed sensing, let us now turn to actually solving the problem. Unfortunately, the l_0 minimization problem stated above is NP hard and thus computationally intractable in its current form. Another key result in compressed sensing states that we can replace the l_0 norm with the l_1 norm and with high probability still recover the original signal, provided it is sufficiently sparse. Since the l_1 minimization problem can be solved efficiently, we have a viable computational approach to compressed sensing.

Recall that we can convert the l_1 minimization problem into a linear program by introducing an additional vector u of length n, and then solving

$$\begin{aligned} \text{minimize} \quad & \begin{bmatrix}\mathbf{1} & 0\end{bmatrix}\begin{bmatrix}u\\x\end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix}-I & I\\-I & -I\end{bmatrix}\begin{bmatrix}u\\x\end{bmatrix} \leq \begin{bmatrix}0\\0\end{bmatrix}, \\ & \begin{bmatrix}0 & A\end{bmatrix}\begin{bmatrix}u\\x\end{bmatrix} = b. \end{aligned}$$

Of course, solving this gives values for the optimal u and the optimal x, but we only care about the optimal x.

Problem 1. Write a function called `l1Min()` that takes a matrix A and vector b as inputs, and returns the solution to the optimization problem

$$\begin{aligned} \text{minimize} \quad & \|x\|_1 \\ \text{subject to} \quad & Ax = b. \end{aligned}$$

Formulate the problem as the above linear program and use CVXOPT to obtain the solution.

Hint: CVXOPT requires the entries of the matrices to be floats.

We will use our l_1 minimizer to reconstruct a sparse image using compressed sensing. Load the image contained in the file `ACME.png` into Python as follows:

```
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> acme = 1 - plt.imread('ACME.png')[:,:,0]
>>> print(acme.shape)
(32L, 32L)
```

The image contains 32^2 pixels, so viewed as a flat vector, it has 32^2 entries. Now we build a random sampling matrix to get m measurements, take those measurements, and reconstruct the image.

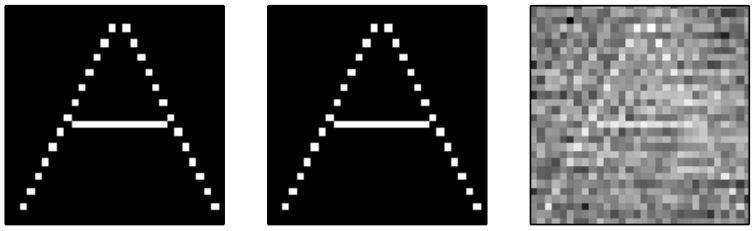


Figure 1.3: A sparse image (left), perfect reconstruction using l_1 minimization (middle), and imperfect reconstruction using l_2 minimization (right).

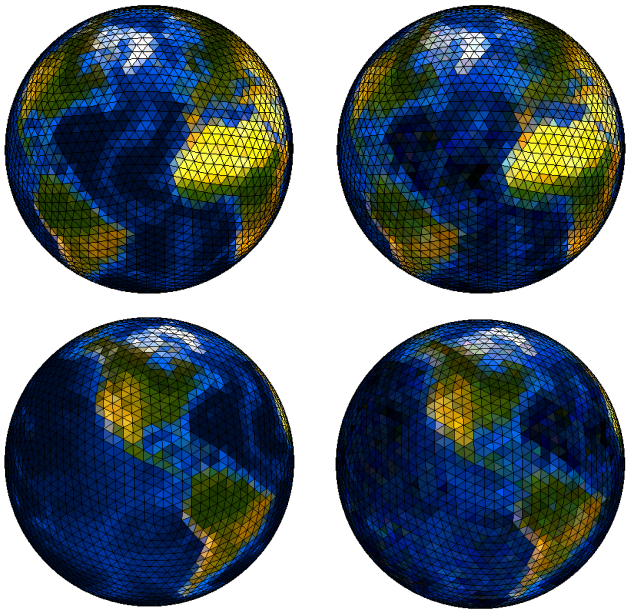
```
>>> m = 100                    # The number of measurements.
>>> np.random.seed(1337)       # Seed the random generator for consistency.

>>> A = np.random.randint(low=0, high=2, size=(m, 32**2))
>>> b = A.dot(acme.flatten())
>>> reconstruction = l1Min(A,b)
```

Problem 2. Following the example above, reconstruct the image using 100, 200, 250, and 275 measurements. Seed NumPy's random number generator with `np.random.seed(1337)` before each measurement to obtain consistent results.

Resize and plot each reconstruction in a single figure with several subplots (use `plt.imshow()` instead of `plt.plot()`). Return a list containing the Euclidean distance between each reconstruction and the original image.

Figure 1.3 shows the results of reconstructing a similar sparse image using both the 1-norm and the 2-norm.



(a) **(b)**

Figure 1.4: A 3-D model of the earth, with 5120 faces. (a) shows two views of the model drawn directly from satellite imagery. (b) shows two views of the reconstruction based upon 2500 single-pixel measurements.

Tesselated Surfaces and the Single Pixel Camera

We now generalize our discussion to reconstructing functions defined on surfaces. Such a function might give a color at each point on the surface, or the temperature, or some other property. The surface may be a sphere (to model the surface of the earth), a cube, or some other desired form. To make the problem tractable, we must break the surface up into a set of discrete polygonal faces, called a tessellation. See Figure 1.4 for an example of a tessellated sphere.

Now that we have a tessellation, our function can be represented by a vector, consisting of the function value for each facet (element of the tessellation). We are thus in a position to apply the techniques of compressed sensing to recover the function.

How do we go about acquiring color measurements on a tessellated surface? One cheap method is to use a *single-pixel camera*. Such a camera can only measure a single color value at a time. The camera points to a specific spot on the surface, and records a weighted average of the colors near that spot. By taking measurements at random spots across the surface, the single-pixel camera provides us with a vector of measurements, which we can then use to reconstruct the color value for each facet of the tessellation. See Figure 1.4 for the results of a single-pixel camera taking measurements of the earth.

The single-pixel camera measurement process can be modeled as matrix multiplication by a measurement matrix A, which fortunately has the Restricted Isometry

Property. In the file `camera.py`, we provide you with code to take single pixel measurements.

```
>>> from camera import Camera
>>> myCamera = Camera(faces, vertices, colors)
```

Where `faces`, `verticies` and `colors` are given by the tesselation.

To gather data with m measurements (taking m single-pixel pictures), do the following:

```
>>> m = 450
>>> myCamera.add_lots_pics(m)
>>> A, b = myCamera.returnData()
```

The matrix A is the sparse measurement matrix, and b contains the data to do compressed sensing on.

In this applications signals are only sparse in some appropriate representation (such as Fourier or wavelet). This method generally can still be applied in such cases. Let V represent the transformation under which s is sparse, or in other words:

$$s = Vp$$

In this case, V is the inverse Fourier. We can then recast $As = b$ as

$$AVp = b \tag{1.1}$$

This then allows us to find p using Compressed Sensing, which in turn allows us to reconstruct s by using V.

You must reconstruct the color functions for the tesselated surfaces, and then plot these surfaces using the code we provide you.

Problem 3. We will reconstruct the surface of the earth from sparse satellite imagery. The earth is modeled by a sphere, which can be tessellated into triangular faces, each having a single color value. The colors are sparse in each channel in the appropriate basis.

The file `StudentEarthData.npz` contains the faces, vertices, and colors, and the inverse fourier matrix, accessed by the keys `'faces'`, `'vertices'`, `'C'`, and `'V'`, respectively. Use the faces, vertices, and colors to initialize a camera object as in the example code above. Extract the data with the `returnData()` function. The resulting matrix A is the sparse measurement matrix, and b is an approximation of the array of colors. There are three channels in the color array, stored as columns. Do compressed sensing on each channel of colors, obtaining three arrays. Then stack these arrays column-wise so the result matches the dimensions of the original array. Remember to transform the measurement matrix first by multiplying on the right by V, then untransform the results by multiplying on the left by V.

The file `visualize2.py` contains code to visualize your results.

```
>>> from visualize2 import visualizeEarth
>>> results = # Do the compressed sensing and store the results.
```

```
>>> visualizeEarth(faces, vertices, results.clip(0,1))
```

We could have reconstructed a more detailed earth by choosing a finer tessellation, but you should be able to make out the shapes of the major continents. Compare your results to the original:

```
>>> visualizeEarth(faces, vertices, colors)
```

Do the reconstruction with 250, 400, and 550 measurements. Return a list containing the Euclidean distance between each reconstruction and the array of actual colors.