Beginners Code for Compressive Sensing

Alejandro Weinstein

September 2009

1 Sparse Signals in the Time Domain

1.1 Using a Random Sensing Matrix

In this first example we will measure a signal that is sparse in the time domain. We will use a random sensing matrix, and we will solve the recovery problem using the l_1-Magic toolbox.

We use the following functions to generate the signals and the sensing matrix:

Listing 1: Sparse signal and random measurement matrix.

```
function f = get_sparse_fun(n,s)
    tmp = randperm(n);
    f = zeros(n,1);
    f(tmp(1:s)) = randn(s,1);

function A = get_A_random(n,m)
    A = sqrt(1/m)*randn(m,n);
```

The following script use these functions to generate the signal, take the measurements and do the recovery. Figure 1 shows the result.

Listing 2: Example 1.

```
1  % CS example 1
2  % Sensing matrix phi is random.
3  % Representation basis Psi is the canonical basis.
4  % Recovering using l_1 magic.
5
6  clc
7  clear all
8  close all
9
10 n = 512; % Signal length
11 s = 25;  % Sparsity level
12 m = 5*s; % Number of measurements
13
14 f = get_sparse_fun(n,s);
15 A = get_A_random(n,m);
16
17 y = A*f; % Take the measurements
18
19 % Solve using l_1 magic.
20 path(path, './Optimization');
21 x0 = pinv(A)*y; % initial guess = min energy
22 tic
23 xp = l1eq_pd(x0, A, [], y, 1e-3);
24 toc
25
26 norm(f-xp)/norm(f)
27 plot(f)
28 hold on
29 plot(xp,'r.')
30 legend('Original', 'Recovered')
```

The recovery can be made by using CVX instead of l_1-Magic. Just replace lines 19 to 24 by

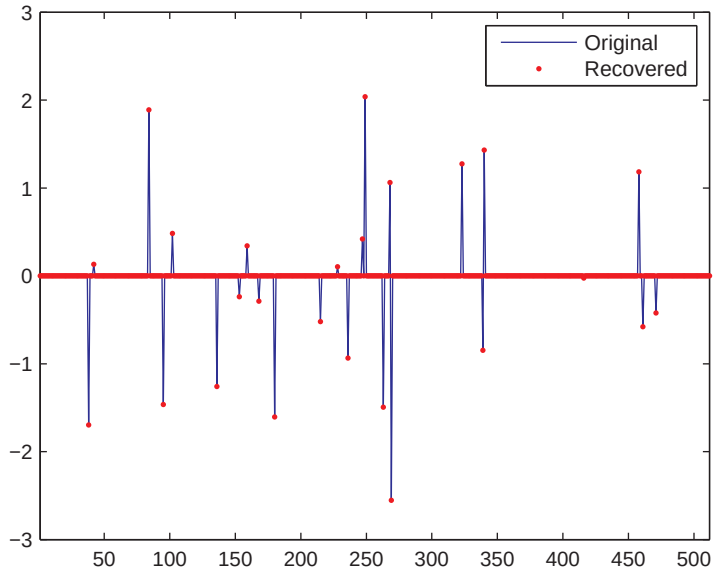

Figure 1: Script 1 results.

Listing 3: Using CVX for the recovery.

```
% Solve using CVX.
cvx_begin
    variable xp(n);
    minimize(norm(xp,1));
    subject to
        A*xp == y;
cvx_end
```

1.2 Using a Fourier Sensing Matrix

Now we are going to repeat the same experiment, but using a sampling matrix based on the Fourier basis. We generate the measurement matrix with the following function:

Listing 4: Fourier based measurement matrix.

```
function A = get_A_fourier(n,m)
    tmp = randperm(n);
    phi = inv(fft(eye(n)));
    A = phi(tmp(1:m/2),:);
    A = [real(A) ; imag(A)];
```

In order to recover the signal using l1-magic, now we need to use the function `l1qc_logbarrier` instead of `l1eq_pd`. On the other hand, there is no need to change anything when solving the problem with CVX. Since in general is simpler and clearer to use CVX, we only use this approach in the following examples.

2 Sparse Signal in the Frequency Domain

Lets try now with a signal sparse in the frequency domain. We generate the signal as:

Listing 5: Sparse signal in the frequency domain.

```
t = [0:n-1]';
f = cos(2*pi/256*t) + cos(2*pi/128*t);
```

Lets solve with a random sensing matrix and CVX. Notice that now we need to specify the representation basis Ψ (see line 5[1]):

Listing 6: Random measurements and CVX recovery.

```
1  A = get_A_random(n,m);
2  y = A*f;
3
4  % Solve using CVX.
5  Psi = inv(fft(eye(n)));
6  cvx_begin
7       variable xp(n);
8       minimize(norm(xp,1));
9       subject to
10          A*Psi*xp == y;
11 cvx_end
```

Figure 2 shows the result. As expected, the recovery is exact. Lets modify our signal slightly, by replacing one of the cosine by a sine:

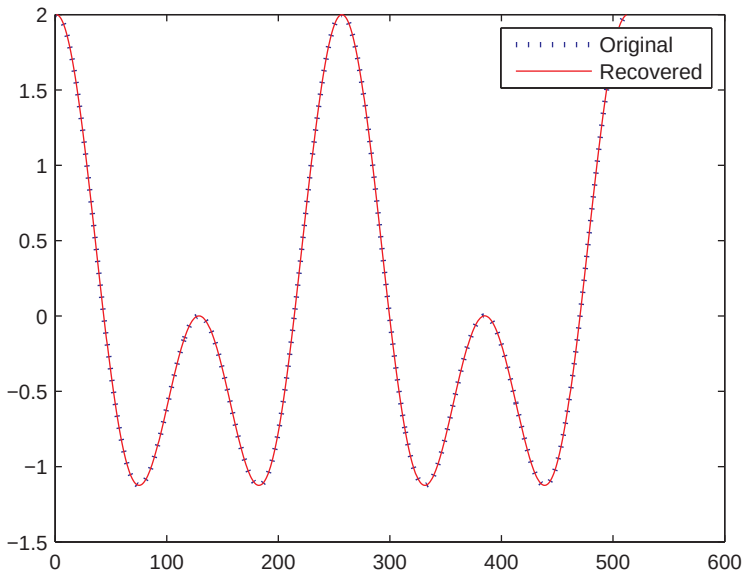

Figure 2: Recovery of a frequency domain sparse signal.

Listing 7: Sparse signal in the frequency domain.

```
t = [0:n-1]';
f = cos(2*pi/256*t) + sin(2*pi/128*t);
```

Figure 3 shows the result. Evidently there is something wrong. The problem is that now the Fourier coefficients have an imaginary component, but CVX is searching for a real x. The solution is easy, we just need to tell CVX to consider a complex x:

Listing 8: Sparse signal in the frequency domain.

```
% Solve using CVX.
Psi = inv(fft(eye(n)));
cvx_begin
     variable xp(n) complex; % We need to tell CVX that xp is complex!
     minimize(norm(xp,1));
     subject to
          A*Psi*xp == y;
cvx_end
```

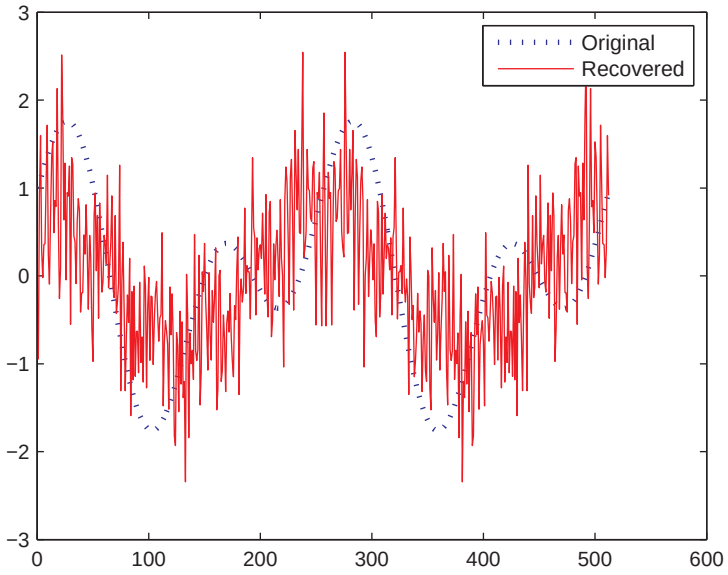

Figure 3: Wrong recovery.

Figure 4 shows the result. Now the recovery is exact.

We can also try with a higher sparsity level. The following script create a signal by adding six sinusoids with random period, amplitude and phases. Notice that the periods are chosen from the vector `[16 32 64 128 256 512]`. Figure 5 shows the result. Once again, the recovery is exact.

Listing 9: Signal made of 6 random sinuoids.

```
s = 6;
amp = rand(s,1); % amplitudes
periods = [16 32 64 128 256 512];
tmp = randperm(length(periods));
freq = (2*pi./round(periods(tmp(1:s)))); % frequencies
phases = 2*pi*rand(s,1);
f = zeros(n,1);
t = [0:n-1]';
for k = 1:s,
    f = f + amp(k)*cos(freq(k)*t + phases(k));
end
```

We now replace the periods we are using by `[18 32 64 128 256 512]`. Notice that the only difference is that now the smallest period is 18 instead of 16. Figure 6 shows the result. Now the result is not exact. The reason for this is that now the signal is not really sparse, since one of the periods is not an integer multiple of the signal length.

3 Acknowledgments

Thanks to Dr. Michael Wakin and Borhan Sanandaji for helping me to solve some of the issues I had with the code.

[1] If the signal is sparse in the time domain, Ψ = Identity matrix, thats why we didn't specify Ψ in section 1.

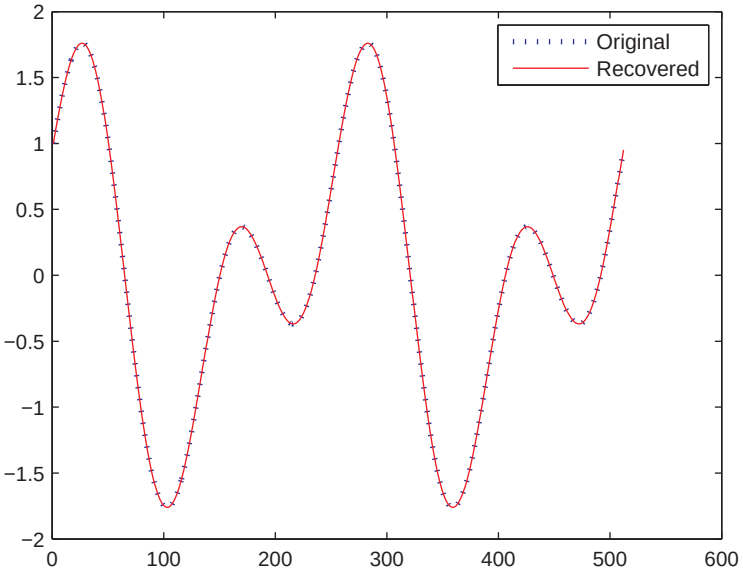

Figure 4: Successfully recovery after telling CVX to use a complex variable.

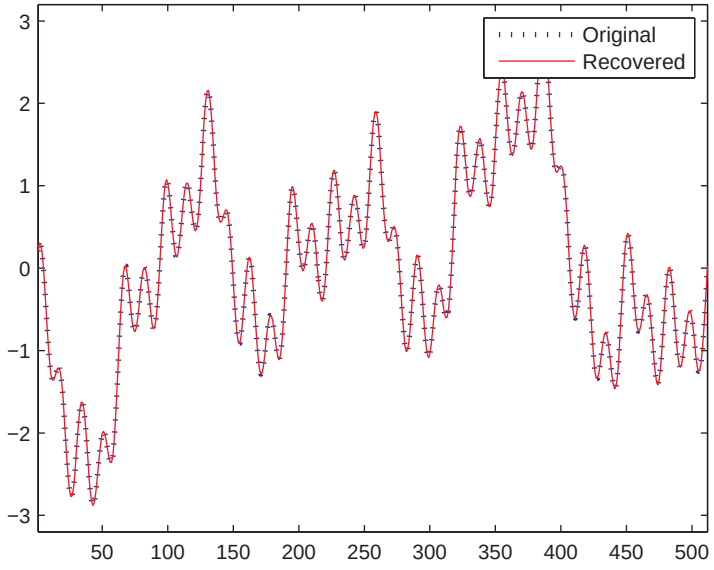

Figure 5: Recovery of a signal made of 6 sinusoids.

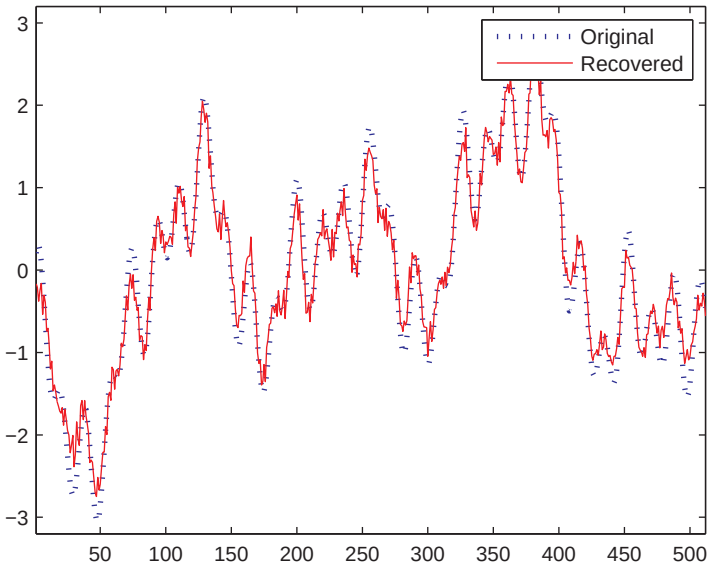

Figure 6: Recovery of a signal made of 6 sinusoids. The smallest period is now 18 instead of 16.