

Compressive sampling, or how to get something from nothing (probably)

Willard Miller and Li Lin

miller@ima.umn.edu, linxx501@umn.edu

University of Minnesota

The problem 1

- A signal is a real n -tuple $x \in R_n$. To obtain information about x we sample it. A sample is a dot product $r \cdot x$ where $r = (r_1, \dots, r_n)$ is a given sample vector. Once r is chosen, the dot product is easy to implement in hardware.
- In the case where $r_i = \delta_{ij}$ for fixed j and $i = 1, \dots, n$ the sample yields the value of the component x_j .

The problem 2

- Take m different samples y_ℓ of x with m distinct sample vectors $r^{(\ell)}$, $\ell = 1, \dots, m$. We can describe the sampling process by the matrix equation $y = \Phi x$ where the $m \times n$ sampling matrix Φ is

$$(\Phi_{\ell,j}) = \left(r_j^{(\ell)} \right) = \begin{pmatrix} r^{(1)} \\ \vdots \\ r^{(m)} \end{pmatrix}$$

and y is an m -tuple.

- The problem is to design the sampling matrix Φ so that we can determine the signal x uniquely from m samples.

The problem 3

- Given the $m \times n$ sampling matrix Φ and the m -tuple sample vector y , find the signal vector x such that

$$y = \Phi x.$$

- This is a linear algebra problem, m equations in n unknowns, solved by Gaussian elimination.
- However, we would like $m \ll n$ so that there are many fewer samples than the length of the signal. If we can reconstruct the n -tuple x exactly from the sample m -tuple y we will have achieved data compression.
- Problem: If $m < n$ the system of equations $y = \Phi x$ is *underdetermined* and never has a unique solution.

The problem 4

- Indeed if $m < n$ and $y = \Phi x^{(0)}$, so that y is the sample produced by the signal $x^{(0)}$ then y is also produced by all signals $x = x^{(0)} + u$ where u is a solution of the homogeneous equation $\Phi u = 0$.
- Fact: If $m < n$ then the equation $\Phi u = 0$ has a vector space of solutions of dimension at least $n - m$.
- How is it possible to find signal $x^{(0)}$ if many signals give the same sample vector?

The strategy 1

- The mathematical problem as stated is impossible to solve. However, we can find unique solutions if we restrict the signal space.
- We are especially interested in the case where x is k -sparse, i.e., at most k of the components x_1, \dots, x_n of x are nonzero. We think of k as small with respect to n .
- Sparse or almost-sparse signals are very common in signal processing applications.
- Modified problem: Given k, n with $k \ll n$ design an $m \times n$ sample matrix Φ such that we can uniquely recover any k -sparse signal x from its sample $y = \Phi x$, and such that m is as small as possible.

The bit counter 1

- Suppose we have a long n -component signal x that is 1-sparse. The signal consists of a single spike at some location s with all other components 0. Number the rows of Φ from 0 to $m - 1$ and the columns from 0 to $n - 1$. Take $n = 2^{m-1}$.
- Write integer j in binary
$$j = j_0 2^0 + j_1 2^1 + \cdots + j_{m-1} 2^{m-1}, j_s = 0, 1.$$
- Fill the 0th row of Φ with 1s.
- Design the remaining rows of the $m \times n$ sampling matrix Φ so that $\Phi_{ij} = j_{m-i}$ for $i = 1, \cdots, m - 1$.

The bit counter 2

- Computing $y = \Phi x$ we have $[y_{m-1}, \dots, y_1] = x_s[\ell_{m-1}, \dots, \ell_0]$ where $s = [\ell_{m-1}, \dots, \ell_0]$ is the location of the spike in binary and y_0 is the value of the spike.
- For example, suppose $m = 4$, $n = 8$ and the signal x has the spike $x_6 = 6.1$.

The bit counter 3

Then $y = \Phi x$ becomes

$$\begin{pmatrix} 6.1 \\ 0 \\ 6.1 \\ 6.1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 6.1 \\ 0 \end{pmatrix}$$

The value of the spike is $v = 6.1$ and the location is $[1, 1, 0] = 6$.

The strategy 2

- Thus the bit counter satisfies $m = \log_2 n + 1$ for $k = 1$. We have logarithmic compression of data.
- Mathematical observation: $x^{(1)}, x^{(2)}$ are k -sparse signals with the same sample $\Leftrightarrow \Phi x^{(1)} = \Phi x^{(2)}$
 $\Leftrightarrow \Phi(x^{(2)} - x^{(1)}) = 0 \Leftrightarrow \Phi u = 0$ has a solution $u' = x^{(2)} - x^{(1)}$ that is $2k$ -sparse.
- If we can design a sampling matrix Φ such that its homogeneous equation has no nonzero solutions u that are $2k$ -sparse, then for a given sample y there is at most one k -sparse signal $x^{(0)}$ such that $y = \Phi x$.

The strategy 3

- Solution: Among all signals x with sample y the k -sparse solution $x^{(0)}$ has minimal ℓ_1 norm (but not minimal ℓ_2 , least squares norm).
- The ℓ_1 norm of an n -tuple x is

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|.$$

Recall that the ℓ_2 norm is

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

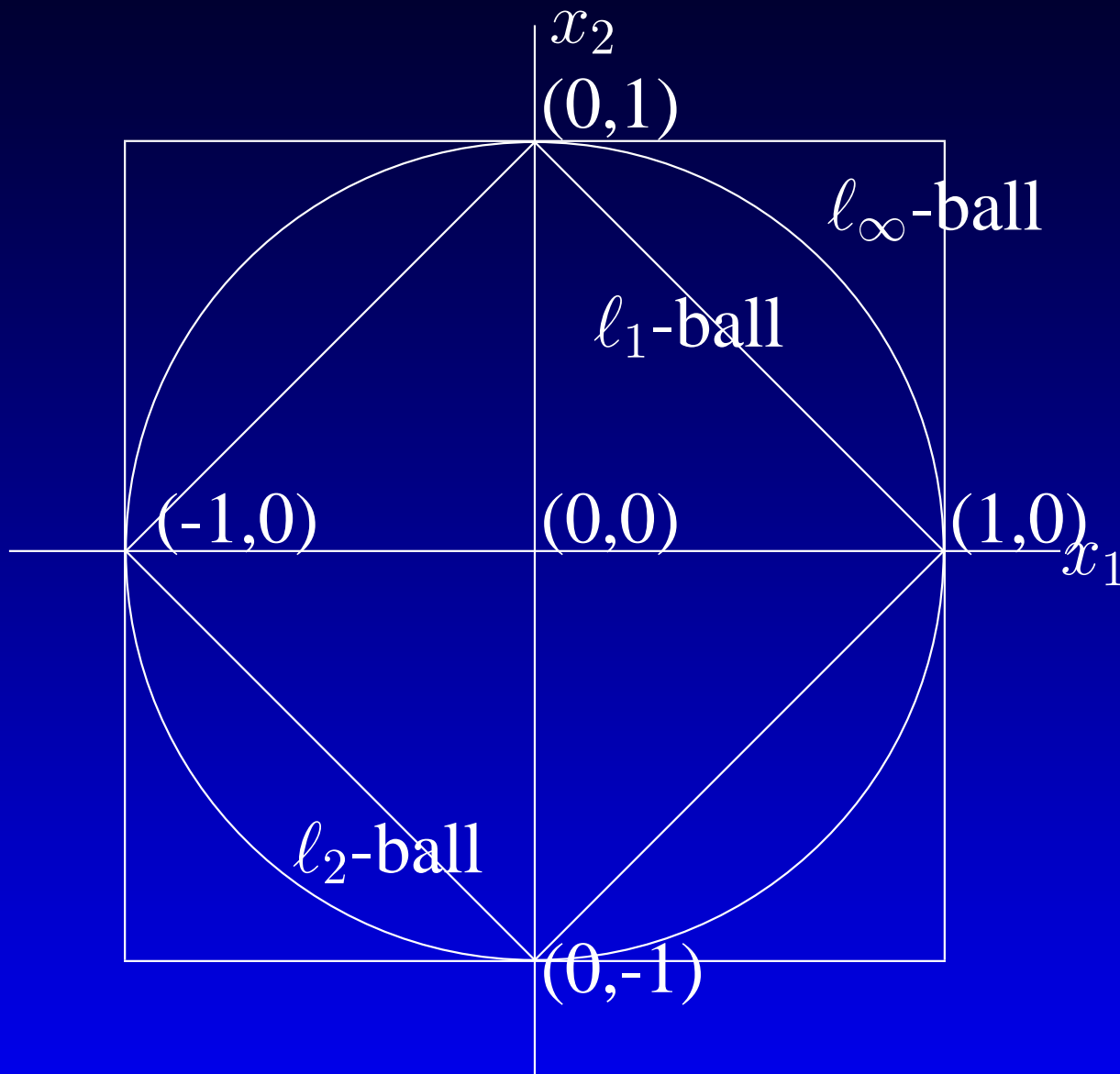
The strategy 4

- If the homogeneous equation has no $2k$ -sparse solutions then we can recover a k -sparse signal from its sample y by solving the minimization problem

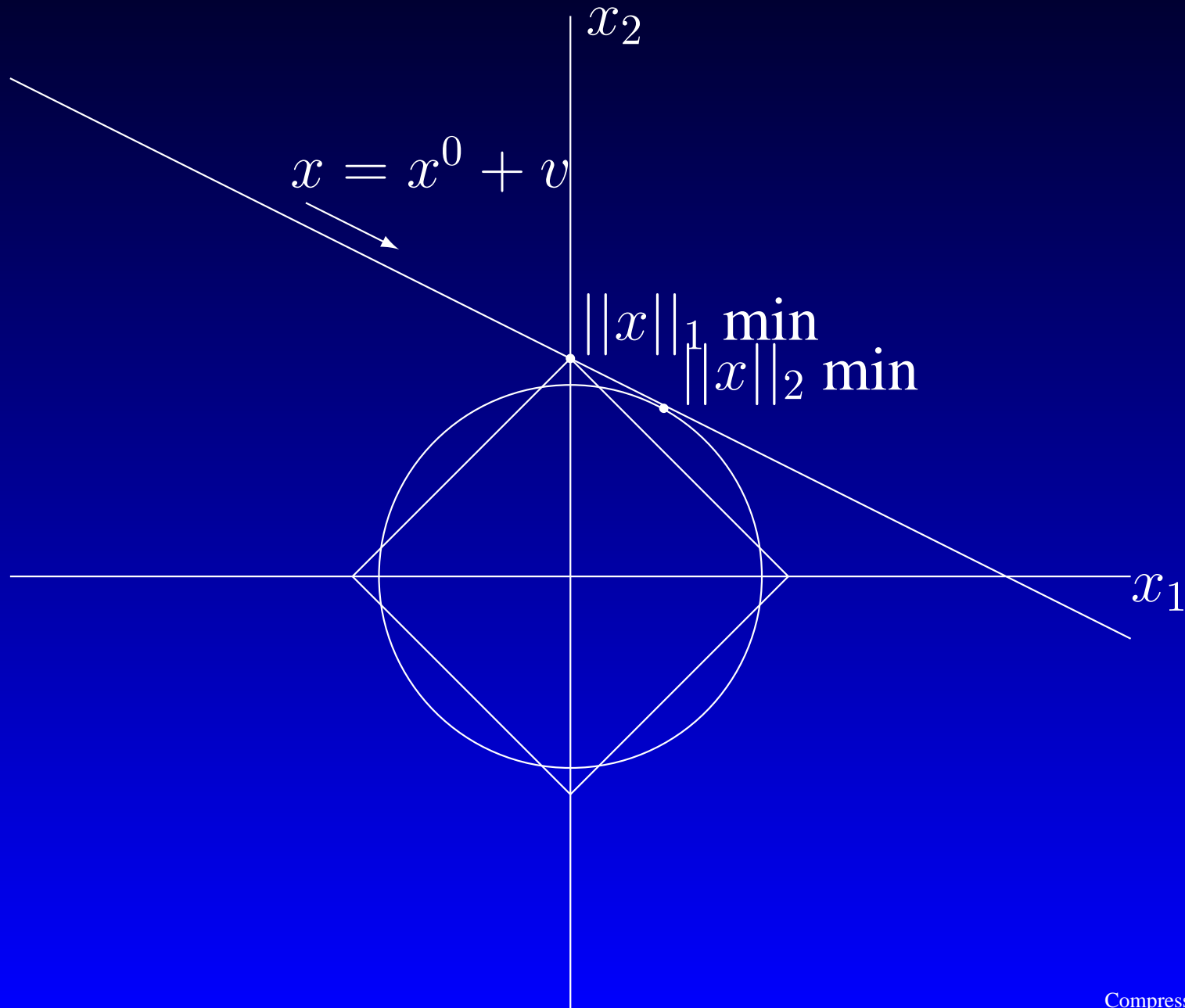
$$x^{(0)} = \operatorname{argmin}_{\Phi x = y} \|x\|_1.$$

- This is a straightforward linear programming problem that can be solved in polynomial time by standard algorithms, freely available even for spreadsheets.
- This works because the ℓ_1 minimum always occurs for a vector with a maximal number of zeros.

The ℓ_1 and ℓ_2 unit balls for $n = 2$



ℓ_1 and ℓ_2 (least squares) minimization



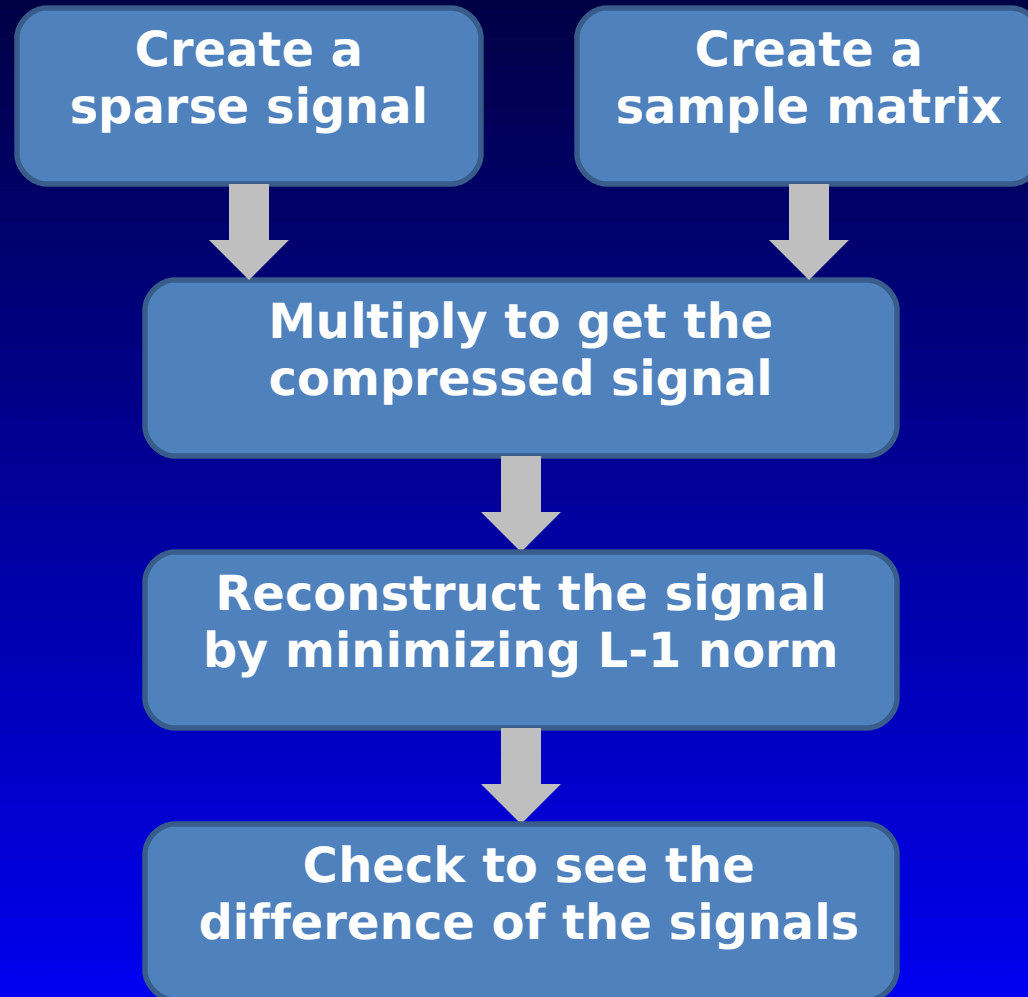
Construction of the sample matrix Φ 1

- Given $k \ll n$ how do we construct an $m \times n$ sample matrix Φ with m as small as possible?
- Surprise! The bit counter example is somewhat misleading. Matrices with homogeneous equations admitting $2k$ -sparse solutions are rather special. An arbitrarily chosen matrix is less likely to have this failing. We can use a random number generator, or coin flipping, to choose the matrix elements of Φ .

Construction of the sample matrix Φ 2

- Using random matrix theory we can show that the smallest m that works is about $m = Ck \log_2 n$ where C is a constant, close to 1 in practice.
- We can construct the matrices via random number generators and guarantee that they will “probably” work, e.g., guarantee no more than 1 failure in 10^9 sample matrices.
- Demonstration to follow.

Demonstration: Method of Programming



Program to Show the Process

% Step 1: Define the relative parameters

```
n=1024           % n is the total length of the signal vector
k=10           % k is the number of sparse signals
m=110         % m*n is the size of the sample matrix
good=0        % Used to check the reconstructed signal
successful=0  % Used to show if the method is successful
t=(0:1:(n-1)) % This parameter is for graphing at last
```

% Step 2: Create the sparse signal

```
support=randsample(n,k) % Determine the positions of non-zero
signals by selecting k random integers from 1 to n without replacement
x0=zeros(n,1)         % Creates a n*1 vector of zeros
x0(support)=randn(k,1) % Fill in the selected positions with a
random normal distributed number
```

Program (Continued)

% Step 3: Create the sample matrix using method of random number

```
A=unifrnd(-1,1,m,n) % Create a m*n sample matrix A which  
contains number from -1 to 1
```

% Step4: Compressing and reconstructing the signal

```
b=A*x0 % Multiply the original signal by the sample matrix  
% b is the compressed signal with length of m
```

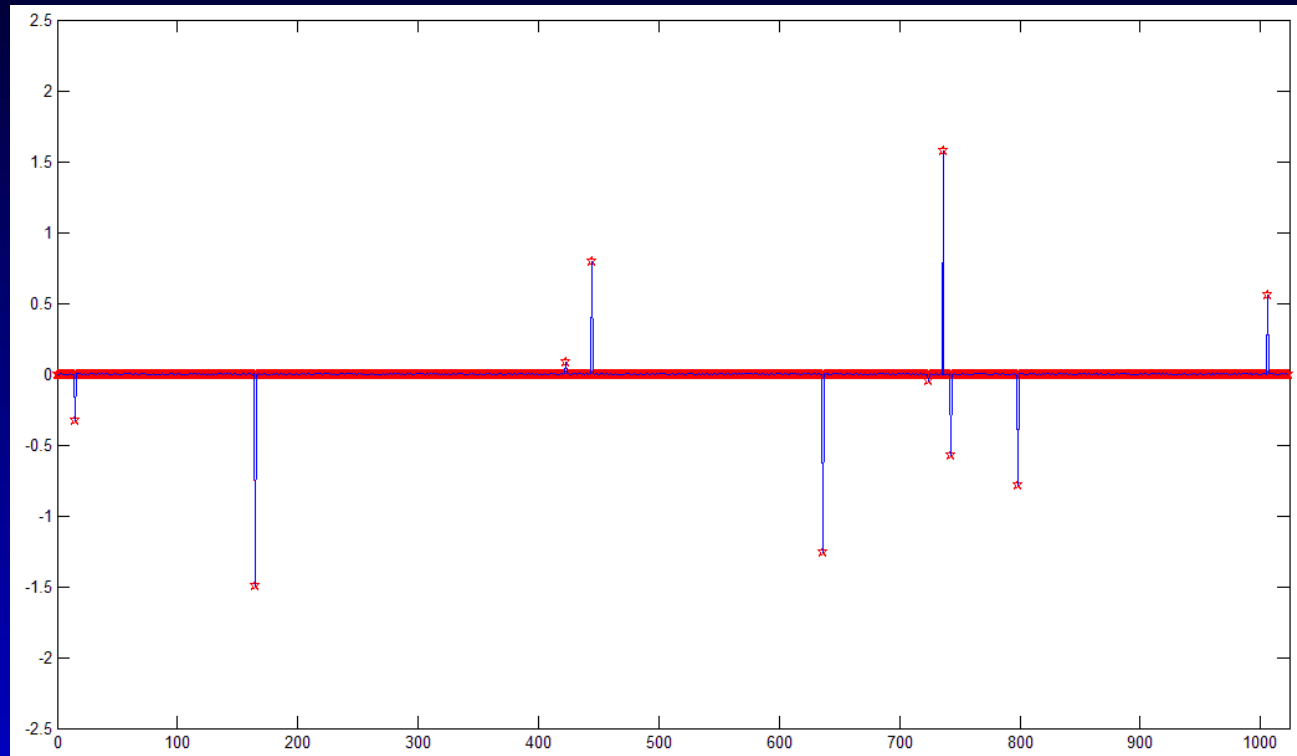
```
cvx_begin % "cvx" is a program used to  
reconstruct the signal  
variable x(n) % This solves for x by minimizing the  
L1 norm given 'A'&'b'  
minimize(norm(x,1)) % and the vector x is the reconstructed  
signal  
subject to  
A*x==b  
cvx_end
```

Program (Continued)

```
% Step 5: Checking the difference from the original and recovered signal
z=x-x0 % Calculate the difference
for i=1:n % When the difference is less than 10^(-8), then it's good
    if (abs(z(i,1))<(10^-8))
        good=good+1
    else
        end
end
if good==n % If all the point on the reconstructed signal is good
    successful=successful+1 % it will show " successful=1 "
else % Otherwise, it will show " successful=0 "
    successful=successful
end

% Last Step: Sketch the two signals and compare them
plot(t,x0,'r:p') % Original signal is represented by red
pentagram
hold on
plot(t,x,'b') % Reconstructed signal is represented by blue lines
hold off
```

Graph of the Signal (Successful!)



- In this demonstration, $m = 110$, $n=1024$, $k=10$
- According to formula $m = Ck \log_2 n$, we get
$$C = 110/(10 * \log_2 10) = 1.1$$

Thank you!