Lecture 22
Signal Reconstruction and Circumventing Nyquist

Jason J. Bramburger

Our goal in this lecture will be to try to reproduce a signal. Let's say that we have a signal that is sampled at Fs = 44100 Hz. That means that there are 44100 samples (or points) per second. Therefore, if we have a 5 second audio signal (i.e. $L = 5$), the total number of points would be $N = 44100 \cdot 5 =$Fs $\cdot L$. If you apply the FFT, the frequencies (in Hertz) are

```
k = 1/L*[0:N/2-1 -N/2:-1]
```

In particular, the largest frequency in your signal is $N/(2L) = \text{Fs}/2$, which is half of the sampling frequency. This is called the **Nyquist frequency**. So, whatever frequency you sample at, you have information about your signal up to half of that frequency. This fact can also be used in reverse. What if you know that you have a signal that has a maximum frequency of 1000 Hz. If you want to be able to reconstruct the signal, you need a to sample at a rate of at least 2000 Hz. This is known as the **Nyquist–Shannon sampling theorem**. It is a HUGE theorem in signal processing. It proves that there is some limit to your sampling rate to reconstruct certain signals. For example, it is impossible to reconstruct a 1000 Hz signal if you use a sampling rate of only 1500 Hz.

Because of the Shannon–Nyquist sampling theorem, for a long time people just thought it was impossible to reconstruct a signal unless you sample at twice the frequency of the maximum signal (that is what the theorem says after all). But then in 2004 there was a breakthrough that essentially kick-started the field of compressed sensing. The main idea is that if you know that your signal is sparse in some way, you can reconstruct it with less information by using the 1-norm.

Suppose that we have a signal $f(t)$. We know from our previous lectures that there are a number of different ways to take this signal and represent it in terms of some basis. For example, the Fourier transform/series represents the signal in terms of a basis of sines and cosines (or complex exponentials):

$$f(t) = \sum_{k=-\infty}^{\infty} c_k \mathrm{e}^{\mathrm{i}kt}.$$

We could also use wavelets too (see Lecture 6). In general, we can write

$$f(t) = \sum_j c_j \psi_j(t)$$

if we are given sufficient basis functions $\psi_j(t)$. Now, we know that practically we don't get functions $f(t)$ - we get vectors (discretized functions). Therefore, the basis expansion becomes a matrix equation

$$\begin{bmatrix} f(t_1) \\ f(t_2) \\ \vdots \\ f(t_n) \end{bmatrix} = \begin{bmatrix} \psi_1(t_1) & \cdots & \psi_n(t_1) \\ \psi_1(t_2) & \cdots & \psi_n(t_2) \\ \vdots & \ddots & \vdots \\ \psi_1(t_n) & \cdots & \psi_n(t_n) \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

or compactly

$$\mathbf{f} = \Psi \mathbf{c}.$$

If you are using a Fourier basis, then $\mathbf{c}$ is just the FFT of $\mathbf{f}$ and Ψ is a matrix that performs the inverse FFT. In order to use compressed sensing, we need to choose a basis where the signal is sparse. This is practical for many applications. A lot of signals might be sparse in Fourier space (think images). Other signals might be sparse in a wavelet basis.

Now, suppose we have a signal represented by a function f for which there are five data points available. We have a vector of five points

$$\begin{bmatrix} f(t_1) \\ f(t_2) \\ f(t_3) \\ f(t_4) \\ f(t_5) \end{bmatrix}$$

If we only want to sample the second and fourth points, we can think of this as matrix multiplication:

$$\begin{bmatrix} f(t_2) \\ f(t_4) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} f(t_1) \\ f(t_2) \\ f(t_3) \\ f(t_4) \\ f(t_5) \end{bmatrix}$$

which we can write compactly as

$$\mathbf{b} = \Phi \mathbf{f}.$$

Recall that we had the matrix equation $\mathbf{f} = \Psi \mathbf{c}$. If we multiply both sides by Φ, we get

$$\Phi \mathbf{f} = \Phi \Psi \mathbf{c}$$

which is just $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A} = \Phi\Psi$ and the unknown $\mathbf{x} = \mathbf{c}$. This is now an underdetermined system. How do I know that? Two ways:

1. $\mathbf{A}$ has more columns than rows (Ψ is square and Φ is longer than it is tall).
2. Think about what we are solving with $\mathbf{A}\mathbf{x} = \mathbf{b}$. If the signal has length n and $\mathbf{b}$ consists of m samples ($m < n$), we are essentially asking: what is the signal of length n that goes through m points given by $\mathbf{b}$? There are infinitely many!

In the previous lecture we saw that we can solve underdetermined systems in MATLAB using backslash, pinv() (which is the same as minimizing the 2-norm), or by minimizing the 1-norm. But, we know that the solution $\mathbf{x} = \mathbf{c}$ is sparse! So, we want to use the 1-norm!

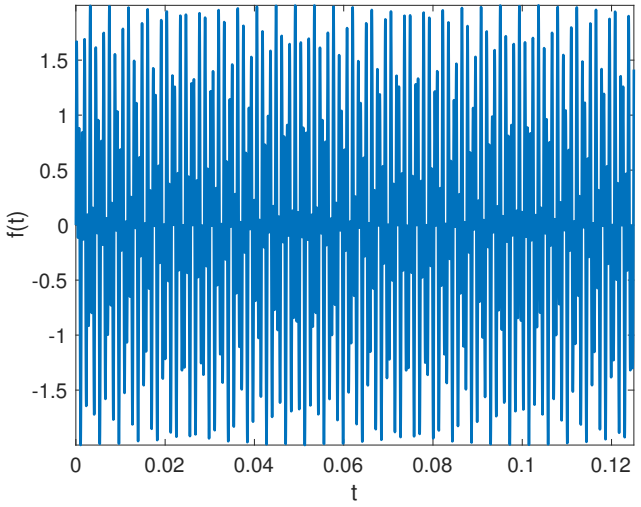
Discrete Cosine Transform

In the coming example we are going to use a variant of the DFT called the discrete cosine transform (DCT). We saw that the FFT uses a basis of complex exponentials, which is equivalent to sines and cosines. The DCT uses only cosines as the basis. How is this possible? Recall that the FFT implicitly assumes that the function is periodic. It does this by imagining the function in $[-\pi, \pi]$ tiled over and over in both directions. The DCT does something similar, but it implicitly assumes that the function is even and periodic. It does this by taking a function on $[0, \pi]$ and mirroring it over 0 to produce an even function on $[-\pi, \pi]$ and then applying the FFT. Remember that even functions have no sine components in their Fourier transform. The advantage here is that the DCT gives coefficients that are all real numbers, unlike the FFT which gives complex numbers. This is just going to make our lives a little easier.

Let's see an example. Consider the 'A' key on a touch-tone phone. That audio signal is the sum of two sines:

$$f(t) = \sin(1394\pi t) + \sin(3266\pi t).$$

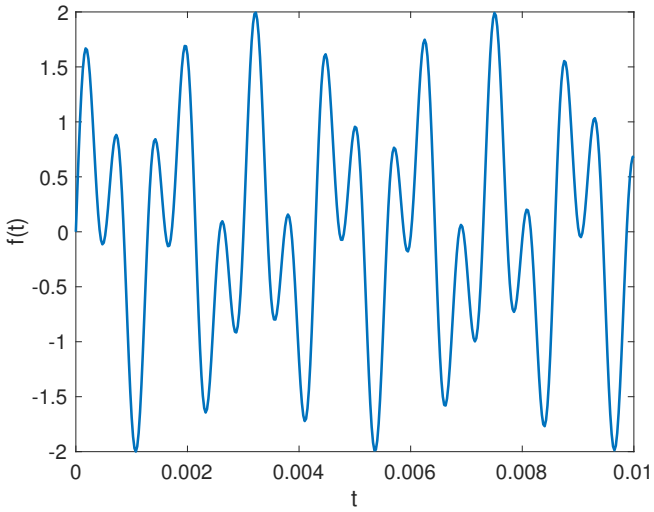
```
N = 5000;
t = linspace(0,1/8,N);
f = sin(1394*pi*t) + sin(3266*pi*t);

figure(1)
plot(t,f,'Linewidth',2)
xlabel('t')
```




```
ylabel('f(t)')
set(gca,'Fontsize',16)
axis tight
```

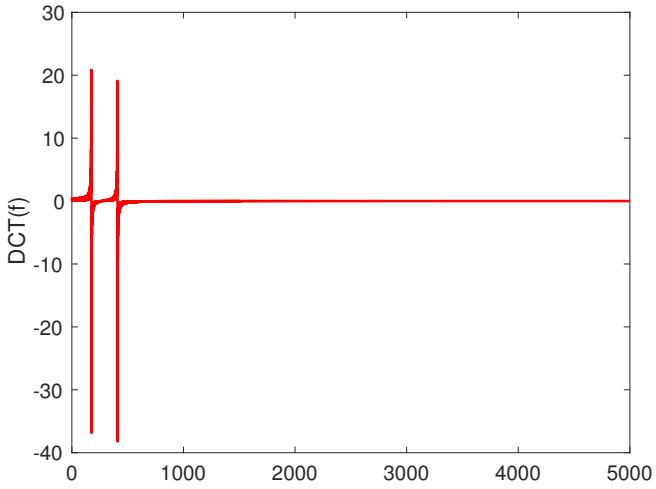
Let's zoom in a bit to better understand the signal.

```
figure(2)
plot(t,f,'Linewidth',2)
xlabel('t')
ylabel('f(t)')
xlim([0 0.01])
set(gca,'Fontsize',16)
```



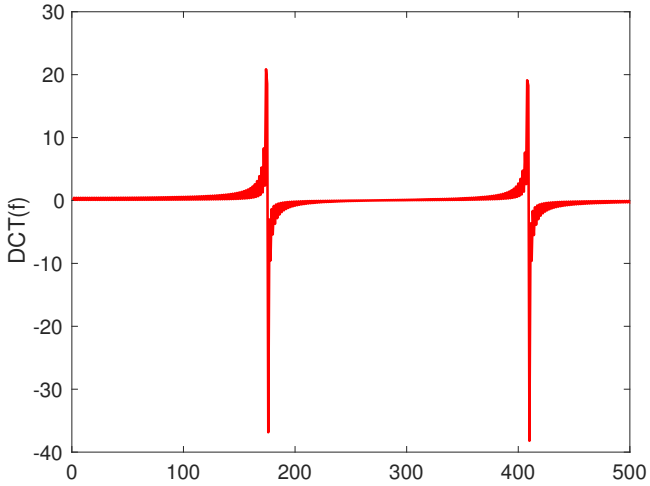
Now, let's calculate the DCT and plot it. We don't need to worry about the scale of the frequencies, we just want to make sure it is a sparse vector.

```
ft = dct(f);

figure(3)
plot(ft,'r','Linewidth',2)
ylabel('DCT(f)')
set(gca,'Fontsize',16)
```



You can see two large peaks and then everything else is approximately zero. However, it is worth noting that it is not truly sparse. First of all, the peaks don't fall exactly on the frequencies given by the DCT so they span several values. But also, the numbers that are close to zero are not exactly zero. They are just really small. This is quite common. This is exactly what minimizing the 1-norm gave us in the last lecture.

```
figure(4)
plot(ft,'r','Linewidth',2)
ylabel('DCT(f)')
xlim([0 500])
set(gca,'Fontsize',16)
```

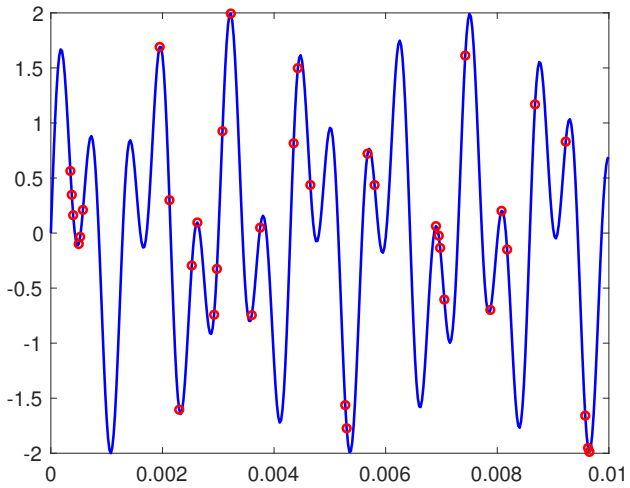


Sparse Sampling

Now we are going to randomly sample 10% of the points in the signal. You might be able to do better by employing a sampling strategy (as opposed to randomly sampling). Indeed, optimizing sampling strategies is its own area of research. For us though, random is good enough. Let's plot the original signal and the sampled points.

```
m = 500; % number of points
perm = randperm(5000); % random permutation of the numbers from 1 to 5000
ind = perm(1:m); % 500 randomly selected numbers (indices of subsampled points)
tr= t(ind); % times of subsampled points
fr = f(ind); % subsampled values

```

```
figure(5)
plot(t,f,'b',tr,fr,'or','Linewidth',2)
xlim([0 0.01])
set(gca,'Fontsize',16)
```



Notice that there are some regions where the sampled points are close together - close enough that we could expect to be able to reconstruct the signal very well in those areas, even without sophisticated techniques. However, there are regions that have full oscillations between points. The Nyquist–Shannon sampling theorem would say that we can't reconstruct that part of the signal without more points.

Reconstructing the Signal

Now we want to create our $\mathbf{A}$ matrix. First, we create the Ψ matrix. Multiplication by this matrix inverts DCT. We can take an identity matrix and perform an inverse DCT on it. Then $\mathbf{A}$ comes from taking the rows of Ψ that correspond to the sampled points.

```
Psi = idct(eye(N,N)); % Create matrix Psi that does inverse DCT
A = Psi(ind,:); % The A matrix is subsampled rows of Psi
```

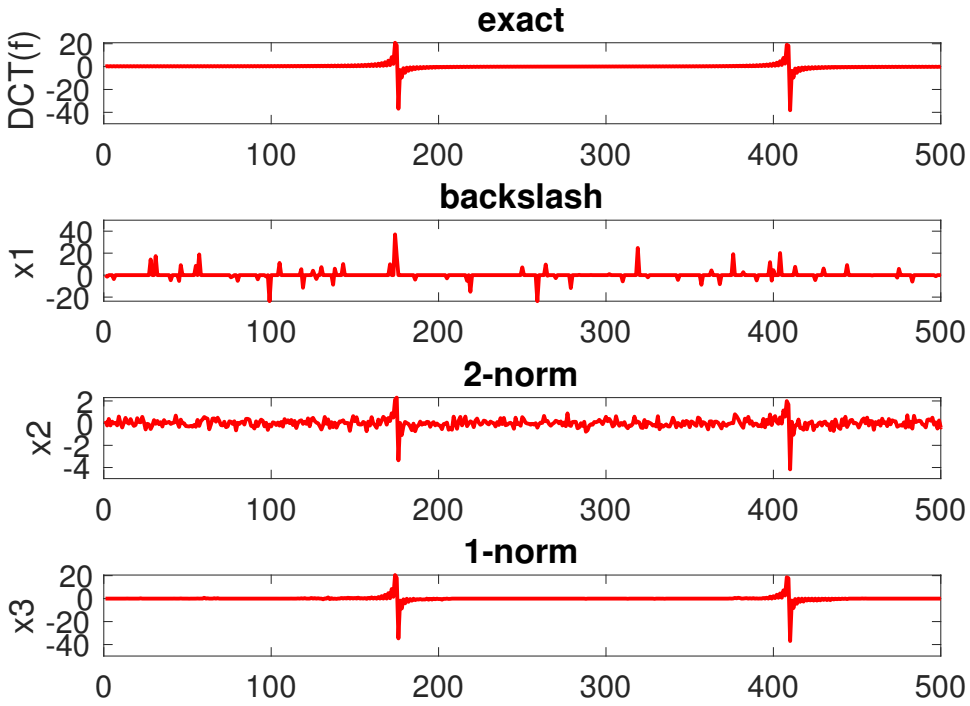
Now we solve $\mathbf{Ax} = \mathbf{b}$. The vector $\mathbf{b}$ is just the vector of sampled points. Currently, Φf (called fr in the code) is a row vector, so $\mathbf{b}$ is its transpose. Then we will solve in three ways: backslash (sparse solution with at least 4500 zeros), pinv() (minimizes the 2-norm), and then the solution that minimizes the 1-norm.

```
b = fr'; % b is a column vector of subsampled values

x1 = A\b;
x2 = pinv(A)*b;

cvx_begin quiet
    variable x3(N);
    minimize( norm(x3,1) );
    subject to
        A*x3 == b;
cvx_end
```

We can plot the three solutions. Recall that these solutions are supposed to represent different c and c represents the coefficients in the basis expansion. In our case, it is the cosine transform of the signal. So we plot the cosine transform of the original signal versus these three approximations that were made from the subsampled signal.

```
figure(6)
```

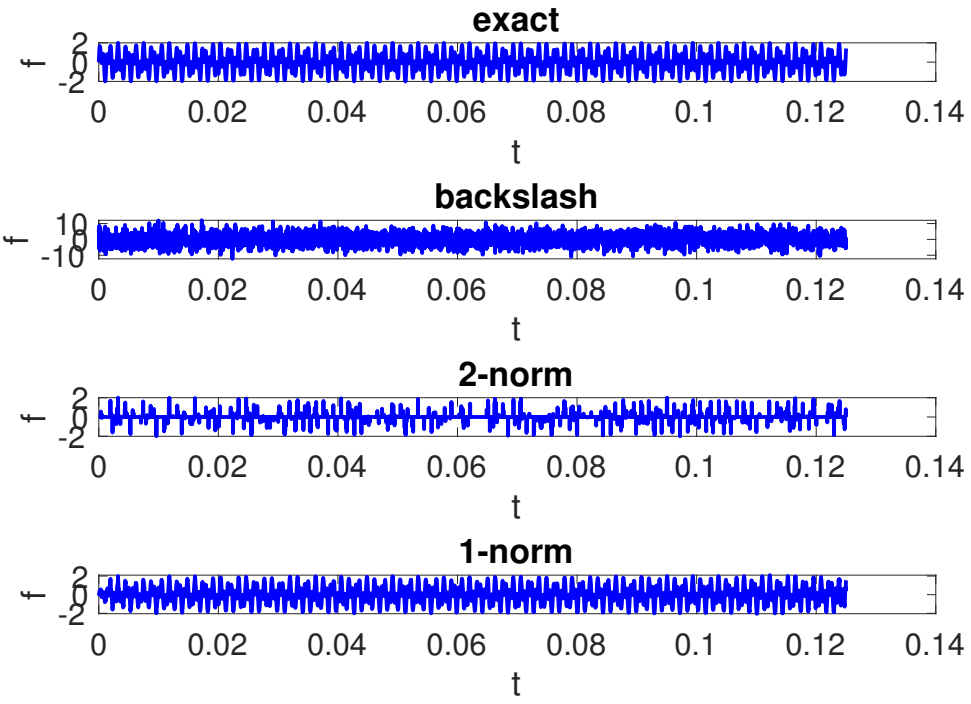
```
subplot(4,1,1)
plot(ft,'r','Linewidth',2)
xlim([0 500])
ylabel('DCT(f)')
title('exact')
set(gca,'Fontsize',16)
subplot(4,1,2)
plot(x1,'r','Linewidth',2)
xlim([0 500])
ylabel('x1')
title('backslash')
set(gca,'Fontsize',16)
subplot(4,1,3)
plot(x2,'r','Linewidth',2)
xlim([0 500])
ylabel('x2')
title('2-norm')
set(gca,'Fontsize',16)
subplot(4,1,4)
plot(x3,'r','Linewidth',2)
xlim([0 500])
ylabel('x3')
title('1-norm')
set(gca,'Fontsize',16)
```



The 1-norm solution looks just like the original! The other two unfortunately do not. The 2-norm gives some peaks near the correct spot, but they aren't nearly high enough (compare the vertical axes). The 2-norm ends up spreading the rest of the energy out across the entire spectrum. Backslash does give a sparse solution, but the peaks aren't anywhere near where they should be. So, it remains to see how well they approximate the signal. To recover the signal, we take an inverse DCT of each vector.

```
sig1 = idct(x1);
sig2 = idct(x2);
sig3 = idct(x3);
```

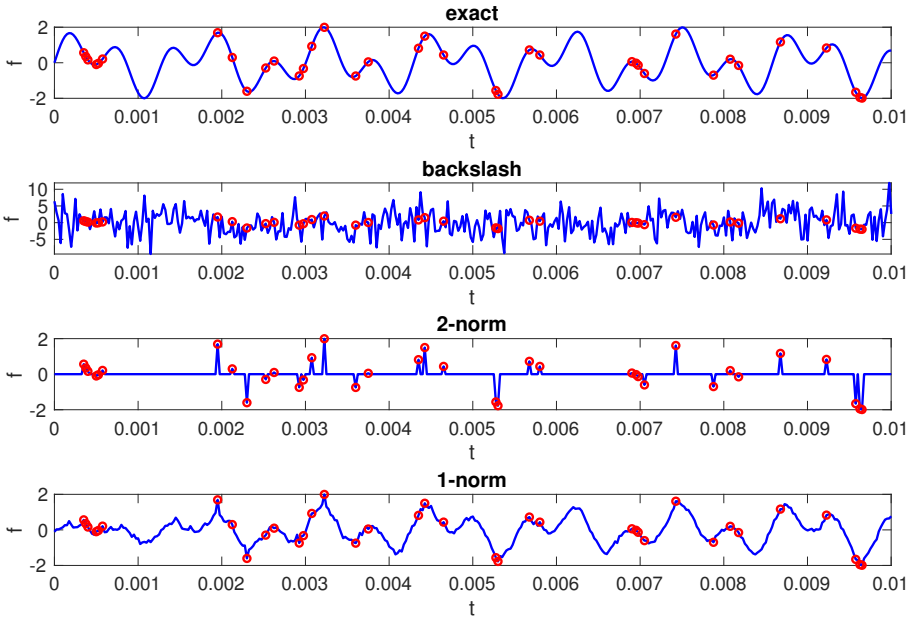
```
4
5  figure(7)
6  subplot(4,1,1)
7  plot(t,f,'b','Linewidth',2)
8  ylabel('f')
9  xlabel('t')
10 title('exact')
11 set(gca,'Fontsize',16)
12 subplot(4,1,2)
13 plot(t,sig1,'b','Linewidth',2)
14 ylabel('f')
15 xlabel('t')
16 title('backslash')
17 set(gca,'Fontsize',16)
18 subplot(4,1,3)
19 plot(t,sig2,'b','Linewidth',2)
20 ylabel('f')
21 xlabel('t')
22 title('2-norm')
23 set(gca,'Fontsize',16)
24 subplot(4,1,4)
25 plot(t,sig3,'b','Linewidth',2)
26 ylabel('f')
27 xlabel('t')
28 title('1-norm')
29 set(gca,'Fontsize',16)
```



Without even zooming in, we can see that the 1-norm gives the best approximation. Now let's zoom in on each signal and add back in the sampled points.

```
1  figure(8)
2  subplot(4,1,1)
3  plot(t,f,'b',tr,fr,'or','Linewidth',2)
4  xlim([0 0.01])
```

```
ylabel('f')
xlabel('t')
title('exact')
set(gca,'Fontsize',16)
subplot(4,1,2)
plot(t,sig1,'b',tr,fr,'or','Linewidth',2)
xlim([0 0.01])
ylabel('f')
xlabel('t')
title('backslash')
set(gca,'Fontsize',16)
subplot(4,1,3)
plot(t,sig2,'b',tr,fr,'or','Linewidth',2)
xlim([0 0.01])
ylabel('f')
xlabel('t')
title('2-norm')
set(gca,'Fontsize',16)
subplot(4,1,4)
plot(t,sig3,'b',tr,fr,'or','Linewidth',2)
xlim([0 0.01])
ylabel('f')
xlabel('t')
title('1-norm')
set(gca,'Fontsize',16)
```



They all go through the sampled points! That is because we solved the linear system that said they have to! They each gave us an answer to the question we were asking. But the 1-norm took advantage of the sparsity in the right way to give the best approximation of the original signal.