**Lecture 23**
**Data (Image) Reconstruction from Sparse Sampling**

Jason J. Bramburger

---

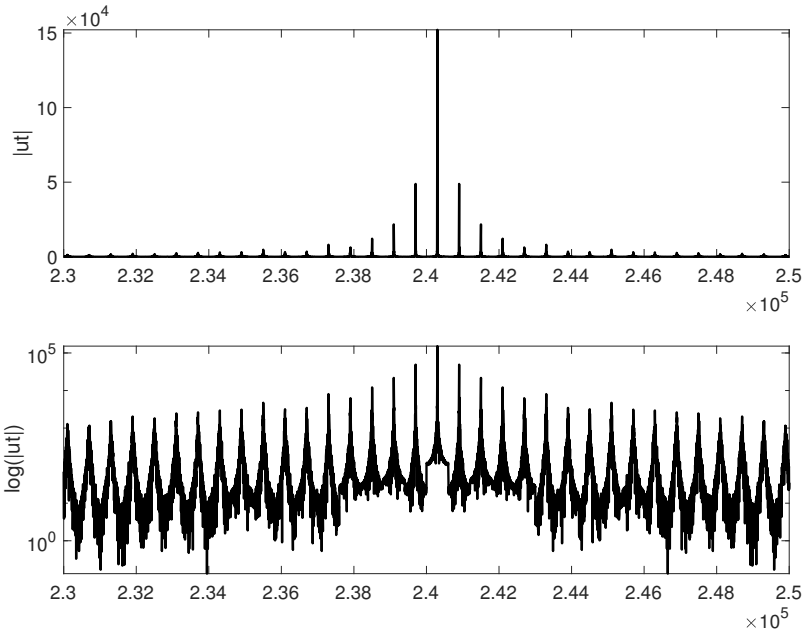
In the previous lecture we explored signal reconstruction. That is, we focussed on 1D data. We are now ready to bring ourselves up to 2D data. Particularly, we will focus on images, and we will see a connection with how image compression works. Let's go ahead and load an image into MATLAB to work with.

```
A = imread('coffee.jpeg');
Abw2 = rgb2gray(A);
Abw = im2double(Abw2);
[nx, ny] = size(Abw);
figure(1), imshow(Abw)
```

The nx and ny variables tell us how large the image is: 600 by 800 pixels. The image is of high resolution and it takes $600 \times 800 = 480000$ pixels to encode the image. That's a lot of values that need to be stored for a simple little picture! We already saw that when we convert to the Fourier domain images typically have a lot of coefficients near zero and so we can reconstruct the image by ignoring (setting to zero) the super small coefficients. Let's confirm this by taking the 2D FFT of our image.

```
At = fftshift(fft2(Abw));

figure(2)
subplot(2,1,1)
plot(reshape(abs(At),nx*ny,1),'k','Linewidth',2)
ylabel('|ut|')
set(gca,'Xlim',[2.3*10^5 2.5*10^5],'Fontsize',16)

% semi-log plot
subplot(2,1,2)
semilogy(reshape(abs(At),nx*ny,1),'k','Linewidth',2)
ylabel('log(|ut|)')
set(gca,'Xlim',[2.3*10^5 2.5*10^5],'Fontsize',16)
```

You can clearly see that many of the coefficients are small relative to the middle spike. The basic idea of image compression follows directly from Fourier analysis. Specifically, we can simply choose a threshold and directly

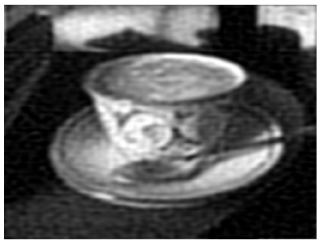
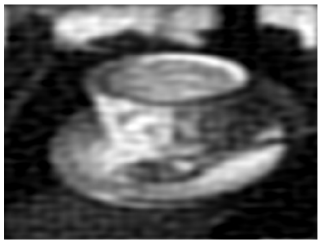



set all Fourier coefficients below this chosen threshold to be zero. If you are able to get away with a fairly large threshold this is going to greatly reduce the amount of data you need to store for the image. Recall that the image is made up of 480 000 pixels, meaning that the FFT contains 480 000 coefficients. Even getting rid of one of these coefficients means we are storing less data.

```
% Open new figure
figure(3)

% Plot original image
subplot(2,2,1)
imshow(Abw)

% Plot thresholded images
count_pic = 2;
for thresh = [1e2 5e2 1e3]
    At2 = reshape(At,nx*ny,1);
    count = 0;
    for j = 1:length(At2)
       if abs(At2(j)) < thresh
          At2(j) = 0;
          count = count + 1;
       end
    end

    percent = 100 - (count/length(At2))*100

    Atlow = fftshift(reshape(At2,nx,ny));
    Alow = (ifft2(Atlow));
    subplot(2,2,count_pic)
    imshow(Alow)
    count_pic = count_pic + 1;
end
```

In our compression procedure we end up using 3.04%, 0.41%, and 0.15%, respectively, of the Fourier coefficients to reconstruct the image. Impressively, the threshold value of 100 results in using only 3% $\approx$ 14600 coefficients but returns a nearly identical image! If I put my glasses on and squint I might be able to find some differences, but we should consider this a big win.



**Compressed Sensing**

In the above procedure we took an image, computed the Fourier transform and then promptly discarded at least 96% of the Fourier coefficients. What if we instead sparse sample the image and try to faithfully reconstruct it instead? In particular, instead of using the 480000 pixels sampled, could we instead randomly sample, say, 5% (24000 pixels) of the pixels and still reconstruct the image? We already know that the image is sparse in the Fourier domain and we've seen that $L^1$ optimization routines allow for faithful reconstruction of sparse data, so let's put these two things together!

If you ran the above thresholding code you probably noticed that it takes a few minutes. To reduce the computation time, I am going to reduce the size of the image to $100 \times 75$ pixels.

```
B = imresize(A,[75,100]);

Abw2 = rgb2gray(B);
Abw = im2double(Abw2);
[ny, nx] = size(Abw);
```

Some comments before moving on. Since the image is now relatively small, it's not nearly as sparse as the high-resolution $800 \times 600$ original. Therefore, the compressed sensing algorithm is not as ideal as it would be in a more realistic problem which would have lots of data points/pixels. Regardless, if you have the time, try to reproduce everything that follows with the original image. It might take a little more time, but it'll be worth it to see the results.

Essentially we are going to follow what we did in the previous lecture. We start by determining how many samples we want from the image and then randomly select that many. We will sample 1/3 of the image, meaning we keep 2500 pixels.

```
k = 2500; % number of sparse samples
Atest2 = zeros(ny,nx);
perm = randperm(nx*ny);
ind = perm(1:k);
for j = 1:k
   Atest2(ind(j)) = -1;
end
imshow(Atest2), caxis([-1 0])
```

This code produces a matrix **Atest2** which is used to illustrate the pixels that are sampled from the image. The 2500 randomly chosen pixels are black, while the 5000 (remaining 2/3) that are white are not sampled.

We will follow in much the same was as we did for the signal in the previous lecture. The image is now represented in the Fourier domain using the two-dimensional DCT. The randomly chosen sampling points are treated like Kronecker delta functions and are responsible for producing the matrix leading to the underdetermined system.

```
Atest = zeros(ny,nx);
for j = 1:k
   Atest(ind(j)) = 1;
   Adel = reshape(idct2(Atest),nx*ny,1);
   ADelta(j,:) = Adel;
   Atest(ind(j)) = 0;
end

b = Abw(ind)';
```

The results are the matrix **Adelta** (which is $2500 \times 7500$) and the vector **b** (which is $2500 \times 1$). This leads us to the underdetermined linear system **Adelta***$\mathbf{y} = \mathbf{b}$. We again use cvx to perform the $L^1$ optimization.

```
n = nx*ny;
cvx_begin;
    variable y(n);
    minimize( norm(y,1) );
    subject to
        ADelta*y == b;
cvx_end;

Alow = uint8(dct2(reshape(y,ny,nx)));

figure(4)
subplot(2,1,1)
imshow(Abw)
subplot(2,1,2)
imshow(Alow)
```

Okay, that's not great... But, keep in mind that we already got rid of a lot of the sparsity in the image when we shrunk it down to a low-resolution image. In the final image of this document we compare (a) the original image, (b) $k = 4000$, (c) $k = 3000$, and (d) $k = 2500$.

**Sampling Strategy**

Sampling randomly is not an optimal way to recover images. In practice it is often desirable to apply a sampling mask in order to enhance the compressed sensing algorithm. That is, we know that the dominant Fourier modes

in images correspond to low-frequency content and that most high-frequency content can be ignored. Hence, we would like to sample in the frequency domain more heavily around the zero wavenumber ($k_x = k_y = 0$).

**Wavelets**

In this lecture and the previous one we demonstrated the compressed sensing algorithm with the Fourier transform. This is largely because I think we have an easier time understanding the Fourier transform. We have already seen that wavelets are very effective for storing images since they typically give a more sparse representation of the image. The issue with using wavelets is that the sampling has to be performed in a very different way. That is, with Fourier modes we sampled according to delta functions in the image space to create a global spanning of the Fourier space. Wavelets require something different since they are highly localized in both space and time. You need both global and noisy sampling modes to perform the task with wavelets in the most efficient manner.