EE369C, Autumn 2007
Medical Image Reconstruction Michael Lustig

Assignment 6

Due Tuesday Nov. 20

In this assignment we will explore some of the basic elements of Compressed sensing: Sparsity, Incoherency and the Sparsity based reconstruction.

1. Sparse Signals and Denoising To get an inuition, let's first explore the world of sparse 1D signals.

a) Sparse Signals Before we start with compressed sensing, we'll look at sparse signal denoising. There's a strong connection between CS and denoising. Here will attempt to denoise a sparse signal that is corrupted by random noise.

Generate a 1x128 vector, x, with 5 non-zero ([1:5]/5) coefficients and permute them randomly,

```
>> x = [[1:5]/5 zeros(1,128-5)];
>> x = x(randperm(128));
```

Add random gaussian noise with standard deviation $\sigma = 0.05$ to the signal, $y = x + n$

```
>> y = x + 0.05*randn(1,128)
```

Many approaches for denoising and regularization use the Tychonov penalty to estimate the signal from noisy data. Specifically, they try to solve:

$$\text{argmin } \frac{1}{2}||\hat{x} - y||_2^2 + \lambda \frac{1}{2}||\hat{x}||_2^2$$

This optimization trades the norm of the solution with data consistency. The nice thing about this approach that it has a closed form solution, and finding the minimum is a linear problem. Show that the solution for this problem is

$$\hat{x} = \frac{1}{1+\lambda} y$$

Observe what happens when we plot the result for $\lambda = \{0.01, 0.05, 0.1, 0.2\}$, and include the plot for $\lambda = 0.1$ in your report. Is the solution sparse?

b) Sparse Signals and the ℓ^1 Norm Instead of Tychonov regularization, which penalizes the ℓ^2 norm ($||x||_2 = \sqrt{\sum x_i^2}$), we will use the an ℓ^1 norm ($||x_1|| = \sum |x_i|$) penalized solution. Specifically we will solve:

$$\text{argmin } \frac{1}{2}||\hat{x} - y||^2 + \lambda ||\hat{x}||_1$$

It turns out that this is very easy to solve. Because the variables $\hat{x}_i$'s are independent we can minimize each of them seperately by solving argmin $\frac{1}{2}|\hat{x}_i - y_i|^2 + \lambda|\hat{x}_i|$. The solution to each $\hat{x}_i$ has a closed form. Show that the solution is

$$\hat{x} = \begin{cases} y + \lambda & \text{if} \quad y < -\lambda \\ 0 & \text{if} \quad |y| < \lambda \\ y - \lambda & \text{if} \quad y > \lambda \end{cases}$$

(Hint: look at the solutions for $y >> \lambda$, $y << -\lambda$ and $|y| < \lambda$). Write a function `SoftThresh` that accepts y and λ and returns $\hat{x}$. Plot it for $y \in [-10, 10]$ and $\lambda = 2$. The effect of this function is often referred to as soft-thresholding or shrinkage. Describe what happens when y is small compared to λ, and when y is large.

Apply `SoftThresh` to the noisy signal with $\lambda = \{0.01, 0.05, 0.1, 0.2\}$, and include the plot for $\lambda = 0.1$ with your report. Is the solution sparse?

c) Random Frequency Domain Sampling and Aliasing As we mentioned before, there is a strong connection between compressed sensing and denoising. We'll now explore this connection and the importance of incoherent sampling.

First, let's set up the undersampled data. To do so, compute the centered Fourier transform of the sparse signal, $X = Fx$ where F is a Fourier transform operator,

```
>> X = fftc(x);
```

In compressed sensing, we undersample the measurements. We measure a subset of k-space, $X_u = F_u x$ where F_u is a Fourier transform evaluated only at a subset of frequency domain samples. This is an underdetermined data set for which there is an infinite number of possible signals. However, we do know that the original signal is sparse, so there is hope we will be able to reconstruct it.

The theory of compressed sensing suggests random undersampling. To see why, we will look at equispaced undersampling and compare it to random undersampling. Undersample k-space by taking 32 equispaced samples. Compute the inverse Fourier transform, filling the missing data with zeroes, and multiply by 4 to correct for the fact that we have only 1/4 the samples,

```
>> Xu = zeros(1,128);
>> Xu(1:4:128) = X(1:4:128);
>> xu = ifftc(Xu)*4;
```

This is the minimum ℓ^2 norm solution (why?). Plot the absolute value of the result. Describe what you see. Will we be able to reconstruct the original signal from the result?

Now, undersample k-space by taking 32 samples at random. Compute the zero-filled inverse Fourier transform and multiply by 4 again,

```
>> Xr = zeros(1,128);
>> prm = randperm(128);
>> Xr(prm(1:32)) = X(prm(1:32));
>> xr = ifftc(Xr)*4;
```

Plot the absolute value, and describe the result. Will we be able to reconstruct the signal from the result? How does this resemble the denoising problem?

This is the important part, so say it out loud: **By random undersampling, we've turned the ill-conditioned problem into a sparse signal denoising problem**. However, the "noise" is not really noise, but incoherent aliasing that is contributed by the signal itself. Therefore, we might be able **EXACTLY** recover the sparse signal.

d) Reconstruction from Randomly Sampled Frequency Domain Data Inspired by the denoising example, we will add an ℓ^1 penalty and solve,

$$\text{argmin } \frac{1}{2}||F_u\hat{x} - y||_2^2 + \lambda|\hat{x}|_1$$

In this case, $\hat{x}$ is the estimated signal, $F_u\hat{x}$ is the undersampled Fourier transform of the estimate, and y are the samples of the Fourier transform that we have acquired. Now the variables are coupled through the Fourier transform, and there is no closed-form solution. Therefore we will solve it iteratively applying soft-thresholding and constraining data consistency. If $\hat{X} = F\hat{x}$, we initially set $\hat{X}_0 = y$, and implement the following iteration

1. Compute the inverse Fourier transform to get an estimate of the signal $\hat{x_i} = F^*\hat{X}_i$
2. Apply SoftThresh $\hat{x_i} = S(\hat{x_i}, \lambda)$ in the signal domain
3. Compute the Fourier transform $\hat{X}_i = F\hat{x_i}$
4. Enforce data consistency in the frequency domain

$$\hat{X}_{i+1}[j] = \begin{cases} \hat{X}_i[j] & \text{if } y[j] = 0 \\ y[j] & \text{otherwise} \end{cases}$$

5. Repeat until $||\hat{x}_{i+1} - \hat{x}_i|| < \epsilon$

This is a Projection Over Convex Sets (POCS) type algorithm. Apply the algorithm (at least 300 iterations) to the undersampled signal with $\lambda = \{0.01, 0.05, 0.1\}$ and plot the results.

To implement this in matlab, assume that the randomly sampled Fourier data is `Y`, with zeros for the non-acquired data. First initialize the estimate of the Fourier transform of the signal to be `X = Y`. The core of the iteration can then be written as

```
>> x = ifftc(X);
>> xst = SofthThresh(x,lambda);
>> X = fftc(xst);
>> X = X.*(Y==0) + Y;
```

You can do this in one line if you want. Note that you will need to modify `SoftThresh` to handle complex signals

$$S(x, \lambda) = \begin{cases} 0 & \text{if } |x| \leq \lambda \\ \frac{(|x|-\lambda)}{|x|}x & \text{if } |x| > \lambda \end{cases}$$

Make a plot of error between the true x and $\hat{x_i}$ as a function of the iteration number, plotting the result for each of the λs. It is really cool to see the evolution of the intermediate result. To plot the signal at each iteration use `drawnow` after the `plot` command.

Now, repeat the iterative reconstruction for the equispaced undersampled signal. What's wrong?

2. Sparsity of Medical Images Except for angiograms, most medical images are are generally not sparse (actually angiograms are also not that sparse, but that's another story). However, medical images can have a sparser representation in a transform domain, such as the wavelet domain.

Download Wavelet.tar.gz from the class website. Untar it to your working directory. Once extracted, a directory named @Wavelet will be created. This is a class (much like in C++) that implements the Wavelet transform. It uses functions from WaveLab written by David Donoho (`http://www-stat.stanford.edu/~wavelab/`). The implementation as a class allows overloading of operators and makes it easy to use. Here's an example how to compute a Daubechies wavelet transform of an image and reconstruct it again:

```
>> W = Wavelet; % defines a wavelet operator
>> im = phantom(256); % Shep-Logan CT Phantom
>> im_W = W*im; % computes the wavelet transform
>> im_rec = W'*im_W;  % computes the inverse wavelet transform
>> figure, subplot(1,3,1), imshow(im,[]), title('phantom image');
>> subplot(1,3,2), imshow(abs(im_W),[0,1]), title('The Wavelet coefficients');
>> subplot(1,3,3), imshow(abs(im_rec),[]), title('The reconstructed image');
```

We will now evaluate the sparse approximation of a very pretty axial T_2 weighted FSE image. Download `brain.mat` from the class website, and load it into matlab. The image matrix is `im`. Compute the Wavelet transform of the image. Plot the image of the wavelet transform coefficients using the function `imshowWAV.m` from the class website. Wavelet coefficients represent both space and spatial frequency information. Each band of wavelet coefficients represent a scale (frequency band) of the image. The location of the wavelet coefficient within the band represent its location in space. What you see are edges of the image at different resolutions and directions.

Threshold the wavelet coefficients retaining only the largest 20% of the coefficients. You can threshold `im_W` for the largest faction `f` of the coefficients with

```
>> m = sort(abs(im_W(:)),'descend');
>> ndx = floor(length(m)*f);
>> thresh = m(ndx);
>> im_W_th  = im_W .* (abs(im_W) > thr);
```

Plot the masked wavelet coefficients. What has been thresholded?

Reconstruct the image and display it in your report. Compare it to the original image qualitatively and by computing the difference image. What has been removed? Examine the results when you retain the largest 12.5%, 10%, 5% and 2.5% of the coefficients (don't include this in your report). What, in your opinion, is the sparsity of the image? Provide a reconstruction and difference image to support your argument.

The wavelet transform sparsifies the brain image, and concentrates the "important" image energy into a subset of the coefficients. This helps us denoise the image by thresholding the coefficients which contain mostly noise!

3) Compressed Sensing Reconsruction We'll now explore a 2D compressed sensing reconstruction from undersampled data. Load the file `brain.mat` into matlab again.

a) Non-Uniform Random Sampling An important aspect of random frequency-domain sampling is matching the power spectrum of the image. If the energy is concentrated in lower spatial frequencies, more samples should be allocated there. We constructed two random 3-fold undersampling patterns for you. The random patterns are in the variables `mask_unif` and `mask_vardens` were drawn from probability distribution functions (PDF) given by the variables: `pdf_unif` and `pdf_vardens` respectively.

Compute the 2D Fourier transform of the image. Multiply by the uniform mask, divide by the appropriate PDF, and compute the zero-filled Fourier transform.

```
>> M = fft2c(im);
>> Mu = (M.*mask_unif)./pdf_unif;
>> imu = ifft2c(Mu);
```

Display the image and the difference image compared to original image. Is the aliasing white random noise? Repeat for the variable density mask. What happened now? Both use a similar number of samples, but which gives you a better reconstruction?

b) Reconstruction from Random Sampled k-Space Data Implement the POCS algorithm described earlier for 2D images. Add another step of computing the wavelet transform before the soft-thresholding and the inverse wavelet transform after the soft-thresholding.

Reconstruct the images from the uniform and variable density undersampled data. First get an idea of reasonable values of λ by examining what would be thresholded. You can do this using

```
>> imuW = W*imu;
>> imshow(abs(imuW)>lambda,[]);
```

since `imshowWav` scales different wavelet scales differently. You want a significant number of coefficients to be below λ, but not all of them.

Start with the variable density data, and experiment with several values of λ. You should only need about 20 iterations, but start with fewer while you convince yourself it is working! Compare the result after soft-thresholding to a zero-filled density compensated reconstruction, the original image, and a the original image soft-thresholded. As an initial image to the POCS, use a zero-filled density compensated reconstruction, it will converge faster. Show the image, and the difference image for the λ you find the most effective.

Then try the uniform density data. Run for at least 50-100 iterations, since this converges slowly. If you want to speed up the convergence, start with a relatively large λ so that the recon will converge rapidly. Then, decrease λ using the previous recon as an initial image. For example, you might divide lambda by two every 10 or 20 iterations. Show the image, and the difference image for the λ (or the final λ if you use a sequence) that you find the most effective. Don't spend too much time on this, the point should be clear by now.

I hope you enjoyed this excercise – Miki.