EE224 Signals and Systems I
Lab 12: Compressive Sensing

1 Brief Introduction

According the Nyquist-Shannon sampling theorem, in order to guarantee you are able to reconstruct a signal from its samples the sampling rate must be at least twice the maximum frequency contained in the signals frequency spectrum.

While this is correct in general, you will find that depending on the spectrum of a specific type of signal you may be able to sample at less than the Nyquist rate. For example, band-pass signals can be sampled at lower rates because they have gaps allowing spectrum shifting (a result of sampling) without distorting aliasing.

Another class of signals (the subject of on-going research) is those that are "sparse" in a particular domain and can be reconstructed from random sampling, with far fewer samples than would be required by the Nyquist rate. You should note that a very large variety of naturally occurring signals (such as MRI scans and images of every day scenes) follow this pattern.

For our purposes we will consider signals that are "sparse" in the normal frequency domain. I.e. they have DFTs (Discrete Fourier Transforms) which have mostly zero values and only a few sinusoidal frequencies make up the signal.

Compressive Sensing (CS), as the name implies involves both compression and sensing, but at the same time. We all deal with very many compressed "signals" on a daily basis. Most images are stored in jpeg format or something similar, while most music files have undergone mp3 compression etc..

Essentially these "signals" can be represented by a number of coefficients in some domain (you might think of these like the frequency domain representation of time domain signals) where a smaller number of the coefficients take on relatively large values, while most of the others are very small and relatively insignificant. Elimination of these many smaller elements is in a very general sense compression.

Usually a signal is gathered at a very high data rate and only later compressed. Overall, compressive sensing is achieving both goals at the same time; only sensing the minimal amount of data necessary to re-create a compressed signal. The implications and applications of such processes are many and as stated earlier the subject of ongoing research.

For a very nice introduction to what we will generally be doing in this lab see:
http://www.acm.caltech.edu/l1magic/examples.html

2 Preparation

You will need to download the MATLAB functions:

```
sparseFreqSigGen.m
randSamp.m
L1min.m
fftPlot.m
unifSamp.m
lpfRecon.m
```

These functions will need to be placed into the same directory (folder) as you will create your own file for this lab. You may name this directory any name you see fit. You also need to have `CVX` installed. Download the `cvx.zip` file from this page http://cvxr.com/cvx/download/ and follow the "Installation Instructions" on the same page to install the `CVX` package.

Step 1: Signal Generation

Once you have this done, create your own new blank M-file. To do so, (with MATLAB open) click File, New, Blank M file. The first thing you will do is use sparseFreqSigGen.m to generate a signal that is sparse in the frequency domain. In order to understand how to do this, type:

```
help sparseFreqSigGen
```

into your MATLAB Command Window and press Enter. This is a very useful way to find information on the usage of all standard MATLAB functions, and any custom functions with written help references (like the ones you're using here). Also, you have direct access to these function scripts! Simply open them to take a look at the code inside (recommended). Just be sure not to edit anything.

Once you are confident you know what to do, generate a signal with 6 frequencies and time duration of 1 second, call the signal `x`. Make sure that the signal has no frequencies higher than 200 Hz[1]

Also, ensure that the signal you generate will include at least 1 complete period. (hint: since your signal will be a sum of pure sinusoids at the frequencies you specify, what must the lowest frequency be in order to have at most a period 1 second?)

You will need to decide on a "sampling frequency" `fs` in order to allow MATLAB to display the signal. After all, MATLAB like all computer programs cannot display or store a continuous signal with an infinite number of points. All signals used in MATLAB are discrete. What you really have is `samples` of a continuous signal. Obviously, you Will need to ensure that `fs` is at least twice your highest frequency. (For simplicity, 1000 may be a good choice)

If the question comes to mind, "What are we doing sampling a signal that is itself simply a vector of samples?" Consider the fact that this is an illustrative simulation. In a real application, of course samples would be being taken from the real world, and re-constructed into a discrete signal. We here, start with such a discrete signal and will show that even if we had only taken a few random samples, we could have come out with the same signal as if we had sampled at a much higher rate!

Once you have generated a proper signal `x` write code into your M-file to plot it. You should plot the signal vs time. Verify that you are correct with your Lab instructor before continuing.

Step 2: Uniform Sampling

Here we will uniformly sample our signal. To do this we will use the unifSamp.m. Reference the code using "`help unifSamp`" or directly by opening the file for syntax reference.

You will need to decide what sampling rate is necessary in order to properly reconstruct your original signal from your samples. (hint: we limited the maximum frequency in our 1 second signal to 200 Hz for a reason. We have 1000 samples (assuming you did not decide to use something else at this point, in which case you'll need to figure out what else needs to change!) representing our signal in MATLAB, of which we need to sub-sample at a given rate.)

The Number of samples we take will essentially be our sampling (sub-sampling) frequency (samples/second). Why is this so? Would it be true with a signal length other than 1 second?

The code will convert our chosen sampling rate, along with the x storage sampling rate, into a sampling interval in samples, i.e. take every 10th sample, or every 5th etc from the `x` vector/signal. Keeping this in mind, what is the maximum sampling rate we can have with the length signal we have (i.e. the set of samples we are going to sub-sample)?

How does this number relate to the maximum frequency from Step 1? Why is this important? Explain this relationship between signal length (in original samples), sampling/(sub-sampling) rate, and maximum frequency and check with your lab instructor before moving on.

[1]Note: this is NOT important for Compressive Sensing purposes. The reason we do this here will be explained in Step 2. You WILL be free to generate signals of arbitrary length and frequency content later in this lab!

- Create a plot showing the original signal plotted as a solid line, with the samples you've taken plotted at their proper time instants superimposed as circles or dots of another color.
- Also create a plot showing the spectrum of both the original signal and the sampled version. Use `fftPlot.m` to make the results of `fft()` look like continuous time spectra. Remember, you should see `copies` of the original spectrum scaled and shifted by your sampling frequency in the sampled version. Double check everything is working correctly!

Step 3: Nyquist Reconstruction (`sinc` interpolation)

Now we will reconstruct the original signal from your uniform samples taken in Step 2. We will use the LPFrecon function in order to do this. There is not much to do in this step except see if you can correctly reconstruct. You should refer to `help LPFrecon` to see how to use the function, and verify that you're getting the proper results with your lab instructor before going to Step 4.

- Create a plot showing your reconstructed signal, the original signal, and the error (absolute value of the difference between them) in a 3X1 subplot figure.

Step 4: Random Under-Sampling

Now we will randomly sample our signal. To do so we will use `randSamp.m`. (Remember to use "help " in your Command Window for further help, or open the function script itself!)In order to do so, you will enter your signal `x` along with a desired number of samples/observations into the function.

A nice guideline for the number of samples is that it should be on the order of $Q \cdot \log(N)$, where Q represents the number of "significant" elements in the sparse domain. In this case Q will be 2*(the number of sinusoids) since each sinusoid corresponds to a negative and positive component in the frequency domain, and N is the length of the desired output signal in samples (in our case equal the length of our input signal in samples). Of course you will need to round the result of $\log(N)$ to an integer! Do this with MATLAB not by hand.

The `randSamp` function will return 3 things for you. First `y` which is a vector of only the observations/samples, then `y2` which is the same as `y` except it has zeros where no sample was taken, and A which is a matrix used in a following function which is beyond the scope of this lab. Type the help command as before to see the exact usage syntax. (As always, feel free to look at the code and ask your lab instructor if you're curious!)

- Produce a plot showing the original signal plotted as a solid line, with the samples you've taken plotted at their proper time instants superimposed as circles of another color.
- Also, create a plot showing the spectrum of the randomly sampled signal (you'll want to use the version with zeros at un-sampled points here). Compare this spectrum to that of the original signal. You should likely be able to clearly see most if not all of the original spectrum standing out above incoherent "noise"/aliasing produced by the random sampling. This is why it is important that the under-sampling be `random` and the signal sparse in some domain.
- Finally, explain why you could not possibly use `sinc` interpolation to reconstruct the signal in this case.

Again, make sure to verify you're getting correct results before continuing.

Step 5: Uniform Under-Sampling

Here you will see, if you have not already predicted, why we cannot under-sample uniformly in general. Sample the original signal uniformly, ensuring that you will end up with a similar total number of samples as with the random sampling.

- Produce a plot of the resulting spectrum and explain why it looks as it does. Of course verify it is correct if you

Step 6: CS reconstruction from random samples

This step involves finding a signal with the minimum L_1 Norm in the sparse domain which would correspond to the observed samples. In our case, we are looking for the signal with the minimum L_1 Norm in frequency domain which would correspond to a time domain signal passing through the points we have sampled in our vector of samples `y`. WHY this works is very far beyond the scope of this lab.

You will be using the L1min function which takes as inputs the vector of observations `y` and the matrix `A` which was an output of the randSamp function. You will receive as outputs both the FFT `Xhat` and signal `xhat` reconstructed from random samples using the L_1 minimization process.

As before, verify that you are getting correct results and create appropriate plots before proceeding.

Step 7: Plotting/Displaying Outputs/Results

Now you will make sure you've got good plots of all your data. Make sure you've got the following:

(a) A plot showing the original signal vs time (you should already have this from Step 1)

(b) A plot showing BOTH the original signal vs time and your uniformly samples points. Your original signal should be a solid line while the samples should appear as isolated circles of a different color superimposed onto the signal.

(c) A plot showing the original, sinc-reconstructed, and error signals vs time in subplot format. See Step 3.

(d) A plot showing the original vs uniformly sampled spectrums.

(e) A plot showing BOTH the original signal vs time and the randomly sampled points, following the format of (II).

(f) A plot showing the spectrum of the original signal using the fftPlot function vs the spectrum of the randomly sampled signal.

(g) A plot showing the original `x`, the output signal `xhat` vs time, and the `ERROR=|x-xhat|` vs time.

(h) A plot showing the output FFT `Xhat` function vs the original (FFT of `x`)

ALL plot axes should be labeled properly (with units where applicable) and the plots themselves given appropriate titles! Verify with your instructor.

Step 8: Experimentation

Now you should try and repeat the previous steps, only now you should experiment with differing numbers of frequency components being present in your signal, as well as higher frequencies and different numbers of observations.

Note how you can use CS methods for signals which would be impossible to even satisfy a Nyquist sampling rate with!

You should find that the number of samples you require for accurate reconstruction depends on ONLY the number of components present in the sparse domain, and NOT at all on the maximum frequency of the components in that domain. (Remember to keep a proper sampling frequency for display in your plots and storage in MATLAB)

Try and verify this to yourself. Ask your lab instructor to help you if you have any questions.