Math Bootcamp - Fourier Transform Problem set

1. Orthogonality of Functions

Show that for two different positive integers n and m, $\sin(nx)$ and $\sin(mx)$ are orthogonal on $x \in [0, 2\pi]$. You can either do this analytically or computationally in MATLAB with a handful of m and n values. If you want to do this analytically, you may want to look up trigonometric product identities and integrals.

2. Do it yourself Fourier Series

We showed in class that you can build an arbitrary periodic function using a series of sin and cos functions. Here, we will use a sum of sin waves to recreate a square wave. In MATLAB, create a square wave function, $f(x)$, defined for $x = [1 : 1000]$ which has $f(x) = 1$ for $x \in [1 : 500]$ and $f(x) = -1$ for $x \in [501 : 1000]$. The sinusoidal basis functions are of the form:

$$s_n(x) = \sin(\frac{2\pi n}{1000}x)$$

The coefficients in front of $s_n(x)$ is given by the dot product,

$$b_n = \frac{2}{L}s_n(x) \cdot f(x)$$

Find b_1, the coefficients for the first sin wave. Plot both the $f(x)$ and reconstructed function $g_1(x) = b_1 s_1(x)$ on the same plot. Calculate the error between $f(x)$ and $g_1(x)$, that is, find the sum of $(f(x) - g_1(x))^2$ between 1 and 1000.

Calculate the coefficient of $f(x)$ on to $s_2(x)$. The new reconstructed function is now $g_2(x) = b_1 s_1(x) + b_2 s_2(x)$. Again, plot both $f(x)$ and $g(x)$ on the same plot and calculate the error.

Do this for all value up to n=100. Show that $g_n(x)$ approaches $f(x)$ and that the error is proportional to $1/n$.

Compare your projection values qualitatively to MATLAB's discrete Fourier transform by looking at the first 100 terms from the fft function on $f(x)$.

3. Sounds Engineering

Matlab can play sounds using the soundsc function. Read the documentation for the soundsc and try playing the sound from the `gong.mat` file. You can increase or decrease the sampling rate by manually changing the playback rate, `Fs`.

We will analyze the sound from the `gong.mat` file. Load the `gong.mat` file which is built into MATLAB. Take the first two seconds of the recording and calculate the power spectrum of the sound. Plot the power spectrum, that is, $|F|^2$ as a function of frequency. What is the dominant frequency of the sound? Also, plot the real component of the Fourier Transform as a function of frequency. The frequency range of this plot should be centered at 0 Hz by using the `fftshift` function. The `fftshift` shifts the output of the `fft` function so that the frequency range goes like $-N/2 : 1 : (N/2 - 1)$ and the 0 frequency is in the middle.

In the Fourier domain, we are able to adjust different frequencies of the sound independently. In this example, we will turn up the bass by amplifying signal in certain frequencies in the Fourier domain. Use the inverse Fourier Transform function, `ifft`, to show that you can recreate the original sound from the Fourier transform. From the output of the `fft`, identify which values correspond to frequencies less than 2000 Hz. Amplify all of these values by multiplying them by 11. Then use the `ifft` and play the new sound using `soundsc`.

It may help to make a mask here. A mask is a vector with the same length as the frequency range, but with a 0 for all frequencies larger than 2000Hz and 1 for all frequencies less than 2000Hz. You can then modify the fft output by using logical indexing.

Repeat the exercise, but now take all frequencies less than 2000 Hz and set them to zero. Play around with amplifying and diminishing different frequencies of the signal.

4. Reconstruction keeping phase versus keeping amplitudes

Download the file `salgado.zip` from the wiki, unzip it, and load the resulting `salgado.mat` file. This will create a variable A, which you can plot

```
>> imagesc(A); colormap(gray);
```

It is a train station in Bombay.

Use `fA = fft2(A)` to compute the two-dimensional Fourier transform of A. For each element of the resulting `fA` matrix, find its amplitude r using `abs.m` and find its phase angle θ using `angle.m`. Each component can thus be represented as $re^{i\theta}$ (where, for each element, you use the corresponding values of r and θ that you found). Create a modified fA that is equal to the original fA, but in which each amplitude has been replaced by a random number between 0 and 1. Use `ifft2.m` to compute A_{mod}, the inverse Foruier transform of your modified fA, and look at its real part using

```
>> imagesc(real(Amod)); colormap(gray);
```

Can you still more or less recognize the original image? Now repeat the previous bullet point, but this time keeping the amplitudes as the originals, and replacing each phase by a random number between 0 and 2π. Can you still more or less recognize the original image? Speculate on why you think randomizing amplitudes and randomizing phases has such a different effect.