32.10.1: Fourier Analysis in Matlab

Fourier analysis encompasses a vast spectrum of mathematics with parts that, at first glance, may appear quite different. In the sciences and engineering the process of decomposing a function into simpler pieces is often called an analysis. The corresponding operation of rebuilding the function from these pieces is known as synthesis. In this context the term Fourier synthesis describes the act of rebuilding and the term Fourier analysis describes the process of breaking the function into a sum of simpler pieces. In mathematics, the term Fourier analysis often refers to the study of both operations.

Introduction

In Fourier analysis, the term Fourier transform often refers to the process that decomposes a given function into the basic pieces. This process results in another function that describes how much of each basic piece are in the original function. It is common practice to also use the term Fourier transform to refer to this function. However, the transform is often given a more specific name depending upon the domain and other properties of the function being transformed, as elaborated below. Moreover, the original concept of Fourier analysis has been extended over time to apply to more and more abstract and general situations and the general field is often known as harmonic analysis.

(Continuous) Fourier transform

Most often, the unqualified term Fourier transform refers to the transform of functions of a continuous real argument, such as time (t). In this case the Fourier transform describes a function $f(t)$ in terms of basic complex exponentials of various frequencies. In terms of ordinary frequency ν, the Fourier transform is given by the complex number.

Evaluating this quantity for all values of ν produces the frequency-domain function.

Matlab and the FFT

Matlab's FFT function is an effective tool for computing the discrete Fourier transform of a signal.The following code examples will help you to understand the details of using the FFT function.

Example **1**

The typical syntax for computing the FFT of a signal is FFT(x,N) where x is the signal, x[n], you wish to transform, and N is the number of points in the FFT. N must be at least

as large as the number of samples in x[n]. To demonstrate the effect of changing the value of N,

synthesize a cosine with 30 samples at 10 samples per period.

- n = [0:29];
- x = cos(2*pi*n/10);

Define 3 different values for N. Then take the transform of x[n] for each of the 3 values that were

defined. The abs function find the magnitude of the transform, as we are not concerned with distinguishing between real and imaginary components.

- N1 = 64;
- N2 = 128;
- N3 = 256;
- X1 = abs(fft(x,N1));
- X2 = abs(fft(x,N2));
- X3 = abs(fft(x,N3));

The frequency scale begins at 0 and extends to N - 1 for an N-point FFT. We then normalize the scale so that it extends from 0 to 1 - 1/N

- F1 = [0 : N1 - 1]/N1;
- F2 = [0 : N2 - 1]/N2;
- F3 = [0 : N3 - 1]/N3;

Plot each of the transforms one above the other

- subplot(3,1,1)
- plot(F1,X1,'-x'),title('N = 64'),axis([0 1 0 20])
- subplot(3,1,2)
- plot(F2,X2,'-x'),title('N = 128'),axis([0 1 0 20])
- subplot(3,1,3)
- plot(F3,X3,'-x'),title('N = 256'),axis([0 1 0 20])

Upon examining the plot one can see that each of the transforms adheres to the same shape, differing only in the number of samples used to approximate that shape. What happens if N is the same as the number of samples in x[n]? To find out, set N1 = 30. What does the resulting plot look like? Why does it look like this?

Example 2

In the last example the length of x[n] was limited to 3 periods in length. Now, let's choose a large value for N (for a transform with many points), and vary the number of repetitions of the fundamental period.

- n = [0:29];
- x1 = cos(2*pi*n/10); % 3 periods
- x2 = [x1 x1]; % 6 periods
- x3 = [x1 x1 x1]; % 9 periods

- N = 2048;
- X1 = abs(fft(x1,N));
- X2 = abs(fft(x2,N));
- X3 = abs(fft(x3,N));
- F = [0:N-1]/N;

- subplot(3,1,1)
- plot(F,X1),title('3 periods'),axis([0 1 0 50])
- subplot(3,1,2)
- plot(F,X2),title('6 periods'),axis([0 1 0 50])
- subplot(3,1,3)
- plot(F,X3),title('9 periods'),axis([0 1 0 50])

The previous code will produce three plots. The first plot, the transform of 3 periods of a cosine, looks like the magnitude of 2 sincs with the center of the first sinc at 0.1fs and the second at 0.9fs.

The second plot also has a sinc-like appearance, but its frequency is higher and it has a larger magnitude at 0:1fs and 0:9fs. Similarly, the third plot has a larger sinc frequency and magnitude.

As x[n] is extended to a large number of periods, the sincs will begin to look more and more like impulses.

But I thought a sinusoid transformed to an impulse, why do we have sincs in the frequency domain? When the FFT is computed with an N larger than the number of samples in x[n], it fills in the samples after x[n] with zeros. Example 2 had an x[n] that was 30 samples long, but the FFT had an N = 2048. When Matlab computes the FFT, it automatically fills the spaces from n = 30 to n = 2047 with zeros. This is like taking a sinusoid and multiplying it with a rectangular box of length 30. A multiplication of a box and a sinusoid in the time domain should result in the convolution of a sinc with impulses in the frequency domain. Furthermore, increasing the width of the box in the time domain should increase the frequency of the sinc in the frequency domain. The previous Matlab experiment supports this conclusion.

Identifying Signals in noise with FFT

A common use of Fourier transforms is to find the frequency components of a signal buried in a noisy time domain signal. Consider data sampled at 1000 Hz. Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and 120 Hz sinusoid of amplitude 1 and corrupt it with some zero-mean random noise:

- Fs = 1000; % Sampling frequency
- T = 1/Fs; % Sample time
- L = 1000; % Length of signal
- t = (0:L-1)*T; % Time vector

- x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
  % Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
- y = x + 2*randn(size(t)); % Sinusoids plus noise
- plot(Fs*t(1:50),y(1:50))
- title('Signal Corrupted with Zero-Mean Random Noise')
- xlabel('time (milliseconds)')

It is difficult to identify the frequency components by looking at the original signal. Converting to the frequency domain, the discrete Fourier transform of the noisy signal y is found by taking the fast Fourier transform (FFT):

- NFFT = 2^nextpow2(L); % Next power of 2 from length of y
- Y = fft(y,NFFT)/L;
- f = Fs/2*linspace(0,1,NFFT/2+1); % Plot single-sided amplitude spectrum.
- plot(f,2*abs(Y(1:NFFT/2+1)))
- title('Single-Sided Amplitude Spectrum of y(t)')
- xlabel('Frequency (Hz)')
- ylabel('|Y(f)|')

The main reason the amplitudes are not exactly at 0.7 and 1 is because of the noise. Several executions of this code (including recomputation of y) will produce different approximations to 0.7 and 1. The other reason is that you have a finite length signal. Increasing L from 1000 to 10000 in the example above will produce much better approximations on average.

Outside Links

- http://www.chem.uoa.gr/applets/Apple...FourAnal2.html

Contributors and Attributions