Mathematical Tools for Neuroscience (NEU 314)
Princeton University, Spring 2016
Jonathan Pillow

Lecture 23: Fourier Transform, Convolution Theorem, and Linear Dynamical Systems

April 28, 2016.

Discrete Fourier Transform (DFT)

We will focus on the discrete Fourier transform, which applies to discretely sampled signals (i.e., vectors).

Linear algebra provides a simple way to think about the Fourier transform: it is simply a change of basis, specifically a mapping from the time domain to a representation in terms of a weighted combination of sinusoids of different frequencies. The discrete Fourier transform is therefore equivalent to multiplying by an orthogonal (or "unitary", which is the same concept when the entries are complex-valued) matrix[1].

For a vector of length N, the matrix that performs the DFT (i.e., that maps it to a basis of sinusoids) is an $N \times N$ matrix. The k'th row of this matrix is given by $\exp(-2\pi ikt)$, for $k \in [0, ..., N-1]$ (where we assume indexing starts at 0 instead of 1), and t is a row vector `t=0:N-1;`. Recall that $\exp(i\theta) = cos(\theta) + i\sin(\theta)$, so this gives us a compact way to represent the signal with a linear superposition of sines and cosines.

The first row of the DFT matrix is all ones (since $\exp(0) = 1$), and so the first element of the DFT corresponds to the sum of the elements of the signal. It is often known as the "DC component". The next row is a complex sinusoid that completes one cycle over the length of the signal, and each subsequent row has a frequency that is an integer multiple of this "fundamental" frequency.

Fast Fourier Transform. Usually, multiplying a vector by an $N \times N$ matrix requires $O(N^2)$ operations (the matrix itself has N^2 entries, each of which must be multiplied with some component of the vector). But an amazing fact about the DFT is that there is a *fast* algorithm for carrying it out that requires only $O(N \log N)$ operations. This algorithm is known as the *fast Fourier transform* (FFT), which can be carried out with '`fft`' in Matlab. For a signal of length $N = 100,000$, it would take nearly $8,700$ times longer to compute the DFT using matrix multiplication than it does with FFT algorithm. Gilbert Strang of MIT once called the FFT the ""the most important numerical algorithm of our lifetime", and it is ubiquitous in engineering and signal processing applications.

The *inverse Fourier transform* maps the representation of a signal in terms of a weighted combination of sinusoids back into the "standard" representation in terms of along a line in space or time. Since the DFT is given by matrix multiplication with an orthogonal matrix, the inverse Fourier transform is simply multiplication by the transpose of this matrix. In Matlab you can get

[1]Note that the standard definition of the DFT does not actually correspond to a unitary transform because the rows of the matrix are not unit vectors. Typically they have Euclidean norm $\sqrt{N}$. But if we divided each row by $\sqrt{N}$ we would then have a unitary matrix

compute the inverse Fourier transform with `ifft`. So `iff(fft(x))` leaves a vector x unchanged, and is equivalent to `M' * M * x` in Matlab, where M is the DFT matrix defined above (with rows normalized to be unit vectors).

Fourier Power Spectrum

One of the things we often want to do with signals recorded in the brain is to analyze the frequencies they contain. To do so, we often we wish to throw away phase information (i.e., how much "sine" vs. "cosine") and just find out the total (squared) weight of each frequency. If the j'th Fourier component is $a + ib$, the Fourier power at that frequency is the squared *modulus* $|a + ib| = a^2 + b^2$.

To compute the Fourier power spectrum of a signal `x`, we can simply take its FFT and then take its modulus squared: `xpow = abs(fft(x)).^2;`

Note that we need only plot the Fourier power for the components from 0 up to $N/2$ because the frequencies above $N/2$ are *complex conjugates* of the components between 1 and $N/2$, so they have the same power.

Beware of bumps in the power spectrum at a frequency of 60 Hz! (They may reflect the frequency of the alternating electric current in the wall, as opposed to anything meaningful going on in the brain.)

Convolution theorem

As we saw in the previous lecture, a linear shift-invariant (LSI) system maps sinusoids to sinusoids of the same frequency, but can alter the amplitude and phase. A convolution with a linear filter (i.e., a linear shift-invariant system) can be therefore be understood entirely in terms of its effects on sinusoids of each frequency. The effect on each sinusoid is actually determined by the filter's Fourier transform.

The famous *convolution theorem* formalizes this point. It says that convolutions can be carried out (very efficiently) via pointwise multiplication in the Fourier domain. Formally, it says that the convolution $x * w$ of a signal x with a filter w can be computed by taking the inverse Fourier transform of $\tilde{x} \circ \tilde{w}$, where $\tilde{x}$ and $\tilde{w}$ are the Fourier transforms of x and w, respectively, and $\circ$ denotes the component-wise product (also known as Hadamard product, or '`.*`' in Matlab).

In Matlab, we can express the convolution theorem as telling us that the convolution of a x with a filter w can be carried out `ifft(fft(x).*fft(w,N))`, where `N` is the length of the signal x, and w is assumed to be length N or shorter.[2]

[2]Note however that this will produce a convolution with *circular* boundary conditions (meaning the right edge of the signal connects to its left edge) which differs from the boundary conditions offered in `conv2`.

Linear Dynamics (A brief introduction to)

A linear dynamical system is characterized by a linear ordinary differential equation:

$$\frac{dx}{dt} = ax$$

which is sometimes written

$$\dot{x} = ax,$$

where $\dot{x}$ denotes the temporal derivative of the signal x.

A linear ODE is a prime example of a linear shift-invariant system! This means it is characterized entirely by its impulse response function, which is $\exp(at)$, for an impulse (or "delta" function) of size 1. If initialized at x_0, the solution to the differential equation is $x(t) = x_0 \exp(at)$.

If x is a scalar (1-dimensional) signal, then there only three possible behaviors of a linear ODE:

- if $a > 0$: runaway (exponential) growth. The system is known as *unstable*
- if $a = 0$: the solution is constant in time. (The derivative is zero, so it has to be!)
- if $a < 0$: exponential decay. The system has a *stable equilibrium* at 0.

If x is vector valued, we might instead write

$$\dot{x} = Ax$$

where A is a matrix. It turns out the system still has solution that can be written $\exp(At)x_0$, where x_0 is the (vector) initial condition and exp is the *matrix exponential* evaluated at the matrix At. The behavior of this system is now determined by the eigenvalues of A. If the real part of these eigenvalues is positive, zero, or negative, then we obtain runaway grown, constant, or exponential decay along the direction corresponding to the eigenvectors of A, just like above. If the eigenvalues have a nonzero complex part, then it will also have sinusoidal oscillations. The behavior of a linear dynamical system can therefore be entirely understood in terms of the real and complex parts of its eigenvalues. In one dimension, we can only get exponential growth or decay, but in higher dimensions we can get growth and decay as well as oscillations.