Statistics, Data Analysis, and Machine Learning for Physicists
Timothy Brandt
Spring 2020

Lecture 18

Time series data are common. We saw an example of one sort of time series data in Homework 1. You observed something with a detector and recorded occasional events; these events occurred as a Poisson process. In that case, your set of observables was the list of times t_i at which you recorded events, and perhaps also one or more properties x_i of those events. We could write down the likelihood of observing all of the events that I saw at the times when I observed them. Taking λ_i to be the rate of events at time t_i, with i running over all possible times, the likelihood is

$$\mathcal{L} = \prod_{!\text{event}} \exp\left[-\lambda_i\right] \prod_{\text{events } i} \lambda_i \exp\left[-\lambda_i\right] \tag{1}$$

$$= \exp\left[-\langle N_{\text{events}}\rangle\right] \prod_{\text{events } i} \lambda_i. \tag{2}$$

Now, if I also want to include information on the properties of the events that I saw, I need to multiply the likelihood (the probability that I saw something) times the probability that my events had the specific properties that I observed. You would have

$$\mathcal{L} = \exp\left[-\langle N_{\text{events}}\rangle\right] \prod_{\text{events } i} \lambda_i p(x_i). \tag{3}$$

Some time series data are like that, and this likelihood approach can be a powerful tool. Other times, we measure a stream of data. Maybe it's a time-varying electric field, maybe it's the light curve (brightness vs. time) of a star or a supernova, maybe it's a sequence of velocity measurements. These are real-valued data that could be measured continuously, but that in practice must usually be sampled discretely. They are certainly not described by Poisson statistics, so the likelihood analysis above will not apply.

Data sets like this are common in physics, and a standard way to analyze them is with Fourier techniques. This class will be an overview (hopefully a review to many of you) of the Fourier transform and its applications. In the next couple of classes we'll go into other techniques, like a wavelet transform and other methods to analyze periodic time series. We'll also cover some signal processing, which would get a lot of attention in an engineering program. You often want to filter your data to remove noise that isn't white (we'll discuss what this means, too). These topics may be especially relevant to those of you working in more experimental labs, though I suspect that you will have seen a lot of this before.

For any continuous function over a finite interval, we can approximate it to arbitrary accuracy using polynomials (the Weierstrauss theorem). In practice, we commonly use low-order polynomials to provide local approximations to functions. This is the basis for linearization and for many numerical integration schemes, since polynomials can be integrated analytically. In Fourier analysis, we use a similar theorem (the Dirichlet theorem, though the conditions aren't quite as clean): any continuous, periodic function may be approximated to arbitrary accuracy using a sum of sines and cosines. In the polynomial approximation case, we didn't usually bother plotting the coefficients of all of the different polynomial terms. In Fourier analysis, plotting the amplitudes (and possibly phases) of the various sinusoids is often the point–we are looking for the particular frequency or frequencies that are most prominent in the data.

The Fourier transform of a function $h(t)$ is defined as

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt. \tag{4}$$

It is the integral of the function times a sinusoid of frequency f (you can also write it in terms of angular frequency $\omega = f/2\pi$, but then you need to worry about factors of 2π in the inverse transform). $H(f)$ will, in general, be complex. If I write

$$H(f) = |H(f)|e^{i\phi}, \tag{5}$$

then the amplitude $|H(f)|$ is the power at frequency f and ϕ is the phase of the sinusoid that gives that power. The inverse of the Fourier transform is given by

$$h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi ift}df. \tag{6}$$

I will not prove that Equations (4) and (6) are inverses of one another; there are a few tricks involved.

There are a few important theorems about the Fourier transform that are very useful in data analysis and signal processing:

1. The **convolution theorem**:

$$g \otimes h = G(f)H(f) \tag{7}$$

   The full convolution costs n^2 operations, where n is the number of data points. A fast Fourier transform can be done in $n \log n$ steps, so applying a convolution by an FFT, multiplication, and inverse FFT can be faster than just convolving. You can do this with `scipy.signal.fftconvolve`.

2. The **correlation theorem**: defining

$$\mathrm{Corr}(g, h)(t) = \int_{-\infty}^{\infty} g(t + \tau)h(\tau)d\tau \tag{8}$$

   then

$$\mathrm{Corr}(g, h)(t) = G(f)H(-f). \tag{9}$$

   If $h(t)$ is real, then $H(-f) = H^*(f)$, where $*$ denotes the complex conjugate. If you want to compute the autocorrelation of a function, the result is just

$$\mathrm{Corr}(g, g)(t) = |G(f)|^2, \tag{10}$$

   the power at the frequency corresponding to the adopted offset.

3. Parseval's theorem, that the total power is the same in the frequency domain and the time domain:

$$\int_{-\infty}^{\infty} |h(t)|^2 dt = \int_{-\infty}^{\infty} |H(f)|^2 df. \tag{11}$$

   If we don't distinguish between positive and negative frequencies, then we define the power spectral density to be

$$P(f) = |H(f)|^2 + |H(-f)|^2. \tag{12}$$

   If $h(t)$ is real, then $P(f) = 2|H(f)|^2$.

Most often when we take a Fourier transform, we are interested in the power at different frequencies. We might have noise sources in the lab, and those noise sources might be most active over a particular range of frequencies (60 Hz or 120 Hz, for example). Taking the Fourier transform of a data stream might show a spike of power at those frequencies. Fourier analysis is also a standard way of doing cosmology, though since the projected sky is a sphere, we use spherical harmonics rather than sinusoids as the basis functions.

Real data are generally not continuously sampled, so we can't just use the integral definition of the Fourier transform (Equation (4)). The data are often sampled at finite points for a finite duration of time. The sampling of the function defines something called the *window function* (or the sampling function). What we actually observe is the product of the actual function we want to know about and the window function (which we know, but do not care about characterizing). The convolution theorem then says that

- the Fourier transform of your observed data is the convolution of the Fourier transform of your function of interest and the Fourier transform of the window function.

This is unfortunate. Suppose that you have data sampled for a finite amount of time: the window function is 0, then 1, then 0 again. This is a rectangular window. The Fourier transform of a rectangle is the sinc function $(\sin x)/x$, so your measured Fourier transform is the true Fourier transform convolved with $(\sin x)/x$. A perfect sine wave will not have a delta function as its Fourier transform, but a sinc function. The leakage of power from the true frequency to other frequencies is called *aliasing*, and it always happens in real data.

Very often, we have regularly sampled data of a function. In this case, the window function is basically a sequence of evenly spaced Dirac delta functions, also called a *Dirac comb*. The Fourier transform of a Dirac comb is another Dirac comb. The Fourier transform of the function is the convolution of the true Fourier transform with a Dirac comb: the Fourier transform is repeated periodically in frequency space. This is another manifestation of aliasing. Note that if you have regularly sampled data over a finite time, then your window function is actually the product of a Dirac comb and a rectangular window (whose Fourier transform is a sinc function)!

For regularly sampled data, the Dirac comb of sampling turns the integral definition of the continuous Fourier transform into a discrete analog with summation rather than integration: the *discrete Fourier transform*. I will assume that h is sampled at the N points $t_0 = 0, \Delta t \ldots (N-1)\Delta t$ spaced by (uniform) Δt. In that case, my window function also includes a rectangular window. The corresponding frequencies of the discrete Fourier transform are

$$f_n = \frac{n}{N\Delta t} \text{ with } n = -\frac{N}{2}, \ldots, \frac{N}{2}. \tag{13}$$

The edges of this range correspond to a special frequency–the Nyquist frequency. A remarkable fact is that is that if a function has zero power at frequencies higher than f_0, it may be recovered *exactly* by sampling with at an interval $\Delta t \leq 1/(2f_0)$. Well, that is, as long as your sampling extends over all time. If you only sampling your signal for a finite amount of time, then the sampling theorem does not apply. In practice it comes close for very large N, and it still defines the highest frequency that you can properly access. If your function does indeed have power at higher frequencies, then Parseval's theorem indicates that this extra power will still be there when I take the Fourier transform, but it cannot be attributed to the correct frequencies. It will be aliased to some other, lower frequency within the limits $-2/\Delta t, 2/\Delta t$. If the Fourier transform does not approach zero at the edges of the sampling window, then aliasing is likely to be affecting the results.

The discrete Fourier transform is then

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt \approx \sum_{k=0}^{N-1} h(t_k) \Delta t e^{2\pi i f_n t_k} = \Delta t \sum_{k=0}^{N-1} h(t_k) e^{2\pi i k n/N}. \tag{14}$$

The discrete inverse Fourier transform replaces Δt with $\frac{1}{N}$, and Parseval's theorem reads

$$\sum_{k=0}^{N-1} |h(t_k)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |H(f_k)|^2. \tag{15}$$

There are efficient ways to compute the discrete Fourier transform using the so-called *fast-Fourier transform*, or FFT. This is all for uniformly sampled data and neglected the treatment of errors; we will need an alternative for data that isn't uniformly sampled.

Before discussion data that are not uniformly sampled, I want to make a bit of an aside on noise. In the first part of this course we have discussed independent (or uncorrelated) and correlated noise. Sometimes you want to generate noise to understand your experiment or your data analysis method (this is often good practice). You did this in the first homework set. Generating uncorrelated Gaussian noise is easy–you can just use

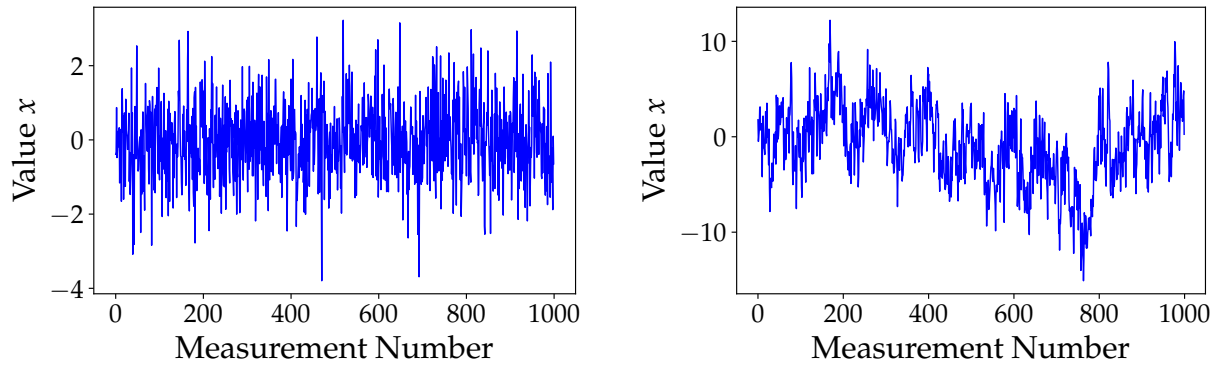
```
noise = np.random.randn(n)
```

if you want `n` samples. Generating correlated Gaussian noise is a bit harder. If you have the full covariance matrix for your data, you can perform an SVD and use the unitary matrix `U` to transform your uncorrelated Gaussian noise into correlated noise. You've done this before in the other direction, going from correlated to uncorrelated noise using `U.T`:

```
U, W, VT = np.linalg.svd(Cov_data)
err_uncorrelated = np.dot(U.T, err_correlated)
err_correlated = np.dot(U, err_uncorrelated)
```

When we have a time series, we usually don't think about variation in this way. It is much more common to talk about the power spectrum of the signal and the power spectrum of the noise. There are a few reasons for this. One is that it can be tricky to construct the covariance matrix for correlated errors: what is the variance for each point? With the power spectrum, we are usually referring to the properties of the *difference* between measurements taken at different times. Another reason is that inverting a large matrix is expensive (this is also a common problem for Gaussian process regression). It is much easier to work in Fourier space via the power spectrum.

Uncorrelated noise is known as *white*: it has the same power regardless of what frequency I measure it at. Two measurements taken equally spaced in time are just as (un)correlated as those taken far apart from one another. White noise is common and is the easiest to treat. In many situations, however, correlated noise is common. One example is the variability of astronomical sources (if you want to call this noise). This can be periodic, quasi-periodic, or just randomly fluctuating, but the fluctuations will have a characteristic timescale. Another example is noise on some kinds of readout amplifiers. A common type of noise in these situations may be called *flicker noise*, *pink noise*, or $1/f$ noise (these are all terms for the same thing). The power spectrum of this noise is not flat, but scales as $1/f$. This has a couple of properties. It is scale invariant, in the sense that a long time series of flicker noise looks the same as a short sequence. It is also correlated: the closer two measurements are in time, the more similar their values are. The figures below show an example of white noise and of flicker noise:



I'll present a brief aside on how to generate noise series like the ones above. It is easy enough to generate white Gaussian noise. You can then compute the Fourier transform and scale the magnitudes of the Fourier components by the square root of the desired power spectrum at that frequency.

```
whitenoise = np.random.randn(n)
whitefft = np.fft.fft(whitenoise)
fftfreq = np.fft.fftfreq(n)
coloredfft = whitefft*coloring(fftfreq)
colorednoise = np.fft.ifft(coloredfft)
```

You can use this approach to generate mock Gaussian time series with any power spectrum you want.

None of this helps much if you have *unevenly* sampled data, and it cannot account for measurement errors

that differ from one measurement to another. Suppose that you measure the flux from a star many times, but at unpredictable times and under varying conditions. In this case, we need an alternative to the discrete Fourier transform. That will be the subject of next class.