Capítulo 10

Métodos de Fourier y aplicaciones

Una gran cantidad de problemas computacionales en ciencias y geociencias son consideradas dentro del tópico general conocido como *métodos de transformadas de Fourier* o *métodos espectrales*. En algunos casos, la transformada de Fourier es simplemente una herramienta computacional eficiente para procesar datos. En muchos casos por ejemplo, los métodos de Fourier pueden ser útiles como herramientas para la integración numérica o para resolver ecuaciones diferenciales. Varios códigos usan los métodos de Fourier para resolver ODEs o PDEs de forma eficiente, para realizar convoluciones o deconvoluciones mucho más rápido que con algoritmos en el dominio del tiempo.

Algunas aplicaciones incluyen:

- Procesamiento de señales: filtrado, muestreo, reconstrucción
- Estadística: promedio./varianza, teorema del límite central, ruido, correlación y deriva (drift)
- óptica: difracción, interferometría
- geociencias: climatología, ciencias planetarias, geodinámica

10.1. Qué es la transformada de Fourier?

Para una función (con características adecuadas) f de una variable real, la definición de la transformada de Fourier es:

$$\hat{f}(\nu) = F[f] = \int\limits_{-\infty}^{\infty} f(t)e^{-2\pi i\nu t}dt \tag{10.1}$$

Esta operación toma una función real y genera a partir de ella otra función compleja, es decir, una función de una variable (en este case ν) que tiene una parte real y una parte imaginaria. Esta operación se puede pensar como una *aplicación lineal* o mapeo lineal (linear mapping), donde una función entra y otra (compleja) sale.

De cierta forma es más sencillo empezar con la idea de crear una función compleja f en vez de una función real. Así, ¿qué significa la transformada? En términos físicos simples, la transformada de Fourier (usaremos la abreviatura FT) produce un espetro de amplitudes complejo, que muestra que tan grande es la amplitud de las funciones seno y coseno, para cada frecuencia ν, al descomponer la señal en partes periódicas, cada parte con una frecuencia definida. Cuando la coordenada de la variable independiente es espacio (no tiempo), usamos el término *número de onda* con símbolo k.

Imagine construir una funci?øn f a partir de una suma (infinita) de senos y cosenos. Esto puede parecer imposible para un intervalo infinito, y de hecho, no todas las funciones se pueden construir así. Sin embargo, muchas si. Debido a que tenemos que incluir en nuestra suma de senos y cosenos todas las posibles frecuencias, la suma es realmente una integral, que se escribe:

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\nu) e^{+2\pi i \nu t} d\nu \tag{10.2}$$

Para una frecuencias particular ν, la contribución a la suma es sólo

$$e^{2\pi i \nu t} \hat{f}(\nu) = [\cos(2\pi\nu t) + i \sin(2\pi\nu t)]\, \hat{f}(\nu) \tag{10.3}$$

Como se puede observar, la función es una onda sinusoidal, que tiene una parte real y una imaginaria. En función del tiempo t, la función en (10.3) se repite a si misma, con un periodo $/1\nu$; para funciones en espacio, la variable correspondiente sería la longitud de onda $\lambda = 1/k$. Si la magnitud de la funci?øn de amplitud

If the magnitude of the amplitude function $|\hat{f}(\nu)|$ is large at some frequency ν_0 compared to that at any frequency, we would expect $f(t)$ built by (10.2) to approximate a complex cosine wave with frequency ν_0.

As an illustration, consider the function $\hat{g}(\nu)$ which is a Gaussian hump with its peak at ν_0 shown in Figure 10.1:

$$\hat{g}(\nu) = e^{-(\nu-\nu_0)^2/2\sigma^2} \tag{10.4}$$

where σ is a standard measure of the width. If we plug this into (10.2) we get (by methods we shall not describe in this class)

$$\hat{g}(\nu) = \sqrt{2\pi}\sigma e^{-2\pi^2\sigma^2 t^2} e^{2\pi i \nu_0} \tag{10.5}$$

You see g is a product of two exponential factors, a Gaussian hump of width $1/2\pi\sigma$, and a complex cosine term with frequency ν_0. As σ becomes small, and the amplitude of $\hat{g}(\nu)$ is more and more concentrated at ν_0, g looks more and more like a pure complex cosine, since the width of the Gaussian factor grows larger.

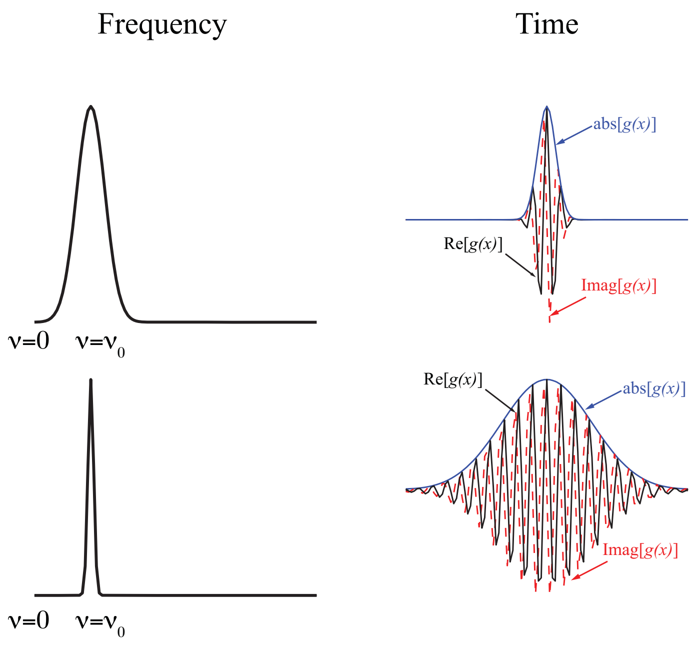

Figura 10.1: Fourier relations for Gaussian functions

If you accept the plausibility of the idea that you can build a non-periodic function from a sum of periodic parts, the next question is, Given a function f , how do we find the corresponding amplitude function $\hat{f}$? Of course, the answer has already been provided by (10.1). Remarkably, the amplitude calculating formula, (10.1) is almost the same as the synthesizing expression, (10.2). The only difference is the sign change in the exponential. Equation (10.2) is called the inverse Fourier transform, and can be written

$$f(t) = F^{-1}[\hat{f}] \qquad (10.6)$$

Obviously, if you take the FT of a function and then take the inverse FT of the result, you should get back the function you started with. This is exactly true only for moderately well behaved functions, as you can see in discussions elsewhere.

A brief discussion of notation is in order. There are several slightly different conventions for the FT. I use the one with the factor 2π in the exponent, which has always been the engineering practice and has become the standard in the applied math literature. A physicists form with a $(2\pi)^{12}$ outside. There are good reasons for preferring my notation, and I strongly recommend using it. First, there no other factors of 2π to remember in any of the other related results, the Convolution Theorem, ParsevalÕs Theorem, etc., while all other notations have random factors; second, the parameter ν is a frequency, not an angular frequency in radians per second, and $1/\nu$ is a period or a wavelength, and so much easier to identify with a physical scale, not six times the scale. I use the notation $\hat{f}$ for the FT, which is common in the mathematical work, while other people have $\tilde{f}$, which is completely unheard of! Some authors will use F for the FT of f .

Two physical realizations of the FT that come to mind. The acoustical idea of decomposing a sound into its component frequencies, something suggested by musical tradition for sounds that are nearly periodic. If the pressure time series is $p(t)$, and its FT is $\hat{p}(\nu)$, then the magnitude squared $|\hat{p}(\nu)|^2 d\nu$ is the acoustical power (or energy, these terms seem to be interchangeable in this context) in the frequency band ν to $\nu + d\nu$; conventionally, we might use f or v for frequency, not ν; for us f is unsuitable because I like to reserve this letter for the name of a function. A second familiar illustration comes from optics: the spectrum of a light source. Here we usually think of the intensity of the light as a function of wavelength, but a light signal in space $f(x)$ can be Fourier transformed and the magnitude squared $|\hat{f}(\nu)|^2$ is the spectrum as a function of wavenumber $k = 1/\lambda$. However, a proper model of light requires the concept of the FT of a random process, called the power spectrum, something we will be briefly talking about below.

You will probably be troubled by the fact even when we start with a real function in (1) we get a complex one out (though not always, as we shall see). What does this mean? Looking again at the process of building up a function from its Fourier spectrum, equation (2), and thinking about all the functions that can be built by summing the real part, you will see that cosine functions of the form $\cos 2\pi\nu t$ with different real amplitudes can make only even functions of t . An even function must satisfy $F(t) = F(-t)$, but obviously not all functions are even. Similarly, sums of sines are always odd functions, with $F(t) = -F(-t)$. While in general a function is neither even nor odd, every real function can be written uniquely as a sum of an odd and an even part. So it turns out that the real part of the FT copes with the even part of the function and the imaginary part with the odd part; in general both parts are necessary. While on this topic, we note the following easily demonstrated symmetry: when $f(t)$ is real, then the real part of $\hat{f}(\nu)$ is always an even function, and the imaginary part is always odd.

I will not discuss some of the conditions needed for a function f so that (2) is valid. But I want to note that there are some functions that don't quite follow the basic rules for (2) to hold. They are distributions or generalized functions, and are treated rigorously by using the idea of an equivalence class, this time a class of sequences of functions, but that goes beyond this course. One of this fucntions is for examplethe Delta function.

10.2. Fourier transform for discrete data

We now turn from Fourier theory to the subject of digital signal processing; though as we will see, much of what would be covered in the literature will be used here. However, our treatment will have a distinctive flavor, deriving from the nature of digital signals. The theory of the Fourier transform assumes that what is being transformed (and the transforms themselves) are functions (even if generalized) on the real line or a higher-dimensional equivalent. We may say that, if we view these functions as functions of time, that they are defined `in continuous time`. However, digital signals are, intrinsically, not so defined: they are collections of numbers, representing (usually) a continuous time signal sampled at regular intervals. Such functions are called sampled data and may be said to be defined in discrete time.

We begin by describing how Fourier theory works applied to such data, for a special case:

- Discrete-time data defined over a finite range. This is of course what we usually have to deal with. The Fourier transform of this turns out to be a finite amount of discrete frequency data, giving the same similarity between a function and its transform that we had in the original definition. This Fourier transform of a finite amount of discrete time data is called (what else?) the Discrete Fourier Transform or DFT.

We will not spend much of our time exploring the properties of the DFT, rather just outlining ,the computation of the DFT, which can be done very efficiently using the Fast Fourier Transform, or FFT.

We suppose that our sequence $\{x_n\}$ is finite in length, with terms $x_0, x_1, \dots x_{N-1}$ (N in all), rather than infinite. How do we now define the Fourier transform? We start in what is a somewhat unusual way, at least in terms of signal processing, which is to consider the fitting of sine waves to the data. We can always represent the data as some sum of sine waves plus a residual:

$$x_n = \sum_{l=0}^{L-1} C_l e^{2\pi i n \nu_l} + \epsilon_n, \quad n = 0, 1, \cdots N-1 \tag{10.7}$$

where the frequencies $\nu_0, \nu_1, \cdots$ are given. The least-squares criterion would tell us to choose the coefficients C_l so as to to minimize the sum of squares

$$\sum_{l=0}^{L-1} |\epsilon_n|^2.$$

with solution

$$C_l = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-2\pi i n l/N}, \tag{10.8}$$

where the l sampling is done this way for a reason (which one?).

A more apropriate definition of the Discrete Fourier Transform or DFT is

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i n k/N}, \tag{10.9}$$

which, if we compare it with the definition of the Fourier transform of an infinite sequence, shows that

$$\hat{x}_k = X(f) \qquad \text{for } f = k/N \tag{10.10}$$

This is thus a sampled version of the continuous-frequency transform. The inverse Fourier transform is then

$$\hat{x}_n = \sum_{n=0}^{N-1} x_k e^{+2\pi ink/N}, \tag{10.11}$$

which can be shown to be correct below.

$$\begin{aligned}
\hat{x}_n &= \sum_{n=0}^{N-1} x_k e^{+2\pi ink/N} \\
&= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_m e^{-2\pi imk/N} e^{+2\pi ink/N} \\
&= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_m e^{+2\pi ik(n-m)/N} \\
&= \sum_{m=0}^{N-1} x_m N\delta_{mn} = x_n
\end{aligned}$$

where the next-to-last step makes use of the orthogonality relationship. We thus have a transform pair - the DFT and the inverse DFT - between finite length sequences of numbers.

10.3. The Fast Fourier Transform

So, how many computer calculations are needed to obtain the DFT of a series with N points? In general, matrix multiplications require N^2 complex multiplications plus a small number of other operations. So, the FT appears to be an $O(N^2)$ process. But, with an algorithm known as the *fast Fourier transform* it can be computed with $O(N \log N)$ operations. This difference is BIGGGG.

We will not reinvent the wheel and we will not really learn how the FFT algorithm works. The basic idea you need to remember is that for a special case, when N is an integer power of 2, the FFT is the fastest. The number of calculations is reduced because Danielson and Lanczos showed that the DFT of N length can be rewritten as the sum of two DFTs, each of length $N/2$. One of the two is formed from the even-numbererd points of the original N, the other from the odd-numbered points. I will not show the proof.

We will use for the DFT within the *Numpy* or `Scipy` packages, both of which behave in a similar way. The best algorithm I know is the FFTW, the *fastest Fourier transform in the West*, which can be implemented in Python with the package pyFFTW available online, but I have not ried doing this. The FFTW is the best and fastest code, but if you are looking for speed, you probably won't be doing the computations in Python anyways, but rather in C or Fortran with a Python wrapper. The FFTW is the same FFT that Matlab and other programs use. The Numpy implementation is quite simple

```
fx = np.fft.fft(x,nfft)
```

with fx as the output, most likely a complex array. The fft requires you to input the number of frequency points to compute, but usually you will use the number of points of the input data array x. You can, if you prefer, use your own FFT.

10.4. Some applications

Fourier methods have revolutionized fields in science and engineering, from radio astronomy to medical imaging, seismology, etc. According the Numerical Recipes (NR) the wide application of Fourier methods may be credited mainly to the existence of the FFT, given the speeding up of non-trivial algorithms. Direct applications of the FFT are convolution and deconvolution, correlation and autocorrelation, optimal filtering, power spectrum estimation and computation of Fourier integrals as well as solving PDEs.

10.4.1. Convolution and Deconvolution

We can define the convolution of two functions for the continuous case as

$$r(t) = s * u = \int_{-\infty}^{\infty} s(\tau)u(t-\tau)d\tau \tag{10.12}$$

where $s * u$ denotes convolution. Now, the *convolution theorem* states that

$$s * u \rightleftarrows \hat{s}(\nu)\hat{u}(\nu) \tag{10.13}$$

In other words, the Fourier transform of the convolution is just the product of the individual Fourier transforms. The convolution theorem is fairly easy to prove and I will leave this to you.

It is important to know, that you can compute the convolution in the time domain, but as the number of points gets large, it turns out to be much faster to take the FFTs of each signal and then inverse FFT.

So, once you know how to use the FFT, convolution is quite simple. One interesting question then arises. If convolution is just multiplication in the frequency domain, we might expect to perform deconvolution by a division in the frequency domain, right? The short answer is yes. At first glance the answer seems easy using the Convolution Theorem: since $\hat{r} = \hat{s}\hat{u}$, and $\hat{u}$ is known, we just divide by $\hat{u}$ and then take the inverse FT: $s = F^{-1}[\hat{r}/\hat{u}]$. Unfortunately in most practical systems, when $\hat{u}$ gets very small the recovered signal is obtained by amplifying small values. Additionally, these small values are often submerged by noise in the system, and so this deconvolution succeeds only in amplifying noise, not true signal. Deconvolution is therefore an unstable procedure and straight division of the FTs hardly ever works; the process must be **regularized**, a topic we will meet in later sections.

One note of caution. Convolution for discrete signals may be contaminated by wrap-around effects, so it is recommended that you pad your signals with zeroes at the ends to avoid this.

10.4.2. Correlation and autocorrelation

Similar to convolution, correlations and autocorrelations can be computed using the FFT. We will not work discuss this, but you can easily derive the way to compute correlations between signals (or the signal with itself) using the FFT.

10.4.3. Power Spectrum estimation

In sciences there is a continuing interest in the spectral analysis of various sorts of recorded or gathered data series. In some cases the researcher may be interested in isolating single frequencies embedded in some noise (e.g., normal mode seismology, solar modes, climate data, etc.) or in a continuous spectrum whose shape may be related to simple functional forms with a few parameters used to describe it (source physics, bathymetry). In general, the Power spectral density function (PSD) describes the power of each frequency contained in the time series signal.

The periodogram , which is one of the early estimation procedures of the PSD is defined as

$$P(f) = \left| \sum_{n=0}^{N-1} x_n e^{-2\pi i f n} \right|^2 \tag{10.14}$$

where x is a sequence of data points (unit sampling rate assumed here), N is the total number of points of the sequences and f is the frequency. Note here that the frequency variable f is continuous and so we are in fact trying to find a function $P(f)$ from the finite series $x(t)$.

You might think that the periodogram is all you need to know, but as you can read in various papers and the literature, the PSD is an *estimate*, not a direct measurement. As stated above, we are really trying to find a continuous function out of a discrete data set x_n.

It turns out that the periodogram is not a particularly good choice for an estimate of the PSD. It has been known for a long time, that windowing is necesary, so a better estimate is the *windowed periodogram*, where

$$P_W(f) = \left| \sum_{n=0}^{N-1} a_n x_n e^{-2\pi i f n} \right|^2 \tag{10.15}$$

we multiply the data sequence x_n with a window or taper a_n before actually performing the FFT calculation. The window a_n can be one out of many choices including the Hanning, Hamming, Bartlett's, prolate, triangular windows. The periodogram is basically like using a boxcar window.

An even more advanced choice is to use the *multitaper* algorithm, which as the name suggests, use a number of tapers instead of only one. We will not talk about this particular choice, but it may be a good idea for you to know that it exists.

10.4.4. Filtering

Suppose that you have a signal you want to filter digitally. It could be that the signals is contaminated by noise of some sort. For example, signals in many cases are corrupted by 60 Hz power-line interference and the underlying, uncorrupted signals lies somewhere inmersed below the noise.

Although we can perform filtering in the frequency domain, by *convolving* the signal $x(t)$ with some other function $u(t)$ so that the noise part of the signal is reduced as much as possible. In the frequency domain we could simply have

$$C(\nu) = X(\nu)U(\nu) \tag{10.16}$$

where the spectrum of the filter $U(\nu)$ has small amplitudes in the frequency range where the noise is dominating (say around 60 Hz for example). We can then *design* a filter $U(\nu)$ as a high-pass, low-pass, band-pass or band-stop filter, also known as *notch filter* (see figure).

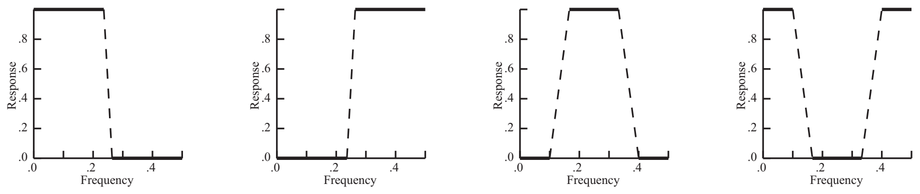

Figura 10.2: Ideal frequency-selective filters: the four commonest types. The parts of the frequency band for which the response is 1 are called the passband(s); where the response is 0 is the stopband(s). The dashed lines show the region in which (for most design methods) the response goes from 1 to 0 (not necessarily linearly as shown here), which is called the transition band.

Now, although we are in Fourier theory, I wish to show to you a filter that is not designed in the frequency domain but rather in the time domain. It is really up to you to choose the way you design your filter, but for simplicity we will use available filters that somebody else has already developed.

Linear filters

The most general linear filter takes a sequences x_n of input points and produces a sequence y_n of output points using the formula

$$y_n = \sum_{k=0}^{M} c_k x_{n-k} + \sum_{j=1}^{N} d_j y_{n-j} \tag{10.17}$$

Here the $M+1$ coefficients c_k and the N coefficients d_j are fixed and define the filter response. The filter above produces each output value from the current and M previous input values, and from its own N previous output values. if $N = 0$, so that there is no second sum, then the filter is called a *nonrecursive* or *finite impulse response* (FIR). If $N \neq 0$, then it is called an *infinite impulse response* filter or (IIR). The IIR means that one could make the impulse response be very long, although in general the response decays as N gets big, becoming negligible.

The relation between the c_k's and d_j's and the filter response $U(\nu)$ is

$$U(\nu) = \frac{\displaystyle\sum_{k=0}^{M} c_k e^{-2\pi i k(\nu\Delta)}}{1 - \displaystyle\sum_{j=1}^{N} d_j e^{-2\pi i j(\nu\Delta)}} \tag{10.18}$$

where Δ is the sampling interval.

Python function

Again, it is not our intention to learn how to design the filters, just the fact that we could if we wanted to. For this class, we will use a simple designed bandpass filter, similar to what is used in matlab. We use a Butterworth filter, with two frequency limits, `fmin` and `fmax` which represent the limits of the frequency you want to keep within the filtered signal. You have to take two steps, first create the numerator and denominator of the IIR filter polinomials `a` and `b`, and second, filter the actual signal `x`. The way we perform a bandpass filter:

```
# Define nyquist and scale filter
fnyq = 0.5/dt
f0   = fmin/fnyq
f1   = fmax/fnyq

# Get polinomials
wn   = [f0,f1]
b, a = signal.butter(4, wn,'bandpass')

# two-pass filter
y    = signal.filtfilt(b, a, x)
```

where `fmin` and `fmax` represent the low and high frequencies that limit the frequencies that are expected to be passed.

For your information, this filter is a Butterworth recursive filter. There are many other filters but this one is used extensively in the scientific community because it is simple. Nevertheless other filters like the Kalman filter or Chebyshev filters may be more suited for your particular problem.

Problemas

10.1. test test