PHY408

Lecture 6: Discrete Fourier Transforms

February 15, 2023

Fourier Series for periodic functions

$$\begin{aligned} G(f) &= \int_{-\infty}^{\infty} g(t) e^{-i2\pi ft}\, dt \\ g(t) &= \int_{-\infty}^{\infty} G(f) e^{i2\pi ft}\, df \end{aligned} \tag{1}$$

- If $g(t)$ is periodic with period of T, then it can be exactly represented over that range by all sinusoids with frequencies that are integer multiples of the fundamental frequency $1/T$ (i.e., a discrete spectrum).

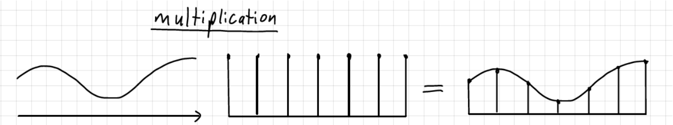
$$f(t) = \frac{1}{T} \sum_{k=-\infty}^{\infty} F_k e^{i\omega_k t}, \quad \text{where } \omega_k = 2\pi k/T \tag{2}$$

where

$$F_k = \int_0^T f(t) e^{-i\omega_k t} dt \tag{3}$$

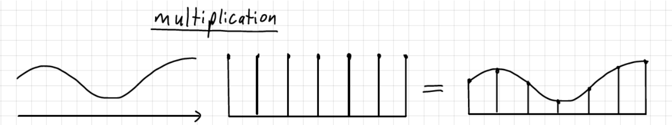
Discrete Fourier transform (DFT) - Sampling

Discrete sampling is equivalent to multiplying a continuous signal by the Dirac comb.

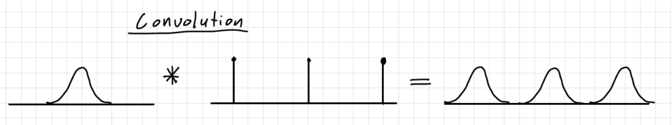


Discrete Fourier transform (DFT) - Sampling

Discrete sampling is equivalent to multiplying a continuous signal by the Dirac comb.



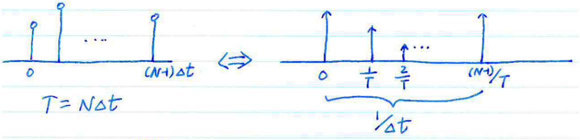
Convolution with the Dirac comb produces a period signal.



Discrete Fourier transform (DFT)

For a discrete signal $\{g_0, g_1, \cdots g_{N-1}\}$ obtained by sampling the continuous signal $g(t)$ at interval Δ:

1. Total length of the signal $N * \Delta = T$: we only need discrete harmonics whose freqs are multiples of $1/T$ to represent $\{g_i\}$.
2. Sampling interval Δ: multiplication in the time domain by Dirac comb with Δ spacing $\rightarrow$ convolution in the freq domain by $\sum_{k=-\infty}^{\infty} \delta(f - \frac{k}{\Delta})$ (period of $1/\Delta$)



How to compute G_k's?

According to Fourier Series:

$$G_k = \int_0^T g(t)e^{-i\omega_k t}dt \sim \sum_j g_j e^{-i2\pi\frac{k}{T}(j\Delta t)}\Delta t = \sum_j g_j e^{-i2\pi kj/N}\Delta t \quad (4)$$

based on $T = N\Delta t$. Similarly, reconstruction of the time series

$$g(t_j) = g_j = \frac{1}{T}\sum_{k=0}^{N-1} G_k e^{i2\pi\frac{k}{T}(j\Delta t)} = \frac{1}{\Delta t}\frac{1}{N}\sum_{k=0}^{N-1} G_k e^{i2\pi kj/N} \quad (5)$$

DFT:

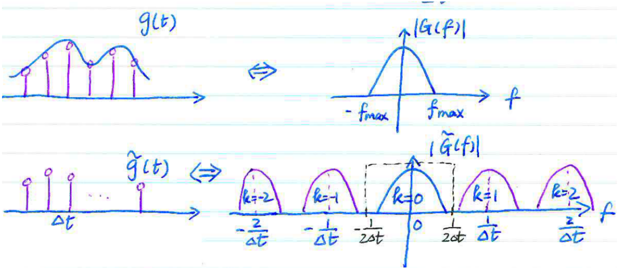
$$G_k = \Delta t \sum_j g_j e^{-i2\pi kj/N}, \qquad g_j = \frac{1}{N\Delta t}\sum_{k=0}^{N-1} G_k e^{i2\pi kj/N} \quad (6)$$

- how good of a representation G_k is to $G(\omega)$?

Sampling theorem

If $G(f)$ is the true spectrum of the continuous time series $g(t)$, then the spectrum of the sampled time series is

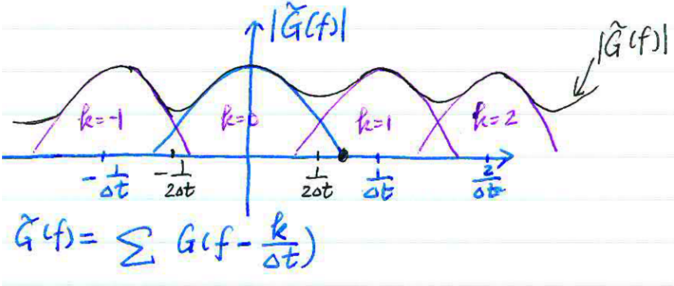
$$\tilde{G}(f) \sim G(f) * \sum_k \delta(f - \frac{k}{\Delta t}) = \sum_k G(f - \frac{k}{\Delta t}) \qquad \text{Periodic!} \qquad (7)$$



Sampling theorem: if $f_{max} < \frac{1}{2\Delta t} = \frac{1}{2} f_s$ (known as Nyquist frequency, a minimum two points to sample one cycle), then $G(f)$ can be fully recovered from $\tilde{G}(f)$, in other words, the discrete time series g_k can fully reconstruct the original continuous signal $g(t)$.

Aliasing

If the sampling criterion is not satisfied, i.e. $f_{max} \geq \frac{1}{2} f_s$



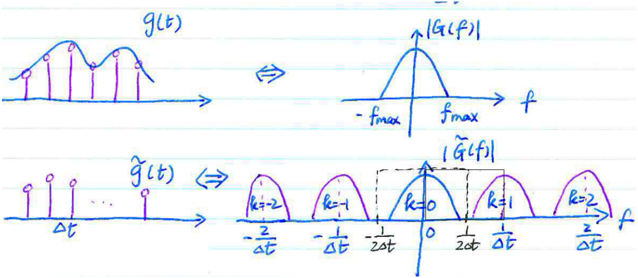
1. high and low frequency content overlap: not possible to extract $G(f)$ accurately from $\tilde{G}(f)$.
2. anti-aliasing filter: given a sampling rate f_s, first low-pass filter the continuous input signal $g(t)$ below $f_{max} = \frac{1}{2} f_s$ before digitization.

Reconstruction of the original continuous signal

1. $\tilde{G}(f)$ periodic:

$$\tilde{G}(f) \sim G(f) * \sum_k \delta(f - \frac{k}{\Delta t}) = \sum_k G(f - \frac{k}{\Delta t}) \qquad (8)$$

therefore we only need to examine $\tilde{G}(f)$ in the freq range of $[0, \frac{1}{\Delta t}]$ (implemented by most numerical algorithms) or more physically $[-\frac{1}{2}\frac{1}{\Delta t}, \frac{1}{2}\frac{1}{\Delta t}]$ (*fftshift*).



The Fourier transform of which should correspond to a continuous signal in the time domain.

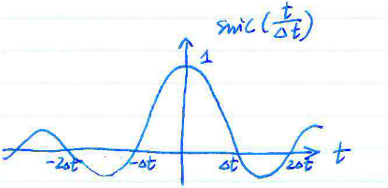
Fourier interpolation

What is the effect of cutting one cycle of $\tilde{G}(f)$ over $[-\frac{1}{2}\frac{1}{\Delta t}, \frac{1}{2}\frac{1}{\Delta t}]$?

1. equivalent to multiplying a boxcar in the frequency domain, i.e. $\tilde{\tilde{G}}(f) = \tilde{G}(f)H(f)$, where $H(f) = 1$, $-\frac{1}{2}\frac{1}{\Delta t} < f < \frac{1}{2}\frac{1}{\Delta t}$
2. according to the convolution theorem, in the time domain this is effectively

$$\tilde{\tilde{g}}(t) \sim g_k * sinc\left(\frac{t}{\Delta t}\right) = \sum_m g_m\, sinc\left(\frac{t - m\Delta t}{\Delta t}\right) \tag{9}$$

This is called Fourier interpolation scheme. $g(t)$ is a weighted sum of g_k's, and the closer t_k is to t, the larger the weight.



Fourier Interpolation

1. $\tilde{\tilde{g}}(t_k) = g_k$, i.e. Fourier interpolation scheme is exact on the sample points. But how about the points in between?

Fourier Interpolation

1. $\tilde{\tilde{g}}(t_k) = g_k$, i.e. Fourier interpolation scheme is exact on the sample points. But how about the points in between?
2. Fourier interpolation is exact: i.e. $\tilde{\tilde{g}}(t) = g(t)$ if the sampling criterion is met: i.e. $f_{max} < \frac{1}{2} f_s$; otherwise, values are still exact on the sample points, but not necessarily in between (which is intuitively true!).

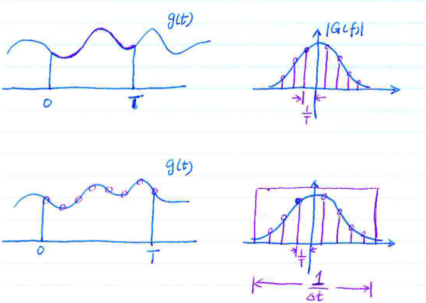
Effect of finite length

If a continuous signal is cut between $[0, T]$,

1. can be represented by a set of Fourier series in the freq domain: freq resolution $\delta f = \frac{1}{T}$.
2. longer time series $\Rightarrow$ higher resolution in freq domain
3. $\delta f\, T \sim 1$, similar to Heisenberg uncertainty principle: a signal can not be infinitely sharp in both time and freq domain; infinite resolution in f can not be achieved with a finite length of time series.
4. Question: what about padding zeros before or after a finite time series? What happens if the start and end of the series have different values? (assumed periodicity)
5. Instead of boxcar, a better windowing function (Cosine/Hann taper: $H(t) = \frac{1}{2}(1 - \cos t)$)

Therefore for 'a sampled finite time series' as $0:\Delta:T$ (e.g. "numpy.arange(0,T,Δ)"):

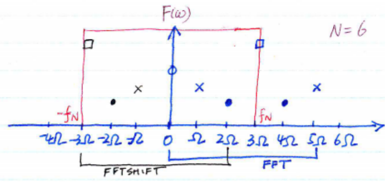
1. $T \Rightarrow$ freq spacing/resolution $1/T$
2. $\Delta \Rightarrow$ freq range $[0, \frac{1}{\Delta t}]$ or $[-\frac{1}{2}\frac{1}{\Delta t}, \frac{1}{2}\frac{1}{\Delta t}]$ (fold back the second half of $[0, \frac{1}{\Delta t}]$ as negative frequencies)

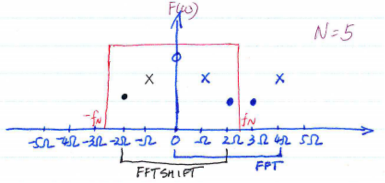


i.e., we can represent the frequency content of g_k by $\frac{1}{\Delta t}/\frac{1}{T} = N$ points.

Numerical implementation

1. $g_k, k = 1, \cdots N \to G_k, k = 1, \cdots N$. But what are the corresponding frequencies of G_k? Because of discretization, it differs for N being even or odd.
2. N is even, e.g. $N = 6$, with interval Ω we have: $0 : (N-1)\Omega$ corresponding to $[0 : (\frac{N}{2} - 1)\Omega, -\frac{N}{2}\Omega : -\Omega]$, or through fftshift: $-\frac{N}{2}\Omega : (\frac{N}{2} - 1)\Omega$ (one fewer point in the positive half).
3. N odd, e.g. $N = 5$, with interval Ω we have: $0 : (N-1)\Omega$ corresponding to $[0 : (\frac{N-1}{2} - 1)\Omega, -\frac{N-1}{2}\Omega : -\Omega]$, or through fftshift: $-(\frac{N-1}{2})\Omega : \frac{N-1}{2}\Omega$ (same number of points in positive and negative halves).





Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]
   f = numpy.linspace(0,10,5, endpoint=False)
   f will be [0.0, 2.0, 4.0, 6.0, 8.0]

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]
   f = numpy.linspace(0,10,5, endpoint=False)
   f will be [0.0, 2.0, 4.0, 6.0, 8.0]
2. f=numpy.arange(0,10,2)
   f will be [0, 2, 4, 6, 8]

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]
   f = numpy.linspace(0,10,5, endpoint=False)
   f will be [0.0, 2.0, 4.0, 6.0, 8.0]
2. f=numpy.arange(0,10,2)
   f will be [0, 2, 4, 6, 8]
3. dt = 0.1
   N=20
   f=numpy.fft.fftfreq(N, dt)

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]
   f = numpy.linspace(0,10,5, endpoint=False)
   f will be [0.0, 2.0, 4.0, 6.0, 8.0]
2. f=numpy.arange(0,10,2)
   f will be [0, 2, 4, 6, 8]
3. dt = 0.1
   N=20
   f=numpy.fft.fftfreq(N, dt)
   f will be [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, -5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5]

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]
   f = numpy.linspace(0,10,5, endpoint=False)
   f will be [0.0, 2.0, 4.0, 6.0, 8.0]
2. f=numpy.arange(0,10,2)
   f will be [0, 2, 4, 6, 8]
3. dt = 0.1
   N=20
   f=numpy.fft.fftfreq(N, dt)
   f will be [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, -5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5]
4. f=numpy.fft.fftshift(np.fft.fftfreq(N, dt))

Numerical implementation

Examples:

1. f = numpy.linspace(0,10,5)
   f will be [0.0, 2.5, 5.0, 7.5, 10.]
   f = numpy.linspace(0,10,5, endpoint=False)
   f will be [0.0, 2.0, 4.0, 6.0, 8.0]
2. f=numpy.arange(0,10,2)
   f will be [0, 2, 4, 6, 8]
3. dt = 0.1
   N=20
   f=numpy.fft.fftfreq(N, dt)
   f will be [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, -5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5]
4. f=numpy.fft.fftshift(np.fft.fftfreq(N, dt))
   f will be [-5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]