Notas de aula

Introdução à Física Computacional

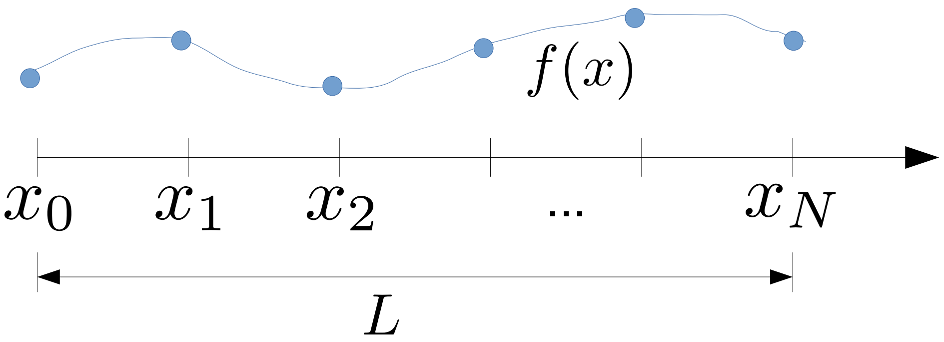
Prof. Gerson – UFU – 2019

Atendimento:

- Sala 1A225
- Email: gersonjferreira@ufu.br
- Webpage: http://gjferreira.wordpress.com
- Horário: sextas-feiras 16:00 – 16:50

Discrete Fourier Transform

Consider a function f(x) on a discrete lattice



The points are set as $x_j = x_0 + j\Delta x$

with periodic boundaries $f(x+L) = f(x)$

and size L $x_N = x_0 + L$

This give us $\rightarrow$ $L = N\Delta x$

Write f(x) as a Fourier series

$$f(x) = \sum_n c_n e^{ik_n x}$$

$$k_n = \frac{2\pi}{L} n$$

with n = {0 … N}, such that $k_N = \dfrac{2\pi}{\Delta x}$

is the largest meaningful frequency,
as its wave-length matches the lattice spacing.

Now: use the plane-wave orthogonality to find cn.

Discrete Fourier Transform

Using the ‘Fourier trick’, we find … and transforming the integral back to a sum

$$c_n = \frac{1}{L}\int_{x_0}^{x_0+L} f(x)e^{-ik_n x}dx = \frac{e^{-ik_n x_0}}{N}\sum_{j=0}^{N-1} f(x_j)e^{-ik_n j\Delta x}$$

Notice that $k_n \Delta x = 2\pi n/N$ … and defining $f_j \equiv f(x_j)$

we get the coefficients of
the discrete Fourier transform: $c_n = \frac{e^{-ik_n x_0}}{N}\underbrace{\Big[\sum_{j=0}^{N-1} f_j e^{-2\pi i \frac{jn}{N}}\Big]}$

Only this part is typically
implemented on the
Fourier libraries

Discrete Fourier Transform (DFT)

… but in physics, we prefer to define the Fourier transform as

$$F_n = \sqrt{N} c_n = \frac{e^{-ik_n x_0}}{\sqrt{N}} \left[\sum_{j=0}^{N-1} f_j e^{-2\pi i \frac{jn}{N}} \right]$$

… such that the inverse Fourier transform reads

$$f_j = \frac{1}{\sqrt{N}} \sum_n e^{ik_n x_0} F_n e^{2\pi i \frac{jn}{N}}$$

Fast Fourier Transform (FFT)

Numpy uses the FFT library [documentation here]

The implementation is "equivalent" (?) to the DFT from the previous page, but with $x_0 = 0$

To get the usual physics normalization with the $N^{1/2}$, use the option **norm='ortho'**

But the x_0 phases have to added by hand if you want to get the canonical Fourier transform

---

FFT and DFT are not really equivalent… FFT is much faster!

To understand why, please check the books and the class biblography: [Tao Pang] [Thijssen]

To take advantage of the FFT speed up, always use N as a power of 2, i.e. $N = 2^p$.

$N = 128, 256, 512, 1024, 2048, ...$

---

important detail: use the fftshift and ifftshift to change the FFT domain betweeen

$k = [0, 2\pi[$ and $k = [-\pi, \pi[$

Exercise...

Implement the DFT sums to define your own the DFT and the iDFT.

→ they will be slow… in practice you should use the FFT library.

Compare your results with my numpy/FFT example [available at the course webpage]