A simple algorithm for calculating numerical non-uniform discrete Fourier transforms

Marius Johannes Hofmann

Table of Contents:

I. Introduction

In science the problem of calculating Fourier transforms of experimental data numerically, is very common. The cosine and the sine transformation are two such variations:

$$g_C(\omega) = \int_0^\infty f(t)cos(\omega t)dt \qquad\qquad g_S(\omega) = \int_0^\infty f(t)sin(\omega t)dt \tag{1}$$

Here, $f(t)$ could be a time signal, for instance, and $g_{C,S}(w)$ the corresponding frequency-domain spectra. The physicists definition of angular frequency $\omega = 2\pi\nu$ is chosen. Conventional Discrete Fourier transform (DFT) algorithms often implemented in commercially available software products such as Fast Fourier Transform (FFT) often require evenly spaced data points. However, when data extends over several orders of magnitude, the equidistant spacing is unfavorable and logarithmic spacing is preferred, for instance. Another typical case of unevenly spaced data occurs when compiling data recorded with different instruments or measured under different conditions. An example is time/frequency – temperature superposition. In such cases, non-uniform DFT (nDFT) is required. One simple approach is described here. It is based on the pioneering work of Filon [1] and was further developed by Blochowicz et al. [2] and Rivera et al. [3].

II. Theory

Exploiting linearity the finite integration range is subdivided into parts given by the finite set of N data points $f(t_i)$.

$$g_C(\omega) = \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} f(t)cos(\omega t)dt$$

(2)

Next, integration by parts is performed:

$$\int_{t_i}^{t_i+1} f(t)cos(\omega t)dt =$$

$$\frac{1}{\omega}[f(t_{i+1})sin(\omega t_{i+1}) - f(t_i)sin(\omega t_i)] - \int_{t_i}^{t_{i+1}} \frac{df(t)}{dt}\frac{sin(\omega t)}{\omega}dt$$

(3)

Based on previous work by Filon [1] and as later suggested by Blochowicz et al. [2] the derivative can be replaced by the (constant) difference quotient of two consecutive data points:

$$\frac{df(t)}{dt} \approx \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i}$$

(4)

Integration of the remaining sine is straightforward and yields:

$$\int_{t_i}^{t_i+1} f(t)cos(\omega t)dt =$$

$$= \frac{1}{\omega}[f(t_{i+1})sin(\omega t_{i+1}) - f(t_i)sin(\omega t_i)]$$
$$+ \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i}\frac{cos(\omega t_{i+1}) - cos(\omega t_i)}{\omega^2}$$

(5)

The summation over all data points is carried out (see eq. 2) reads:

$$g_C(\omega) =$$
$$\sum_{i=0}^{N-1} \frac{1}{\omega}[f(t_{i+1})sin(\omega t_{i+1}) - f(t_i)sin(\omega t_i)]$$
$$+ \sum_{i=0}^{N-1} \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} \times \frac{cos(\omega t_{i+1}) - cos(\omega t_i)}{\omega^2}$$

(6)

In the first sum of eq. 6 almost all terms cancel out. Only the first $i = 0$ and the last one $i = N - 1$ survive:

$$g_C(\omega) = \frac{1}{\omega}[f(t_N)sin(\omega t_N) - f(t_0)sin(\omega t_0)] + \sum_{i=0}^{N-1} \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} \times \frac{cos(\omega t_{i+1}) - cos(\omega t_i)}{\omega^2} \quad (7)$$

The sine transformation is obtained equivalently and the exact deviation can be found in the Appendix:

$$g_S(\omega) = \frac{-1}{\omega}[f(t_N)cos(\omega t_N) - f(t_0)cos(\omega t_0)] + \sum_{i=0}^{N-1} \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} \times \frac{sin(\omega t_{i+1}) - sin(\omega t_i)}{\omega^2} \quad (8)$$

Expressions 7 and 8 approximate the nDCT and nDST, respectively. The ideal spacing in Fourier space is given by:

$$\omega_i = {t_i}^{-1} \quad (9)$$

The calculation of each ω_i requires summation over all the N data points ${t_i}^{-1}$, therefore, the complexity is $O(N^2)$.

III. Some Examples

Expressions 7 & 8 are the main results. They were implemented in a Python script. In what follows, a couple of examples are given. The raw data of the examples is also made available.

A) The first example concerns a real-valued time signal $f(t_i)$ which could be a voltage measured by a sensor, for instance. Figure 1a displays the function

$$f(t) = cos(20t) + cos(37t) + 2sin(78t) \quad (10)$$

which was superimposed by 10% noise and logarithmically sampled. Figure 1b shows the corresponding sine and cosine transforms, $g_S(\omega)$ and $g_C(\omega)$, respectively. The three frequency components are well recognized, yet, the transformation suffers from truncation errors leading to rapid oscillations. This issue is easily overcome by multiplying $f(t)$ with a Gaussian with a decay parameter Δ, which corresponds to a convolution in Fourier space.

$$f(t)exp(-t^2/\Delta) \tag{11}$$

Figure 1c shows the result for $\Delta = 10$. The function now decays to zero within the given time range. This removes the truncation errors, as demonstrated in Fig. 1d, on the cost of spectral resolution. The peaks at $20rad/s$ and $37rad/s$ are in-phase in $g_C(\omega)$, whereas the one at $78rad/s$ is out-of-phase, as expected. For the sine-transformation it is the opposite.

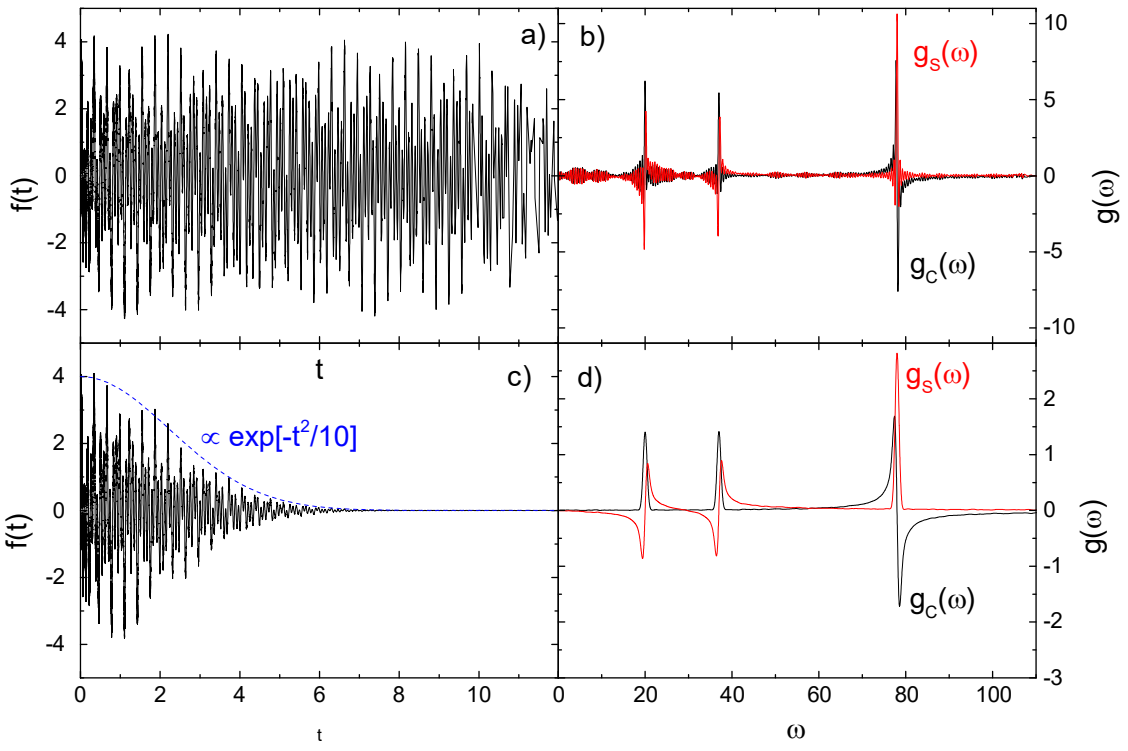

Figure 1. a) Nonuniformly spaced and real-valued time signal $f(t_i)$ according to eq. 10 and (b) the corresponding nDFT's $g_C(\omega)$ and $g_S(\omega)$, respectively. c) Time signal $f(t_i)$ after multiplication with a Gaussian (see eq. 11) and corresponding nDFT (d).

B) Next, a non-uniformly sampled biexponential decay extending over several orders of magnitude is considered, specifically, the function

$$f(t) = 0.9exp(-x/1) + 0.1exp(-x/10)$$

(12)

which is plotted in Fig. 2a. The sampling is again logarithmic, as demonstrated in the inset. Furthermore, t_i as well as $f(t_i)$ were superimposed by 2% noise to mimic experimental conditions. Such a function could reflect a time-autocorrelation function, for instance. The two decay times are given by $\tau_1 = 1$ and $\tau_2 = 10$ and the mean decay time by

$$\langle\tau\rangle = \sum_i a_i\tau_i = \int_0^\infty f(t) = 1.9$$

(13)

Figure 2b displays the corresponding Fourier transforms $g_S(\omega)$ and $g_C(\omega)$, respectively. In $g_C(\omega)$ two peaks are recognized which reflect ${\tau_1}^{-1}$ and ${\tau_2}^{-1}$. At high frequencies, power laws $g_S(\omega) \propto \omega^{-1}$ and $g_C(\omega) \propto \omega^{-2}$ are found. The analytical expressions of $g(\omega)$ are also plotted in Fig. 2b:

$$g_S(\omega) = 0.9\omega/(\tau^{-2} + \omega^2) + 0.1\omega/((10\tau)^{-2} + \omega^2)$$

$$g_C(\omega) = 0.9\,\tau/(1 + (\omega\tau)^2) + \tau/(1 + (10\omega\tau)^2)$$

(14)

The numerical nDFT's reproduce the theoretically expected curves very well, over several orders of magnitude.

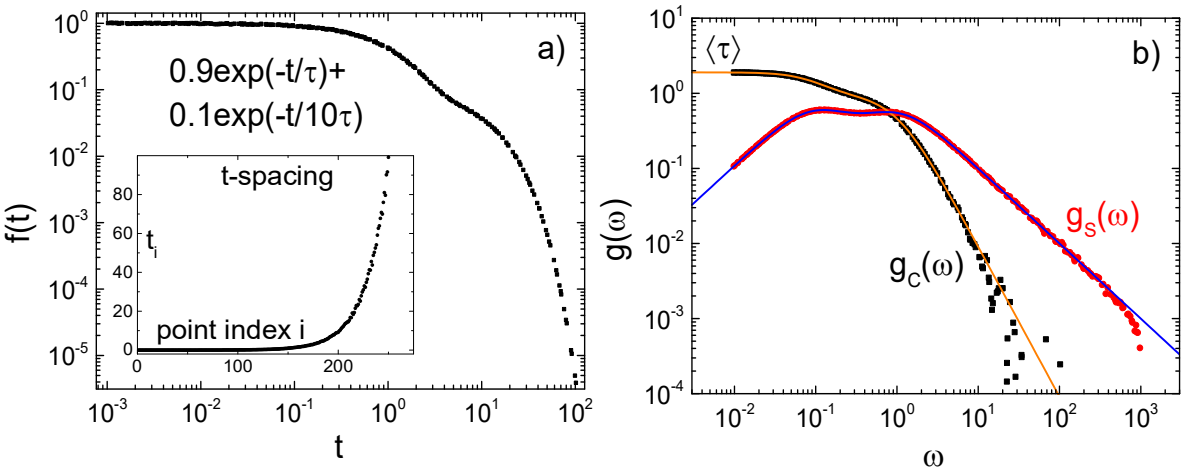

Figure 1. a) Real-valued biexponential time signal $f(t_i)$ according to eq. 12. The inset shows the nonuniform spacing of the time values t_i. b) Corresponding nDFT's , $g_C(\omega)$ and $g_S(\omega)$. The solid lines reflect the analytical expressions eq. 14.

The prefactors of unity in Eq. 1 are chosen to accommodate $g_C(0) = \langle\tau\rangle$ and $\int_0^\infty g_C(\omega)\mathrm{d}\omega = \pi/2$. Consequently, when applying nDFT twice, yields the original function shifted by a factor of $\sqrt{2}$ i.e.

$$nDCT^2\big(f(t_i)\big) = nDST^2\big(f(t_i)\big) = \sqrt{2}f(t_i) \tag{15}$$

IV. Usage of the Python script

The program calculates the Discrete Fourier Transform of nonuniformly spaced data (nDFT) in terms of the Nonuniform Discrete Cosine Transform (nDCT) and the Nonuniform Discrete Sine Transform (nDST). The code is intentionally kept short and simple. The only thing needed is a 2-column ASCII file separated by "tab" with the first column containing the x-values $\boldsymbol{t_i}$ and the second one the y-values $\boldsymbol{f(t_i)}$**.** Remove headers and special symbols (NaN etc.) and copies of data points (especially two consecutive $\boldsymbol{t_i}$ with the same value), to avoid division-by-zero. Here is an example:

0.00010375284158180127 0.9999055897634689
0.00010423174293933036 0.9999051540081028
0.00010471285480508996 0.9999047162415895
0.00010519618738232224 0.9999042764546475
0.00010568175092136585 0.9999038346379535
…and so on…

From the Python, side *numpy* is required. *tkinter* can be used to Browse through the file system and specify an input file. Alternatively, the input file can be specified directly in the script, by uncommenting and changing the line

filename='C:/… /inputfile.txt'

accordingly. Two new files (ASCII 2-column arrays) will be created by the script in the directory of the input file, containing the nDCT and the nDST, respectively.

filename_DCT.pef

filename_DST.pef

As Python is comparatively slow, and given the complexity $O(N^2)$, computation might take several seconds. Using two nested for-loops for computation is unfavorable but computation

can be accelerated using parallelization (not implemented in this version). In future work, C code will be implemented to accelerate computation.

V. Appendix: Calculation of the nDST:

$$g_S(\omega) = \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} f(t) sin(\omega t) dt$$

(A1)

$$\int_{t_i}^{t_i+1} f(t) sin(\omega t) dt = \frac{-1}{\omega} [f(t_{i+1}) cos(\omega t_{i+1}) - f(t_i) cos(\omega t_i)] + \int_{t_i}^{t_{i+1}} \frac{df(t)}{dt} \frac{cos(\omega t)}{\omega} dt$$

(A2)

$$\int_{t_i}^{t_i+1} f(t) sin(\omega t) dt = \frac{-1}{\omega} [f(t_{i+1}) cos(\omega t_{i+1}) - f(t_i) cos(\omega t_i)] + \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} \times \frac{sin(\omega t_{i+1}) - sin(\omega t_i)}{\omega^2}$$

(A3)

$$g_S(\omega) = \sum_{i=0}^{N-1} \frac{-1}{\omega} [f(t_{i+1}) cos(\omega t_{i+1}) - f(t_i) cos(\omega t_i)] + \sum_{i=0}^{N-1} \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} \times \frac{sin(\omega t_{i+1}) - sin(\omega t_i)}{\omega^2}$$

(A4)

Again, only the first $i = 0$ and the last one $i = N - 1$ survive, leading to eq. 8.

VI. References

[1] Filon, L. N. *Proc. Roy. Soc. Edinburgh* 1928, 49, 38-47.

[2] Blochowicz, T. Ph.D. thesis, University of Bayreuth, Germany, 2003.

[3] Rivera, A.; Blochowicz, T.; Gainaru, C.; Rössler, E. A. J. Appl. Phys. 2004, 96, 5607.