Computational Physics, Project 5
Discrete Fourier Transform

Ena Salihovic

Jacobs University Bremen

Due April 26th, 2022

1 Background

The Fourier transform is an operation F that acts on the function f, and the following pair of functions can be defined:

$$F[f](\omega) \equiv \tilde{f}(\omega) \equiv \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) \exp(-i\omega t) dt \tag{1}$$

$$f(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} d\omega \tilde{f}(\omega) \exp(i\omega t) \tag{2}$$

known as the Fourier Transform Pair. [1]

Among other useful properties, the convolution of two functions $f(t)$ and $g(t)$ is the function $(f * g)(t)$ and is defined as follows:

$$(f * g)(t) \equiv \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \tag{3}$$

The Fourier transform of the convolution takes the following form:

$$F[f * g] = \tilde{f}(\omega)\tilde{g}(\omega) \tag{4}$$

This definition is useful to define the autocorrelation function $R(t)$, which is only the convolution of the function with itself. It is defined as:

$$R(t) \equiv \int_{-\infty}^{\infty} d\omega S(\omega) \exp(-i\omega t) \tag{5}$$

where $S(\omega) \equiv \tilde{f}(\omega)\tilde{f}^*(\omega)$ is the power spectral density of f(t), i.e. original timeseries, and will be used later in the project.

1.1 DFT

The discrete Fourier transform, DFT, is a mathematical technique used to convert temporal data into frequency domain data. The fast Fourier transform, FFT, is an algorithm for computing the DFT, which achieves high speed by storing and reusing results of computations as it progresses. [2] It has a more efficient algorithmic complexity, namely $O(n\log(n))$, whereas that of DFT is $O(n^2)$. The DFT tells us which frequencies or "notes" to expect in our signal.

The Fourier transform takes a signal in the time domain and turns it into a spectrum, i.e. a set of frequencies with corresponding complex values.

Here, a discrete Fourier transform pair can also be computed as follows:

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} \exp\left(2\pi i \frac{kn}{N}\right) g_k \tag{6}$$

$$g_n = \sum_{n=0}^{N-1} \exp\left(-2\pi i \frac{kn}{N}\right) f_k \tag{7}$$

where we can introduce $W_N \equiv \exp(-2\pi i/N)$ to simplify the equations.

DFT can be seen as a matrix multiplication:

$$\begin{aligned} \mathbf{g} &= F_N \mathbf{f} \\ &= \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} \begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix} \end{aligned} \tag{8}$$

, where g is the DFT, F_N is the so-called DFT matrix [3].

1.2 FFT

The main advantage of FFT is to achieve the appearance of the empty submatrices 0 in Equation (8), which reduces the number of multiplications, and yield to a better algorithmic complexity.

2 Project

Task 1:
The sound of a clean g note was taken from an online source. The wavefile module provided by python was used to read the file.

Task 2 - plotting the signal:
The entire timeseries as well as the short section ($0.1s$) were plotted in 2 separate panels, as can be seen in Figure (1).

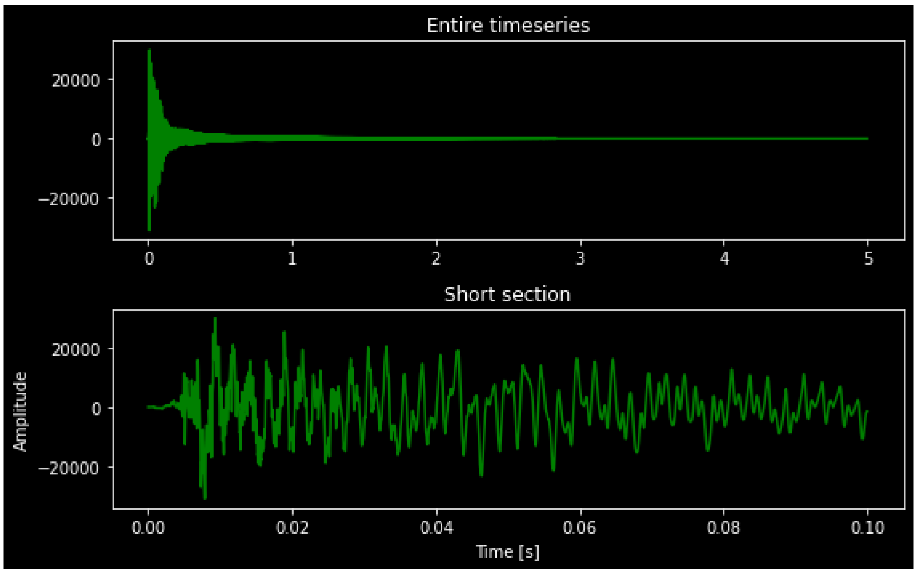

Figure 1: Graph showing the entire timeseries of the original signal, as well as a short $0.1s$ section

From the plot, it can be observed that the signal exhibits repeated pattern. A dominant frequency can be observed, and its value is approximately 420Hz, since there is around 42 periods on the $0.1s$ interval.

Task 3 - DFT:
A discrete Fourier transform was implemented. All values of exponentials W_N^k for $k \in \{0,...,N-1\}$ were stored in an N-element array (called e in $DFT.py$) and these values were re-used. This function was used to go from the time-domain seen in Figure (1) to frequency domain, as can be seen in Figure (2).

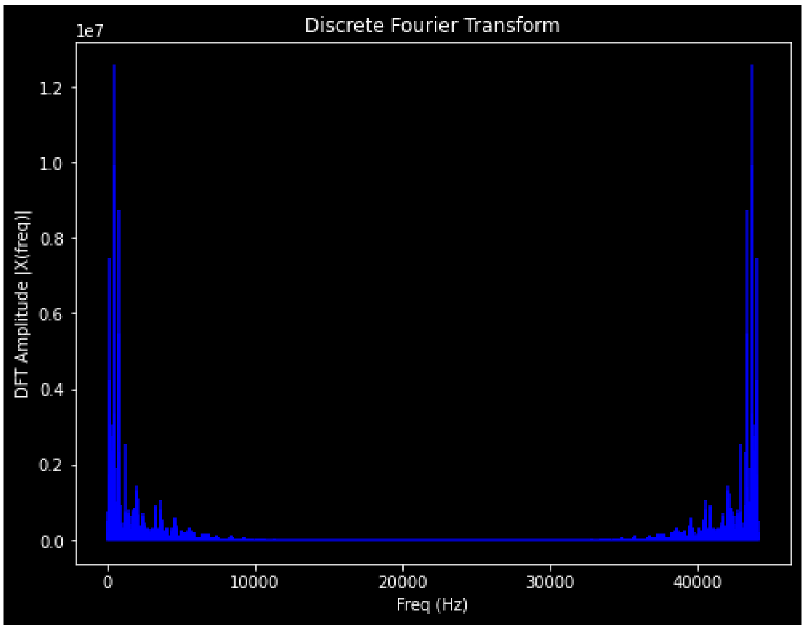

Figure 2: Graph showing amplitudes in the frequency domain, obtained via Discrete Fourier Transform

Task 4:
Here, python's implementation of the Fast Fourier Transform was imported. FFT was applied to original data to convert go from time- to frequency-domain (Figure (3) - on the left) as well as to obtain the power spectrum, defined in the section Background, for a section of the used signal (Figure (3) - on the right).

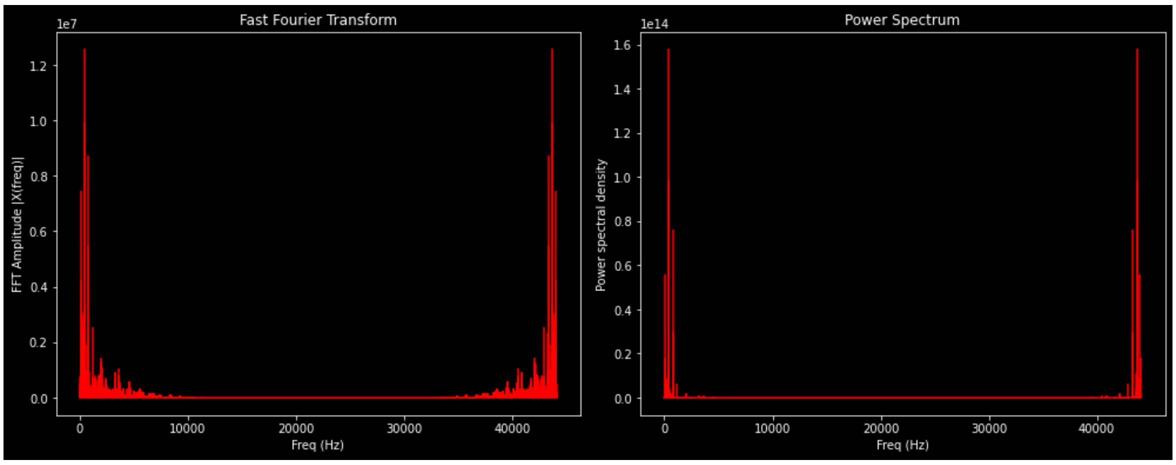

Figure 3: Graph showing amplitudes in the frequency domain, obtained via Discrete Fourier Transform

The spectrum matches the intuition about the recorded note; the spectrum is symmetric (normally around 0, but since the absolute values were plotted, it is symmetric around frequency of $\sim 210Hz$). The DFT and FFT give the same spectrum, indicated by Figure (2) and left Figure (3). On the right of Figure (3), a maximum frequency at around $420Hz$ can be observed, which matches the initial esimate about ω_{max}.

Task 5:
Both user-defined DFT and python's FFT were timed, using perf_counter() function from the time module. Although the project asked to compute for sample sizes of $N = 10^a$ where $a \in 1,5$, the computation was not possible for the sample size $N = 10^5$, and the program would crash, I assume due to the large computational power required. Therefore, only 4 sample sizes were used, and sometimes they were adjusted (e.g. from $N = 1000$ to

$N = 1003$ to match the number of data points sampled). Finally, both DFT and FFT were applied to the original signal and the scaling of computing time vs. N was determined, see Figure (4).

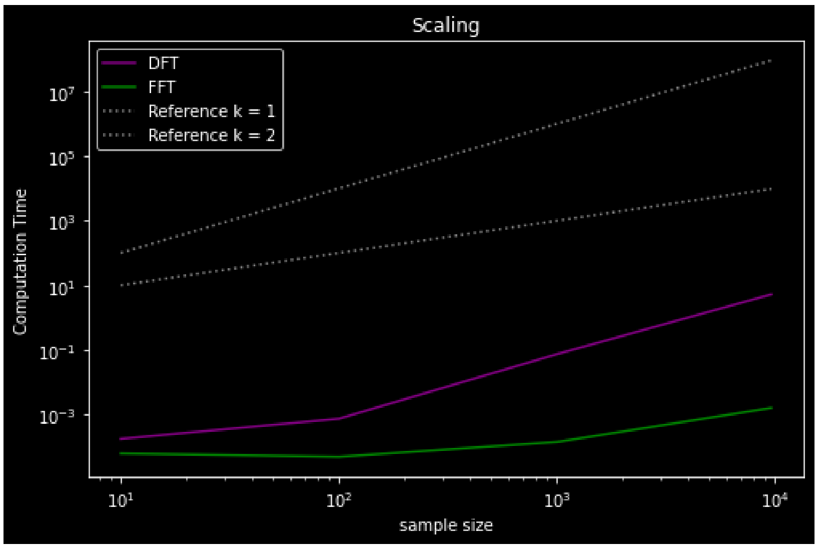

Figure 4: Graph showing the computation time as a function of sample size, for both FFT and DFT

Plotting was done on a log-log scale to extract the power-law exponent for DFT. It can be observed that for small sample sizes, DFT follows linear complexity, but for sample sizes larger than 100, it grows as n^2. On the other hand, computation time of FFT grows linearly with sample size. It can also be observed, that as the sample size increases, the two complexities diverge more, making the FFT ultimately more efficient.

Task 6:
FFT was applied to the entire timeseries. Then, FFT was applied back to the time domain, and by plotting (Figure (5)), it was verified that the timeseries is recovered.

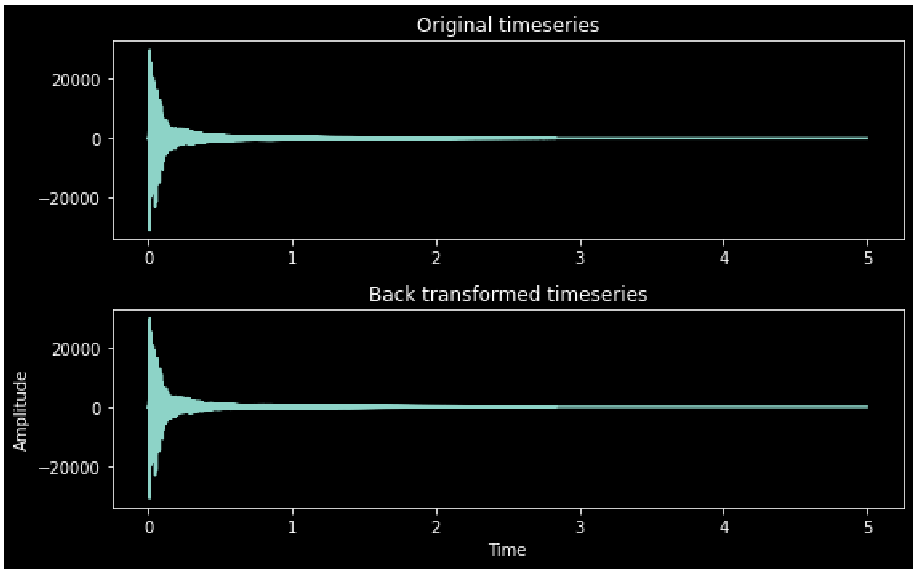

Figure 5: Graph showing the original time series and back-transformed time series

Then, a fraction $f_a = 1-(1/2)^a$ with $a \in 1, 10$ of the original frequency data was removed by setting the fraction f_a of lowest-amplitude Fourier modes to zero (as measured by the power spectral density $S(\omega)$). Code-wise, this meant iterating over the matrix of size $10xlen(S)$ where S is the array storing values for power spectral density, and setting the specific fraction of the lowest amplitudes to 0. Then, from the remaining, non-zero amplitudes the system was back-transformed to the time domain and a short 0.1 sec section of the resulting timeseries was plotted (see Appendix - Figure (7)).

As more data was removed, with increasing a, the more of the original data, so more information about the original signal was being lost.

Using the following command: wavefile.write('filename_out.wav', samplerate, timeseries) an attempt was made to write a .wav file for each a, but certain errors were generated in such a case. I would however assume, based on the shape of the plot in Figure (7), that the sound gets more distorted with larger a, as more of the original data is being lost. Furthermore, I would assume that the sound would get shorter and more high pitched, as only the higher frequencies are preserved.

Task 7:
Finallt, FFT was performed only for each 10th or 100th sample in the original timeseries. Then, $\widetilde{f}(\omega)$ was obtained using each of these two smaller sample sizes and an attempt was made to recover $f(t)$ by back-transforming, i.e. applying $ifft$ from $scipy$ library. The $0.1s$-section of the entire timeseries was plotted to to compare the data for the different cases, and can be seen in Figure (6).

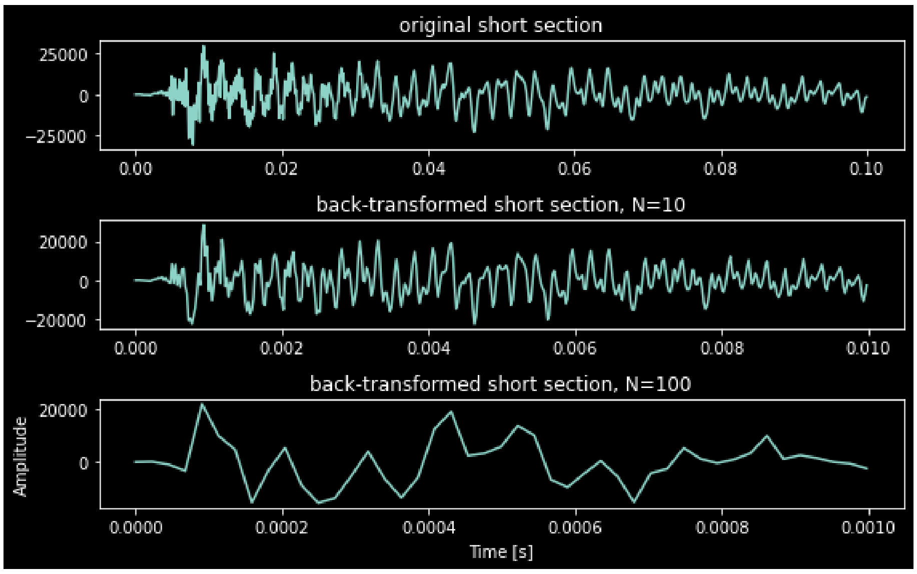

Figure 6: Graph showing the original time series for the short section, and back-transformed short section for sample sizes N=10 and N=100

Evaluating the results in context of the Shannon-Nyquist Theorem, which states that a function which contains no frequencies higher than ω_{max}, is completely defined by the data points spaced by $\frac{1}{2\omega_{max}}$ seconds. Therefore, the sample rate should be higher than $2\omega_{max}$. For the specific signal used in this project, the $\omega_{max} = 420Hz$, the time interval between two data points is $\frac{1}{840}s$ at maximum, and the minimum sampling rate is $840Hz$.

If every 10^{10} element is sampled, then the function is completely defined according to the SN-theorem, but this is not the case if every 100^{th} point is sampled.

3 Green's theorem

Forced harmonic oscilator can be described with the following:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + \beta\frac{\mathrm{d}y}{\mathrm{d}t} + \omega^2 y = f(t) \tag{9}$$

where
$$\frac{\mathrm{d}^2G(t;\tau)}{\mathrm{d}t^2}+\beta\frac{\mathrm{d}G(t;\tau)}{\mathrm{d}t}+\omega^2G(t;\tau)=\delta(t-\tau) \tag{10}$$
defines the Green's function $G(t;\tau)$. Obtain the Green's function $\widetilde{G}(\omega;\tau)$ by Fourier transform.

Applying Fourier Transform to Equation (10) results in the following:
$$(j\omega)^2\widetilde{G}(\omega,\tau)+\beta(j\omega)\widetilde{G}(\omega,\tau)+\alpha^2\widetilde{G}(\omega,\tau)=\exp(j\omega\tau) \tag{11}$$
where α is the same as ω in Eq. (10), but was renamed to avoid confusion between frequency in the frequency-domain. Factorizing Equation (11) gives:
$$\widetilde{G}(\omega,\tau)((j\omega)^2+\beta(j\omega)+\alpha^2)=\exp(-j\omega\tau) \tag{12}$$
$$\widetilde{G}(\omega,\tau)=\frac{\exp(-j\omega\tau)}{\beta\left(j\omega+\frac{\alpha}{2\sqrt{\beta}}\right)^2} \tag{13}$$
Equation (13) gives us the Fourier Transform of the Green's function, where the following property was used to solve the expression:
$$x(\tau-t_0)\overset{FT}{\rightarrow}X(\omega)\exp(-j\omega t_0) \tag{14}$$

Applying Inverse Fourier Transform to the $\widetilde{G}(\omega,\tau)$ gives:
$$F^{-1}[\omega,\tau]=F^{-1}\left[\frac{1}{\beta\left(j\omega+\frac{\alpha}{2\sqrt{\beta}}\right)^2}\right](t-\tau) \tag{15}$$
$$G(t,\tau)=\frac{1}{\beta}(t-\tau)\exp\left(-\frac{\alpha}{2\sqrt{\beta}}(t-tau)\right)u(t-\tau) \tag{16}$$
where $u(t-\tau)$ is a step function. With this, we arrive back to the time-domain of the Green's function.

The following plots were obtained for forced harmonic oscillator, by applying FFT to the Green's function.

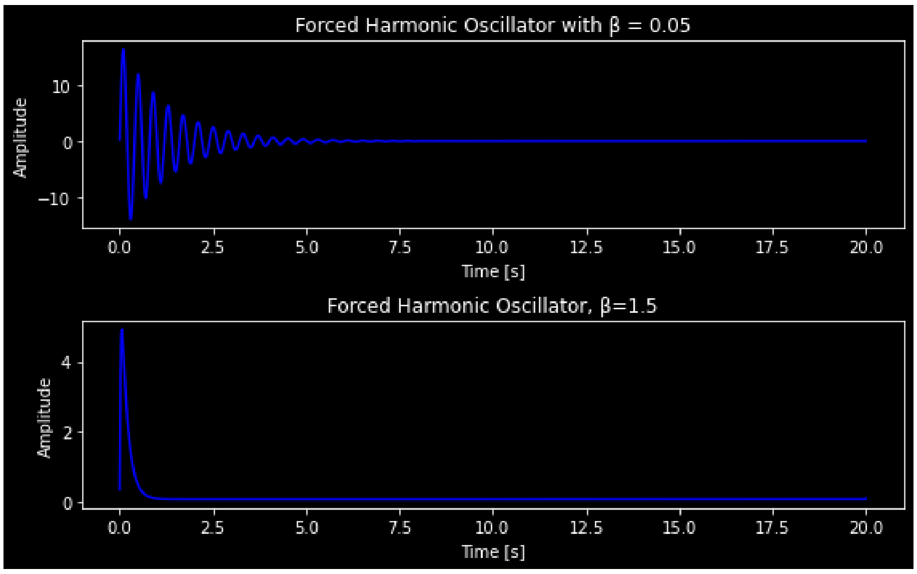

Figure 7: Graph showing the forced harmonic oscillator after applying FFT to Green's function

These plots behave as expected; namely, in the underdamped case $\beta=0.05$, the amplitude decreases exponentially, whereas in the case of overdamping $beta=1.5$, there are no oscillations.

4 Bibliography

[1] Haerter, Jan O. "Chapter 3: Fourier Theory." Computational Physics II, retrieved from https://elearning.jacobs-university.de/pluginfile.php/54949/mod_folder/content/0/Chapter%203%2C%20Fourier%20Analysis.pdf?forcedownload=1

[2] O'Reilly Online Learning. 2022. Elegant SciPy. Retrived from https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch04.html

[3] Computing the DFT Matrix. (2020, March 29). [Video]. YouTube. https://www.youtube.com/watch?v=Xw4voABxU5c

5 Appendix

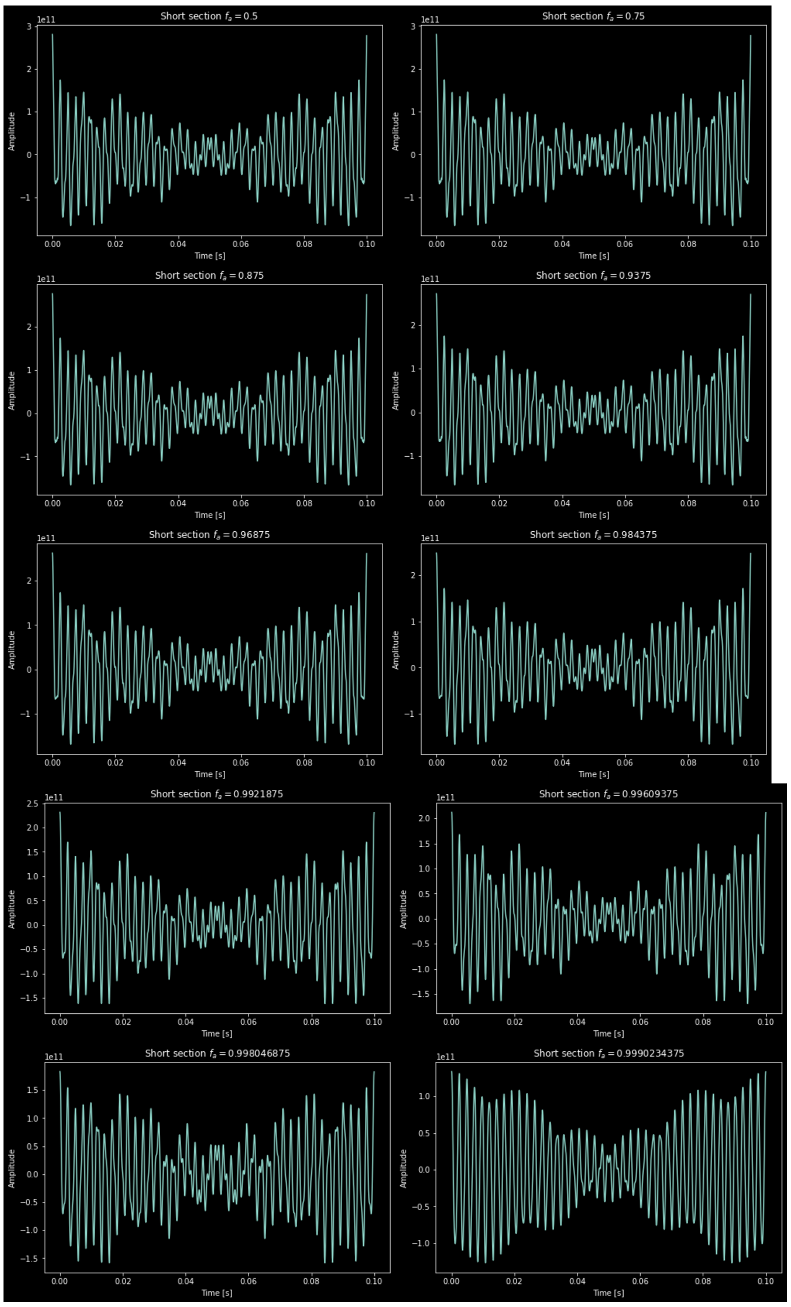

Figure 8: Graph showing the original time series and back-transformed time series