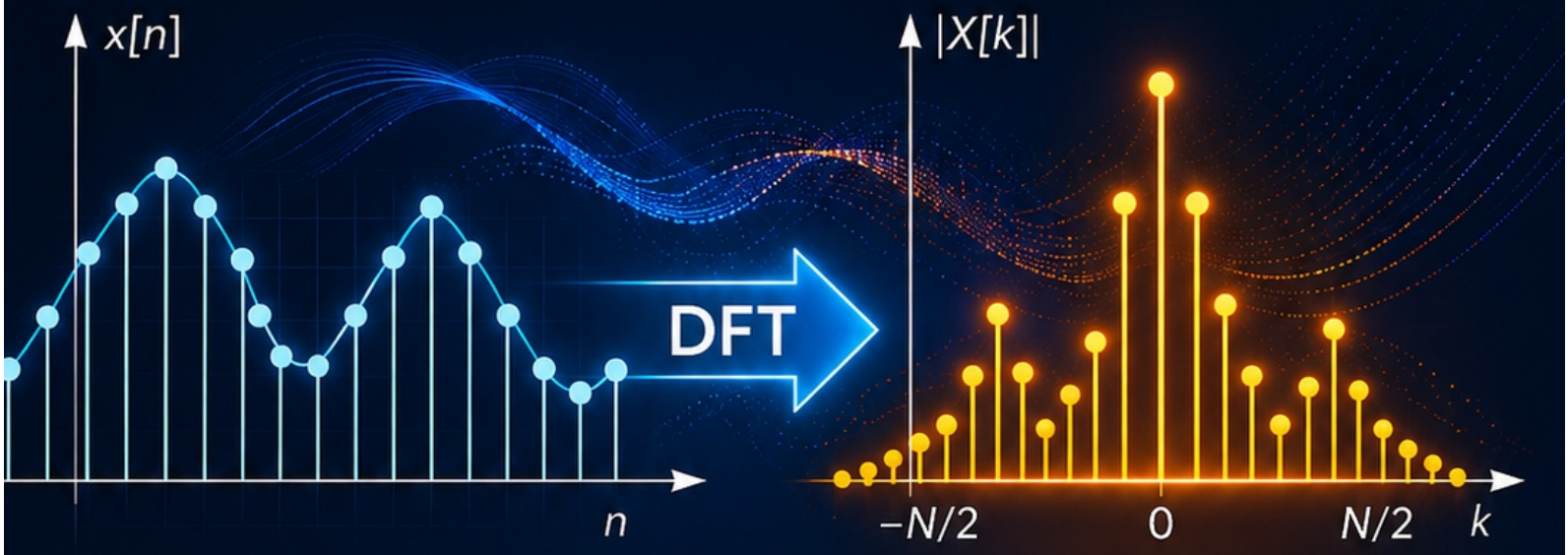
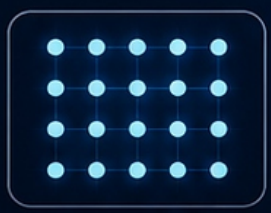
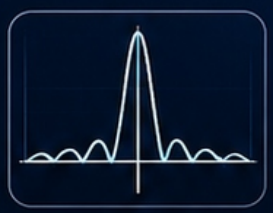
An Introduction to the Discrete Fourier Transform and Its Applications in Signal Processing

MATLAB and Python/NumPy edition




Sampling & Aliasing


Spectral Leakage

Phase & fftshift


DFT Properties


Applications in Practice

Laurent Nony • Jean-Marc Themlin

MATLAB® | python™ NumPy

An Introduction to the Discrete Fourier Transform and Its Applications in Signal Processing

MATLAB and Python/NumPy edition — v1.0

Laurent Nony, Jean-Marc Themlin

2026-07-20

This course monograph introduces the main concepts underlying the Discrete Fourier Transform (DFT), a fundamental tool in signal processing. The theoretical foundations are presented with a minimal amount of mathematical formalism. The emphasis is placed on the practical implementation of the DFT through paired MATLAB and Python/NumPy examples.

Table of contents

Included with this course monograph:

- Complete PDF course monograph
- Executable Jupyter notebook (.ipynb)
- HTML version of the course
- High-resolution figures

The notebook contains all Python/NumPy examples presented throughout the text and allows readers to reproduce, modify, and extend the computations discussed in the monograph.

Companion materials and support available upon request:

signal.processing.course@gmail.com

When requesting the companion materials, please attach your Leanpub purchase receipt.

Released edition:

This edition builds on earlier teaching material made available in open access through the French HAL archive (https://hal.science/), and extends it with revised explanations, paired MATLAB and Python/NumPy examples, and additional practical material.

About the authors:

Laurent Nony *is Associate Professor (Maître de Conférences Hors Classe) in Physics at Aix-Marseille Université (France) and a member of the Institute of Materials, Microelectronics and Nanosciences of Provence (IM2NP, CNRS UMR 7334, France), where he is currently leading the "Nanostructuration" research group. He received his Ph.D. in Physics from the University of Bordeaux (France) in 2000 and subsequently held postdoctoral research positions at the IBM Zurich Research Laboratory (Switzerland) and the University of Basel (Switzerland), before joining Aix-Marseille Université in 2005. He obtained his Habilitation (HDR) in 2013. His research activities focus on experimental and computational nanophysics, scanning probe microscopy, signal acquisition,*

instrumentation, and data analysis. Throughout his academic career, he has supervised numerous student projects and developed original scientific instrumentation as well as numerical simulation tools. For more than twenty years, he has taught physics, instrumentation, numerical methods, signal processing, sensors, and scientific computing at undergraduate and graduate levels. He has also created a wide range of pedagogical resources, including lecture notes, laboratory projects, online courses, and educational videos. His teaching philosophy emphasizes the connection between theory, experimentation, and practical implementation. This course monograph reflects that approach by combining conceptual explanations with concrete MATLAB and Python/NumPy examples, enabling students to develop both intuition and practical skills in signal processing. He is the author, or co-author, of more than 50 peer-reviewed research articles, book chapters and patents, and developer of several digital pedagogical resources dedicated to signal processing, scientific computing, and experimental physics.

Jean-Marc Themlin *is Professor of Physics at Aix-Marseille Université (France), affiliated with the Institute of Materials, Microelectronics and Nanosciences of Provence (IM2NP, CNRS UMR 7334, France), where he is currently leading the PHaNO scientific department. As an experimental physicist specialized in the electronic structure of solids, surfaces and nanomaterials, he benefits from over 35 years of experience in higher education and research. His research interests focus on Angle-Resolved Inverse Photoemission Spectroscopy (ARIPES) to probe unoccupied electronic states. His work has led to significant advances in understanding pristine and N-doped epitaxial graphene on SiC, two-dimensional Mott insulators, monolayers of π-conjugated molecules (fullerenes,…) adsorbed on surfaces,… With more than 60 top-tier publications, his career is driven by a profound commitment to pedagogy: the conviction that training the next generation of physicists requires an active, experimental, and inclusive approach. Head of the Experimental Physics Service of the Physics Department of AMU since 2002, he leads this multi-site service, ensuring the quality and modernization of practical labs for all physics students through resource sharing and equipment upgrades. Adept of "Teaching Through Practice", his teaching philosophy centers on reforming traditional methods to prioritize "well-structured minds over well-filled ones" through providing more active learning opportunities to students. To this aim, he restructured numerous courses (Signal theory, Signal & image Processing, Electronics,…) to reduce formal lecture time in favor of hands-on labs, experimental projects, and case studies, utilizing open-source tools and collaborative platforms. Applying these recipes, he created an interdisciplinary "Digital Physics" course tailored for first year B.Sc. students with a computer science background. More recently, he designed and launched the enhanced "Initiation to Scientific Research" track for Bachelor's students in their second and third year, allowing them to immerse themselves in "close-to-real" research. In 2025, he initiated the PhysLab project, funded by the University foundation AMidex, to create dedicated spaces for undergraduate experimental research, bridging the gap between standard lab sessions and authentic laboratory research. Beyond the lab and the classroom, he also held structural leadership roles within the faculty, especially as Head of the Physics Department of AMU. Serving two terms (2018–2024), he led the largest department in the Faculty of Sciences, overseeing a complete redesign of the degree programs (Bachelor's and Master's).*

4 The Discrete Fourier Transform (DFT)

The **Discrete Fourier Transform** (DFT) of a generic discrete-time signal $z_s(t_s)$ can be viewed as an approximation of its DtFT (Eq. (14c)) obtained by:

- truncating the summation over n;
- discretizing the frequency variable f;
- retaining only one period of the f_s-periodic spectrum.

4.1 Truncating the summation: temporal windowing

When dealing with realistic discrete-time sampled signals, it is impossible to handle an infinite number of samples. **The discrete-time sampled signal to be analyzed, $z_s(t_s)$, always consists of a finite number of samples.** Let us assume that the sampling operation yields N samples of the discrete-time signal. Hence the truncation of the summation over n in Eq. (14c).

The N samples are obtained by **windowing operation** of $z_s(t_s)$, i.e. the multiplication of the signal by a windowing function $w(t)$ to form the **windowed sampled signal**:

$$z_{sw}(t_s) = z_s(t_s)w(t) \tag{15}$$

$w(t)$ may be chosen as a rectangular window (though this is not the only choice), whose width sets the **windowing duration** T_w, i.e. the duration of the time support of $z_{sw}(t_s)$. $w(t)$ and its FT $W(f)$ form a FT pair defined as:

$$w(t) = \begin{cases} 1, \forall t \in [0; T_w] \\ 0, \text{ otherwise} \end{cases} \rightleftharpoons W(f) = T_w \frac{\sin(\pi T_w f)}{\pi T_w f} e^{-j2\pi f \frac{T_w}{2}} := T_w \text{sinc}(T_w f) e^{-j\pi f T_w} \tag{16}$$

By definition:

$$T_w := NT_s \tag{17}$$

Now, the n^{th} sample, with $n \in \mathbb{N}, n \in [0; N-1]$, of the windowed discrete-time support t_s is $t_s[n] := nT_s$, and the corresponding sample of $z_{sw}(t_s)$ is $z_{sw}(t_s[n]) \rightarrow z_{sw}[n]$.

The figure below completes the previous figure about the sampling process of a continuous-time signal into its sampled and windowed version.

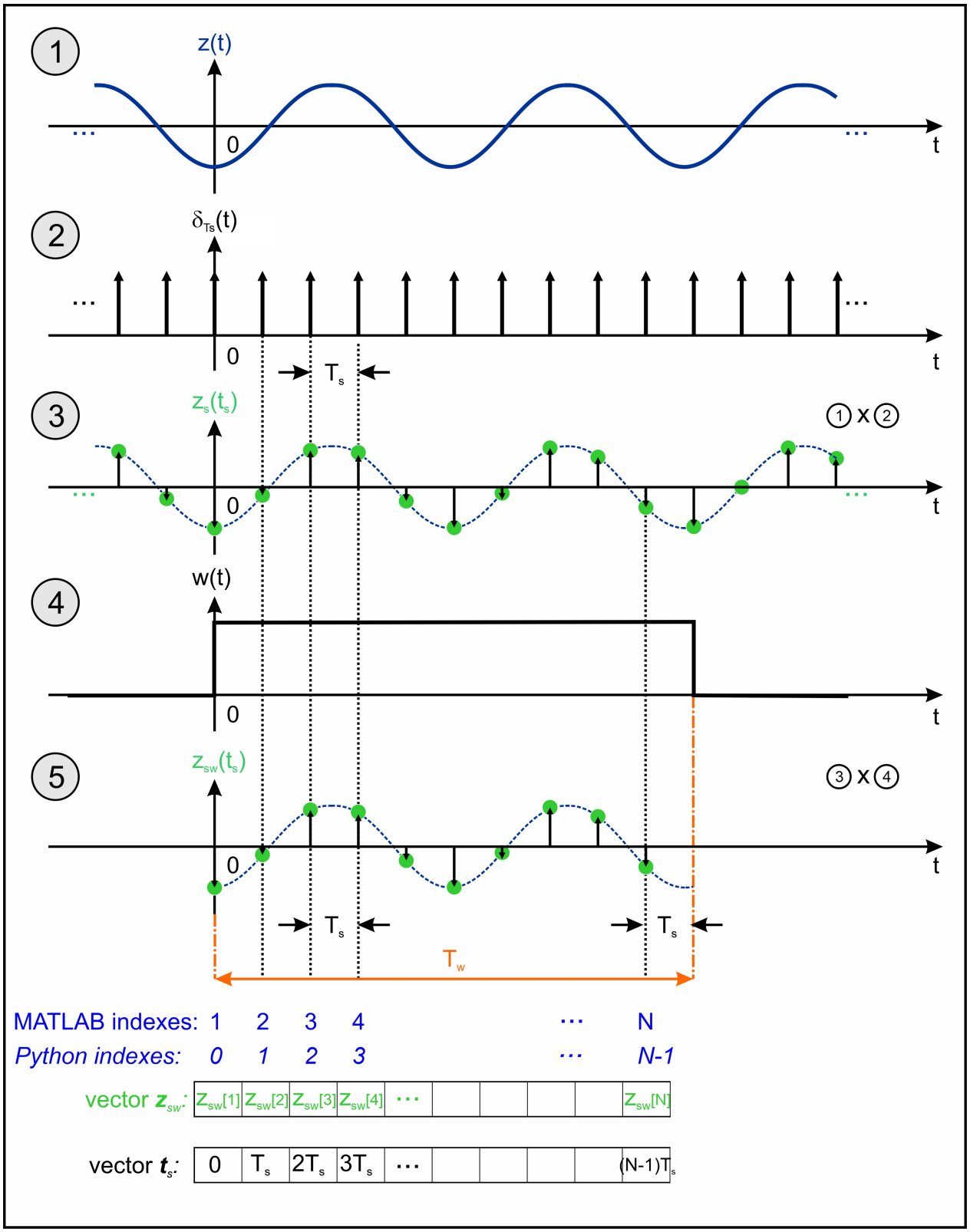


Figure 2: Sampling and temporal windowing process.

Important note: *Although the last sample of the windowed discrete-time support is*

written as $(N-1)T_s$, the total duration of the signal is $T_w = NT_s$. In other words, each sample covers a duration of T_s in time. In the example illustrated here, since the signal is T-periodic and sampled, the window duration T_w has been optimally chosen as a multiple of T, i.e. $T_w = kT,\ k \in \mathbb{N}$. *However, note that the last sample $z_{sw}[N-1]$ is not equal to the first sample $z_{sw}[0]$.*

4.2 Revisiting the expression of the DtFT

The windowing operation has a non-trivial effect on the DtFT, which forces us to reconsider its expression (Eq. (14c)).

For this purpose, we start from the **FT of the windowed continuous-time signal** $Z_w(f) := FT\{z(t)w(t)\}$, thus: $z(t)w(t) \rightleftharpoons Z_w(f)$. Recall that: $w(t) \rightleftharpoons W(f)$ (see Eq. (16)) and $z(t) \rightleftharpoons Z(f)$.

We then have:

$$Z_w(f) := FT\{z(t)w(t)\} = Z(f) * W(f) \rightarrow Z_w(f) = \sum_{n=0}^{N-1} z_{sw}[n] e^{-j2\pi f n T_s} T_s * W(f) \tag{18}$$

Taking into account Eq. (12), the Fourier transform of the sampled and windowed original signal reads:

$$Z_{sw}(f) := Z_w(f) * f_s \delta_{f_s}(f) = \sum_{n=0}^{N-1} z_{sw}[n] e^{-j2\pi f n T_s} T_s * W(f) * f_s \delta_{f_s}(f) \tag{19a}$$

$$= \sum_{n=0}^{N-1} z_{sw}[n] e^{-j2\pi f n T_s} * W(f) * \delta_{f_s}(f) \tag{19b}$$

$$= \sum_{n=0}^{N-1} z_{sw}[n] e^{-j2\pi n \frac{f}{f_s}} * W(f) * \delta_{f_s}(f) \tag{19c}$$

The previous equation constitutes the most complete expression of the DtFT of a continuous-time signal $z(t)$, whose windowed and discrete-time version is $z_{sw}(t_s)$.

The figure below illustrates the influence of the temporal windowing, both in the time domain and in the frequency domain by FT pairing. It is important to notice that we have taken a symmetric interval of definition of the continuous-time signal $z(t)$.

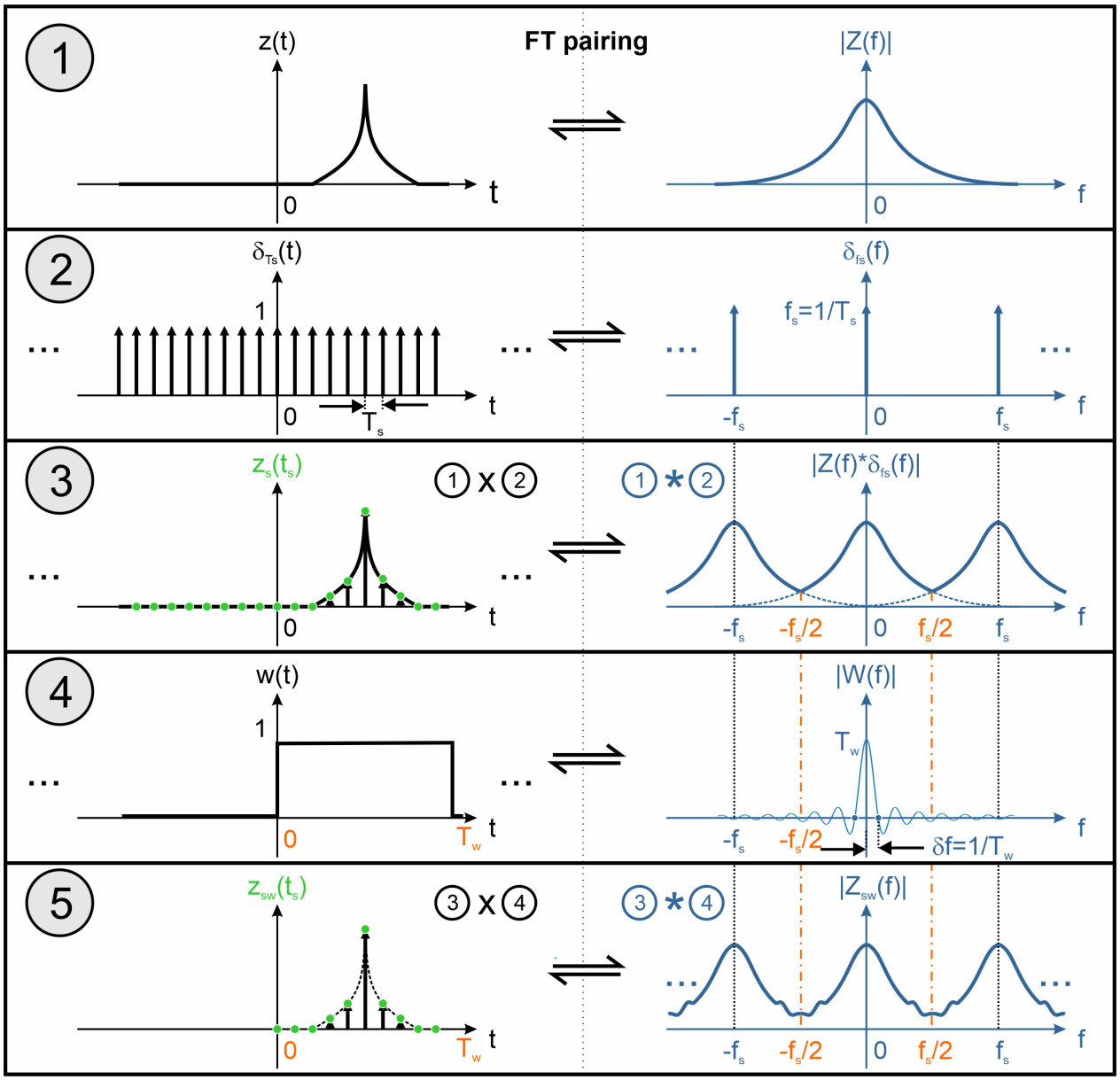


Figure 3: Effect of temporal windowing in the time and frequency domains.

4.3 Frequency sampling

4.3.1 Sampling the frequency domain

In order to calculate numerically the DFT on a finite number of points, one period of the DtFT $Z_{sw}(f)$ one period of the DtFT (i.e. one f_s-wide period) should now be discretized, and it is customary to perform it also on N points, hence a sampling interval in the frequency domain of $\delta f = \frac{f_s}{N}$.

Therefore the continuous frequency is sampled as:

6 Computation of the DFT

In most software packages, the DFT defined by Eq. (35) is computed using the *fft* routine. For example, if $z_{sw}[n]$ is a windowed sampled signal represented by a vector of N samples in MATLAB, its DFT is obtained with the instruction *fft*(z_{sw}).

Below, we give several implementation examples, compare the direct calculation with the result returned by *fft*, and highlight some common pitfalls.

6.1 Raw computation

We illustrate the calculation of the DFT with a simple cosine waveform signal of period T_1 (frequency $f_1 = 1/T_1$) over a windowing duration: $T_w = 1$ s:

$$z(t) = \cos(2\pi f_1 t + \pi/3), \forall t \in [0; 1] \tag{36}$$

whose FT is well-known:

$$Z(f) = \frac{1}{2}\left[\delta(f - f_1)e^{j\pi/3} + \delta(f + f_1)e^{-j\pi/3}\right] \tag{37}$$

On the other hand, the signal being T_1-periodic, it can be expanded in Fourier series, whose set of coefficients are simply:

$$Z_1 = Z^*_{-1} = \frac{1}{2}e^{j\pi/3} \tag{38}$$

The MATLAB code below implements Eq. (35) and compares the result to the one returned by the "*fft*" function. We discuss the DFT spectra in the range $\left[0; \frac{f_s}{2}\right[$ (i.e. $\left[0; \frac{f_s}{2} - \delta f\right]$) for now.

6.1.1 Illustrative code

The code below computes one half of the two-sided spectrum of a sinusoidal signal, windowed over an integer number of periods, without normalization.

MATLAB version:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Illustration of the fft function from hard coding %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Initializations
% Regular Initializations lines in MATLAB
clear;
close all;
clc;
```

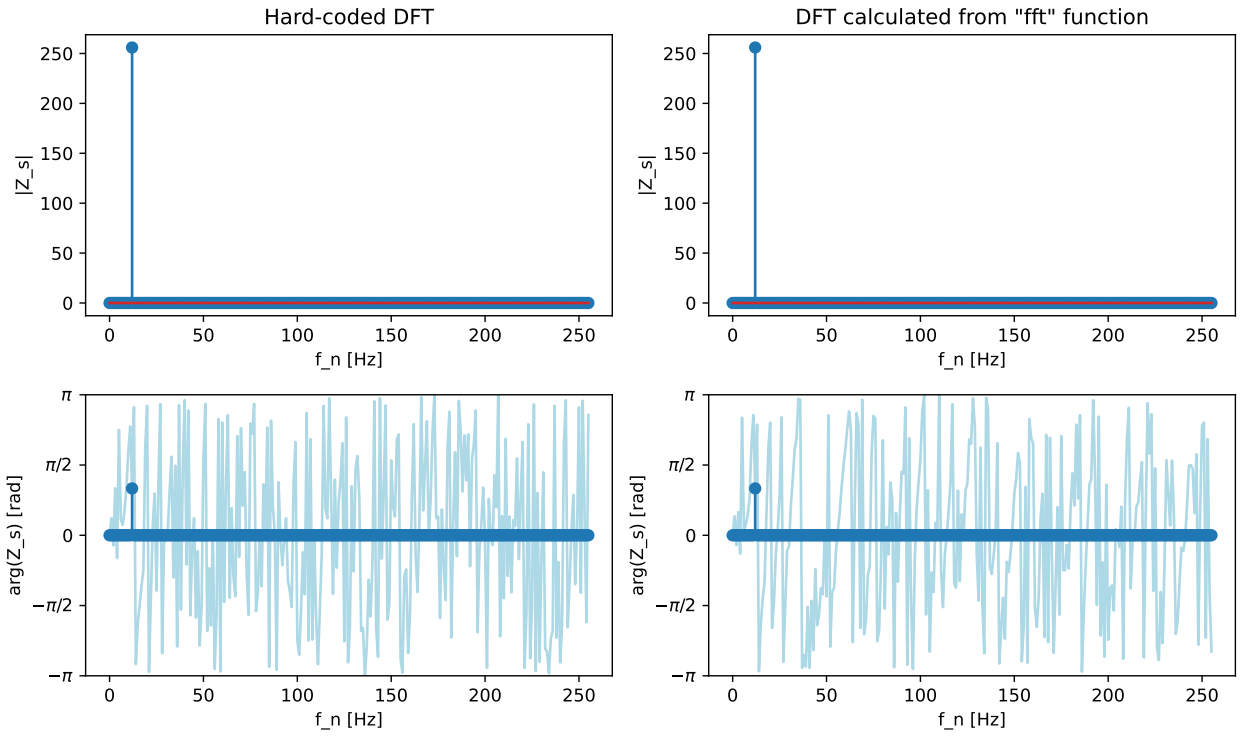
```
%% Parameters
N  = 512;     % number of samples
Tw = 1.0;     % windowing duration (s)
f1 = 12;      % cosine frequency (Hz)

%% Built-in variables
Ts = Tw / N;
fs = 1 / Ts;
df = fs / N;

%% Support vectors
n   = 0:N-1;
t_s = n * Ts;
f_n = (0:N/2-1) * df;

%% Building the sampled signal
z_s = cos(2*pi*f1*t_s + pi/3);

%% DFT calculation
%-----------------------------------------------------------
% 1 - Hard-coded DFT
%-----------------------------------------------------------

Z_s1 = zeros(1,N);

for k = 0:N-1
    k_hat = k/N;
    Z_s1(k+1) = sum(z_s .* exp(-1j*2*pi*n*k_hat));
end

Z_s1 = Z_s1(1:N/2);

phases_Z_s1    = angle(Z_s1);
phases_Z_s1_cl = clean_phases(Z_s1);

%-----------------------------------------------------------
% 2 - fft function
%-----------------------------------------------------------
Z_s2 = fft(z_s);
Z_s2 = Z_s2(1:N/2);

phases_Z_s2    = angle(Z_s2);
phases_Z_s2_cl = clean_phases(Z_s2);


%% Function clean_phases
function phases = clean_phases(coefficients, rtol)
```

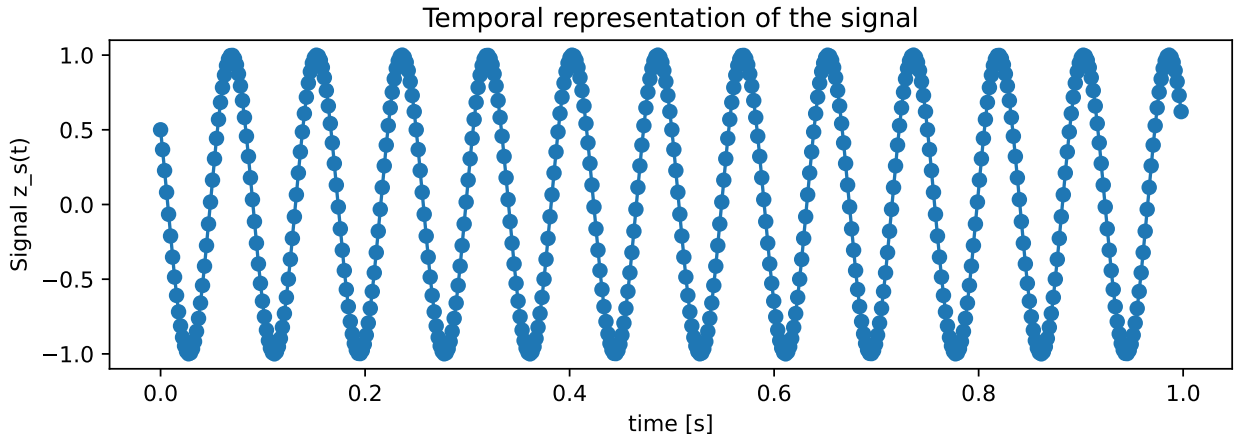
```
stem(f_n,phases_Z_s2_cl,'filled')
hold off

xlabel('f_n [Hz]')
ylabel('arg(Z_s) [rad]')

%% Temporal representation

figure

plot(t_s,z_s,'o-','LineWidth',1.2)

xlabel('time [s]')
ylabel('Signal z_s(t)')
title('Temporal representation of the signal')
grid on
```

Python / NumPy version:

```
import numpy as np
import matplotlib.pyplot as plt

def clean_phases(coefficients, rtol=1e-12):
    """
    Returns the phases of complex Fourier coefficients, setting the phase
    to zero when the corresponding magnitude is numerically negligible.

    Parameters
    ----------
    coefficients : array_like
        Complex Fourier coefficients.
    rtol : float
        Relative threshold with respect to the maximum magnitude.

    Returns
    -------
    phases : ndarray
        Cleaned phase values in radians.
    """
    coefficients = np.asarray(coefficients)

    magnitudes = np.abs(coefficients)
    phases = np.angle(coefficients)

    threshold = rtol * np.max(magnitudes)
    phases[magnitudes <= threshold] = 0.0
```

```
ax[1, 0].set_ylim(-np.pi, np.pi)
ax[1, 0].set_yticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
ax[1, 0].set_yticklabels([r"$-\pi$", r"$-\pi/2$", "0", r"$\pi/2$", r"$\pi$"])

ax[0, 1].stem(f_n, np.abs(Z_s2))
ax[0, 1].set(xlabel="f_n [Hz]", ylabel="|Z_s|",
             title='DFT calculated from "fft" function')
ax[1, 1].plot(f_n, phases_Z_s2, color="#ADD8E6",
              label="Raw numerical phase")
ax[1, 1].stem(f_n, phases_Z_s2_cl, linefmt="C0-",
              markerfmt="C0o", basefmt=" ")
ax[1, 1].set(xlabel="f_n [Hz]", ylabel="arg(Z_s) [rad]")
ax[1, 1].set_ylim(-np.pi, np.pi)
ax[1, 1].set_yticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
ax[1, 1].set_yticklabels([r"$-\pi$", r"$-\pi/2$", "0", r"$\pi/2$", r"$\pi$"])
fig.tight_layout()

plt.figure(figsize=(8, 3))
plt.plot(t_s, z_s, "o-")
plt.xlabel("time [s]")
plt.ylabel("Signal z_s(t)")
plt.title("Temporal representation of the signal")
plt.tight_layout()
plt.show()
```





Conclusion:

With a single non-zero value, the modulus spectrum obtained from the direct implementation of Eq. (35) and the values returned by the fft function are identical.

The moduli feature a single peak at the cosine frequency $f_1 = 12$ *Hz in the range* $\left[0; \frac{f_s}{2}\right[$ *(here* $[0; 256[$ *Hz). However, the peak magnitude does not match the expected Fourier-series coefficient* $|Z_1| = \frac{1}{2}$*; instead, it is equal to* $\frac{N}{2}$ *(here 256, since* $N = 512$*). This point is discussed in the next section.*

While the values of the phase spectra are indeed different, it should be realized that most of them (all but one) are meaningless since they correspond to a vanishing modulus. The only significant phase values ($\pi/3$ *at* $f_1 = 12$ *Hz) are equal, and the other ones could safely be forced to zero. This issue will be discussed in the section 6.4 below. The calculated phases (light blue) have therefore consistently been zeroed when the corresponding modulus was lower than* $\epsilon = 10^{-12}$*, and appear then in dark blue.*

6.2 Normalization: Fourier series & Fourier transform

The discrepancy between the magnitude of the modulus of the computed peaks and the expected one is due to the lack of **normalization** of the DFT, as computed by Eq. (35), or by the *"fft"* function.

The **normalization of the DFT is usually performed upon division of the result of the *"fft"* function by the number of samples** $z_s(t_s)$ **consists of (e.g.** N**).** By doing so, the normalized DFT stands for the Fourier series coefficients of the T_w-periodic or finite duration sampled signal $z_s(t_s)$, whose physical units are those of $z_s(t_s)$.

However, to use the FT to evaluate the (continuous) Fourier transform of an infinite-duration signal, whose physical units are those of the signal multiplied by time (e.g. $V \cdot s$), it is necessary to use the scaling factor $T_s = T_w/N$.

These points are discussed in the following two subsections.

7 Applications

Although the theoretical framework of the DFT has been established for the general case of **non-causal signals, i.e. signals that exist for both negative and positive times,** $t \in \left[-\frac{T_w}{2}; +\frac{T_w}{2}\right]$, the examples considered so far have been built as **causal signals, i.e. signals defined for positive time only**.

In the following examples, we focus on periodic and non-periodic non-causal signals.

7.1 Fourier series coefficients of a periodic, non-causal signal

We start with periodic, non-causal signals.

7.1.1 Problem positioning

- **Signal 1:** We consider the former T_1-periodic sine waveform signal defined over a windowing duration $T_w = 1$ s, but we arbitrarily define it as:

$$z(t) = \sin(2\pi f_1 t), \forall t \in \left[-\frac{1}{2}; \frac{1}{2}\right] \tag{48}$$

Hence, the non-causal nature of that signal.

7.1.1.1 Illustrative code

The code below reproduces the previous code for that signal, but, since the signal is periodic, we now focus only on the Fourier series coefficient spectrum.

MATLAB version:

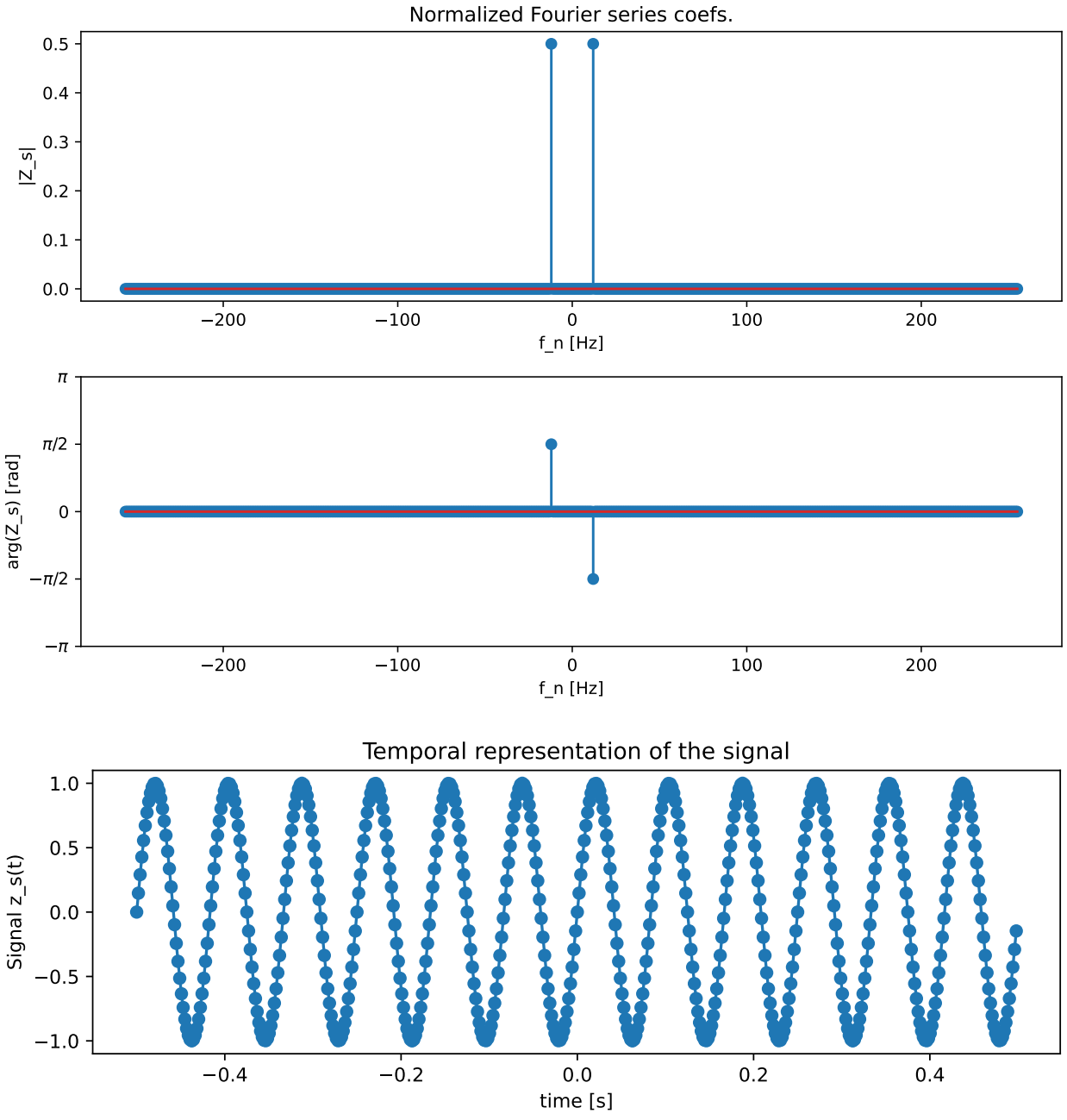
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APPLICATION                                                           %
% Fourier series coefficients of a periodic, non-causal signal          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Initializations
% Regular initializations lines in MATLAB
clc
clear all
close all

% Parameters
N  = 512;  % number of samples
Tw = 1.;   % windowing duration in natural units, i.e. seconds

f1= 12;   % frequency of the cosine waveform signal in Hz
```

```
plot(t_s, z_s, 'o-'), xlabel('time [s]'), ylabel('Signal z_s(t)'),...
    title('Temporal representation of the signal')
```

Python / NumPy version:

```
import numpy as np
import matplotlib.pyplot as plt

def phase_with_threshold(Z, threshold=1e-8):
    phi = np.angle(Z) * (np.abs(Z) > threshold)
    return phi * (np.abs(phi) > threshold)

# Parameters
N = 512
Tw = 1.0
f1 = 12.0

# Built-in variables
Ts = Tw / N
fs = 1 / Ts
df = fs / N

# Support vectors
t_s = np.arange(-Tw/2, Tw/2, Ts)
f_n = np.arange(-N//2, N//2) * df

# Building the test discrete-time sampled signal
z_s = np.sin(2 * np.pi * f1 * t_s)

# DFT calculation
Z_s_FSC = np.fft.fftshift(np.fft.fft(z_s) / N)
Phi_Z_s_FSC = phase_with_threshold(Z_s_FSC)

# Displays
fig, ax = plt.subplots(2, 1, figsize=(9, 6))
ax[0].stem(f_n, np.abs(Z_s_FSC))
ax[0].set(xlabel="f_n [Hz]", ylabel="|Z_s|",
          title="Normalized Fourier series coefs.")
ax[1].stem(f_n, Phi_Z_s_FSC)
ax[1].set(xlabel="f_n [Hz]", ylabel="arg(Z_s) [rad]")
ax[1].set_ylim(-np.pi, np.pi)
ax[1].set_yticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
ax[1].set_yticklabels([r"$-\pi$", r"$-\pi/2$", "0", r"$\pi/2$", r"$\pi$"])
fig.tight_layout()

plt.figure(figsize=(8, 3))
plt.plot(t_s, z_s, "o-")
```

```
plt.xlabel("time [s]")
plt.ylabel("Signal z_s(t)")
plt.title("Temporal representation of the signal")
plt.tight_layout()
plt.show()
```



Conclusion:

The spectrum is consistent with that of the causal signal, as expected from the periodicity of the signal.

- **Signal 2:** Let us now look at the situation where, instead of a 1 s-windowing, the windowing is now defined as $T_w = T_1 = 1/f_1$, symmetrically around 0. In other words:

$$z(t) = \sin(2\pi f_1 t), \forall t \in \left[-\frac{T_1}{2}; \frac{T_1}{2}\right] \tag{49}$$

7.1.1.2 Illustrative code

The code below illustrates that.

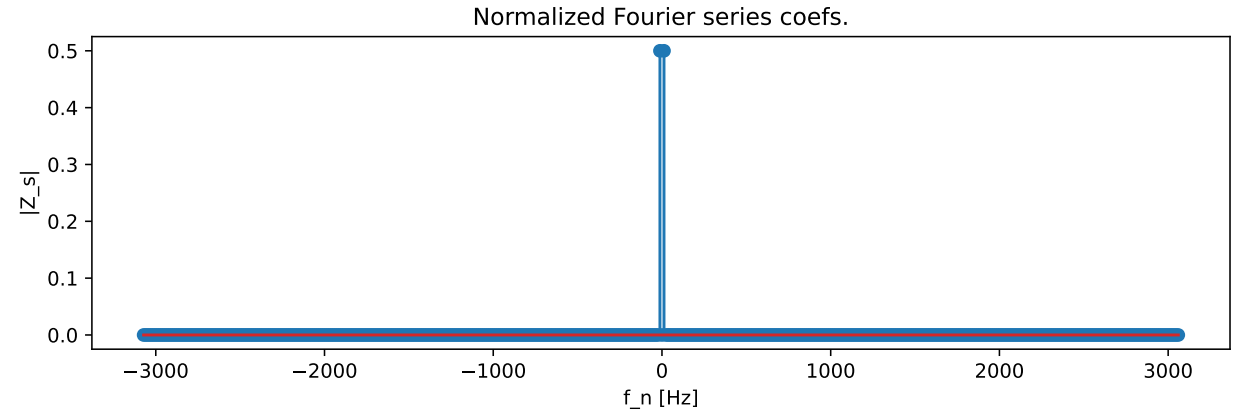
MATLAB version:

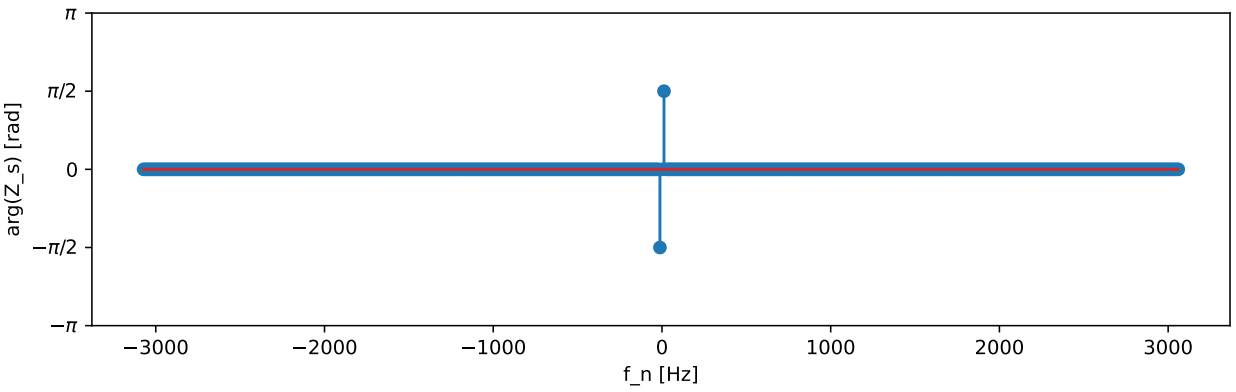
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APPLICATION                                                            %
% Fourier series coefficients of a periodic, non-causal signal           %
% (continued)                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all


% Parameters
N  = 512;  % number of samples

f1= 12.;    % frequency of the cosine waveform signal in Hz
Tw = 1/f1; % windowing duration now defined over one period only

%In-built variables
Ts = Tw/N; % sampling period
fs = 1/Ts; % sampling frequency
df = fs/N; % spectral resolution

% support vectors
t_s= -Tw/2:Ts:Tw/2-Ts; % time support, of duration Tw
f_n= -fs/2:df:fs/2-df; % two-sided frequency support


%%%%% Building the test discrete-time sampled signal
z_s = sin(2*pi*f1*t_s); % a simple sine waveform signal, of frequency f1


%%%%% DFT calculation
% 2-fft function
Z_s_FSC = fftshift(fft(z_s)/N);  % fft-based normalized FSC to be
% represented over a two-sided spectrum

phases_Z_s_FSC    = angle(Z_s_FSC);
```

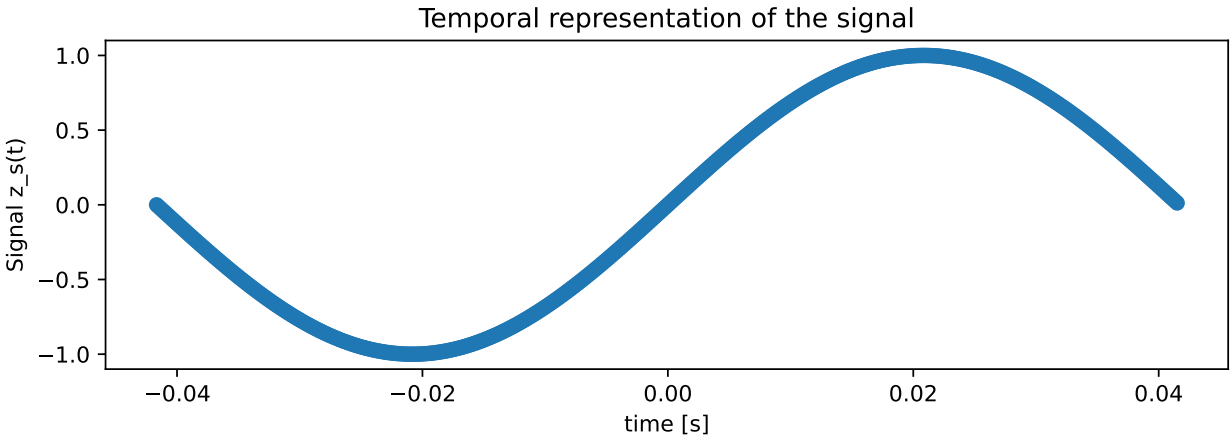
```
phases_Z_s_FSC_cl = clean_phases(Z_s_FSC);


%% Function clean_phases
function phases = clean_phases(coefficients, rtol)

if nargin < 2
    rtol = 1e-12;
end

magnitudes = abs(coefficients);
phases = angle(coefficients);

threshold = rtol * max(magnitudes);
phases(magnitudes <= threshold) = 0;

end

%%%%% Displays
figure
subplot(2,1,1), stem(f_n, abs(Z_s_FSC)), xlabel('f_n [Hz]'),...
    ylabel('|Z_s|'), axis([-15, 15,-0.1, 0.5]), title('Zoom in')
subplot(2,1,2), stem(f_n, phases_Z_s_FSC_cl),  xlabel('f_n [Hz]'),...
    ylabel('arg(Z_s) [rad]'), axis([-15, 15, -pi, pi])

figure
subplot(2,1,1), stem(f_n, abs(Z_s_FSC)), xlabel('f_n [Hz]'),...
    ylabel('|Z_s|'), title('Normalized Fourier series coefs.')
subplot(2,1,2), stem(f_n, phases_Z_s_FSC_cl),  xlabel('f_n [Hz]'),...
    ylabel('arg(Z_s) [rad]')

figure
plot(t_s, z_s, 'o-'), xlabel('time [s]'), ylabel('Signal z_s(t)'),...
    title('Temporal representation of the signal')
```

Python / NumPy version:

```
import numpy as np
import matplotlib.pyplot as plt

def phase_with_threshold(Z, threshold=1e-8):
    phi = np.angle(Z) * (np.abs(Z) > threshold)
    return phi * (np.abs(phi) > threshold)

# Parameters
N = 512
f1 = 12.0
```







Conclusion:

We have added a zoom in the $[-15; +15]$ *Hz range, where only three samples are visible due to the frequency resolution of the problem* $\delta f = 1/T_w = 1/T_1 = f_1 = 12$ *Hz. The peak magnitudes at* $\pm f_1$ *are* 0.5 *as expected, but the phase peaks are surprisingly reversed compared to the former calculation of the same signal. Owing to its periodicity, the spectrum should be identical, though.*

Questions & answers:

So, what is the problem?

Answer:

The DFT makes no assumption about the causal or non-causal character of the sampled signal $z_s(t_s)$. It is the user who decides whether the signal is to be plotted against a causal, or a non-causal, time interval. The DFT does not consider the actual time support $\{nT_s\}$ but only the (time) index n, which always begins at $n = 0$ (Python) or $n = 1$ (MATLAB). Therefore, **the DFT inherently assumes that the signal $z_s(t_{sw})$ is causal**, *i.e.* that the first half of the input signal ($\frac{N}{2}$ samples) correspond to positive times starting from $t = 0$, while its second half (/2 samples) is interpreted as representing negative times.

In the example above, the DFT algorithm therefore interprets the signal as starting from a "false reference time" $t_{frt} = 0$ (the signal "goes down") and lasting a duration $T_w = T_1$. In this case, doing so, the operation falsifies the phase of the signal at the "true reference time" $t_{trt} = 0$, as defined by the user for the temporal representation of the signal (the signal "goes up"). If the signal is interpreted as starting from $t_{frt} = 0$, it is immediately seen that the corresponding expression is $z(t) = -\sin(2\pi f_1 t)$, not $z(t) = +\sin(2\pi f_1 t)$ as initially assumed. Hence the inversion of the phase peaks in the spectrum by a factor of π.

But, why did this effect not occur when considering the initial time interval $t \in [-1/2; +1/2]$?

Answer:

Looking at that signal, it is readily seen that, over that windowing, the phases of the signal at $t_{frt} = 0$ and $t_{trt} = 0$ are similar. Hence, the consistent calculation of the phase of the DFT.

Is there a way to treat non-causal signals without having to care about the way the windowing is performed?

Answer:

Yes, if we know the signal is truly non-causal and that the way its DFT phase is calculated matters, we can force the non-causal character of the signal to be taken into account, as demonstrated below.

7.1.2 Forcing the non-causality to be considered in the DFT spectrum

The first half of the signal $z_{sw}(t_s)$ (first $N/2$ samples) which represents the negative part of the time support is placed at the end of the signal vector. By doing so, the "false reference time" of the DFT is forced to match the "true reference time" of the signal, and the DFT can be computed consistently.

This is achieved by using the *"fftshift"* function again, but now applied to the signal $z_{sw}(t_s)$ itself, before computing the DFT. The figure and the code below illustrate this.

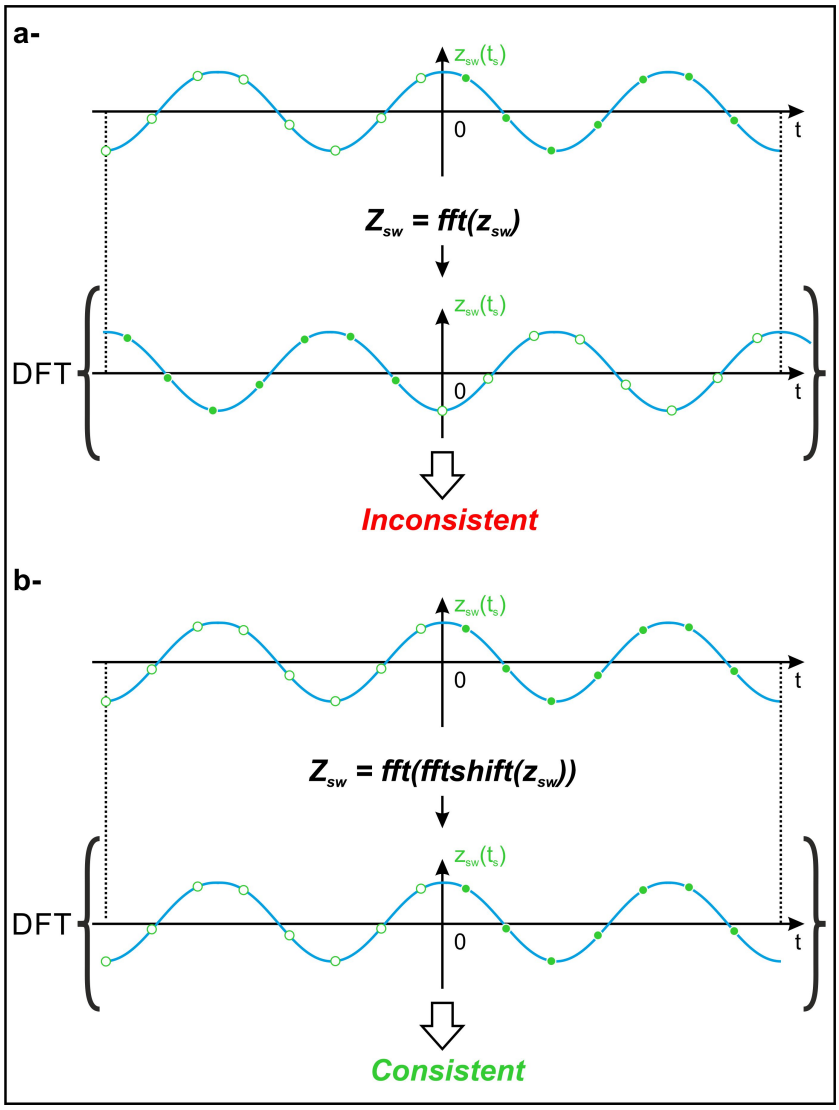


Figure 9: About the use of fftshift to account for the non-causality of a signal.

7.1.2.1 Illustrative code

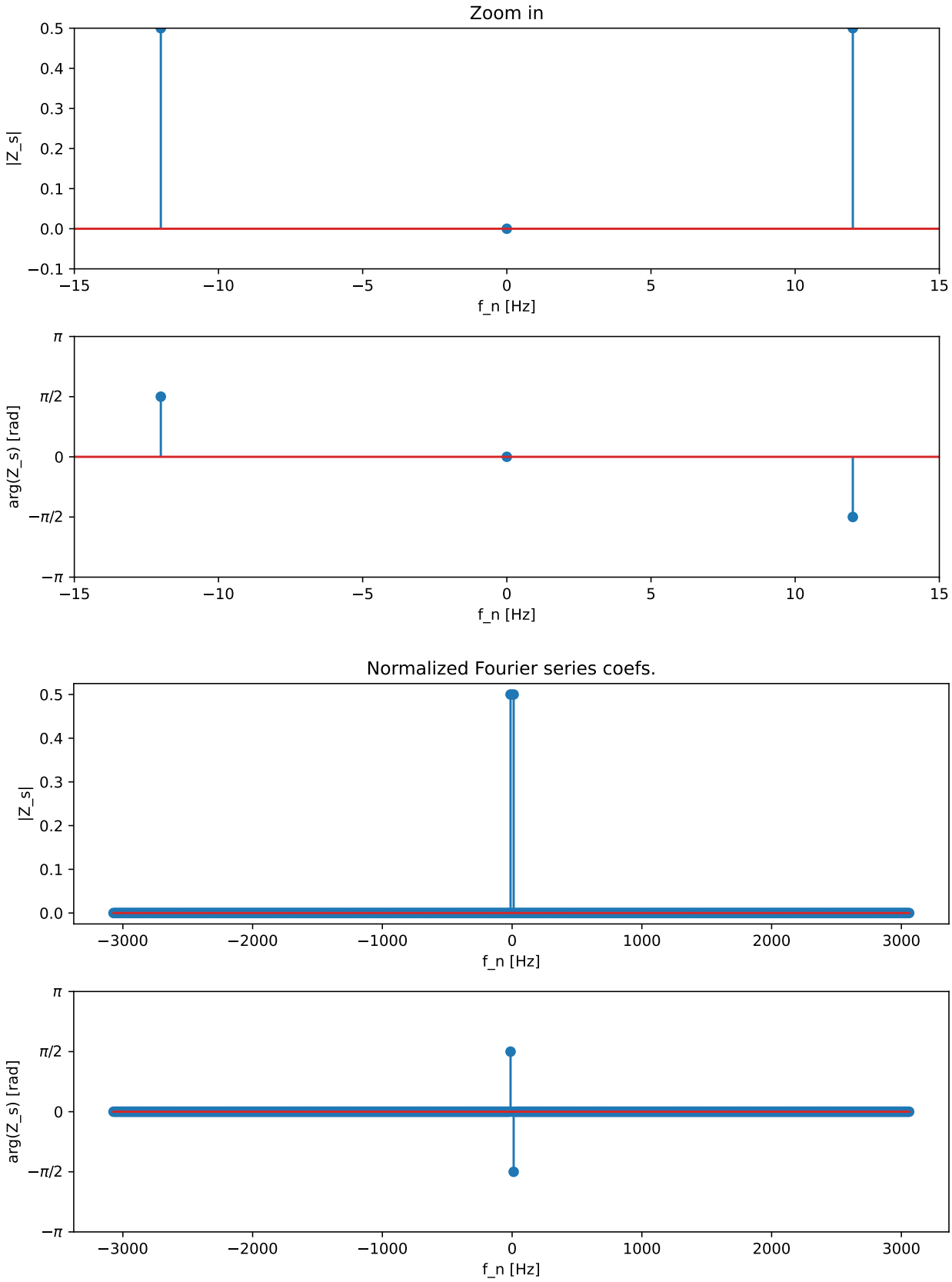
MATLAB version:

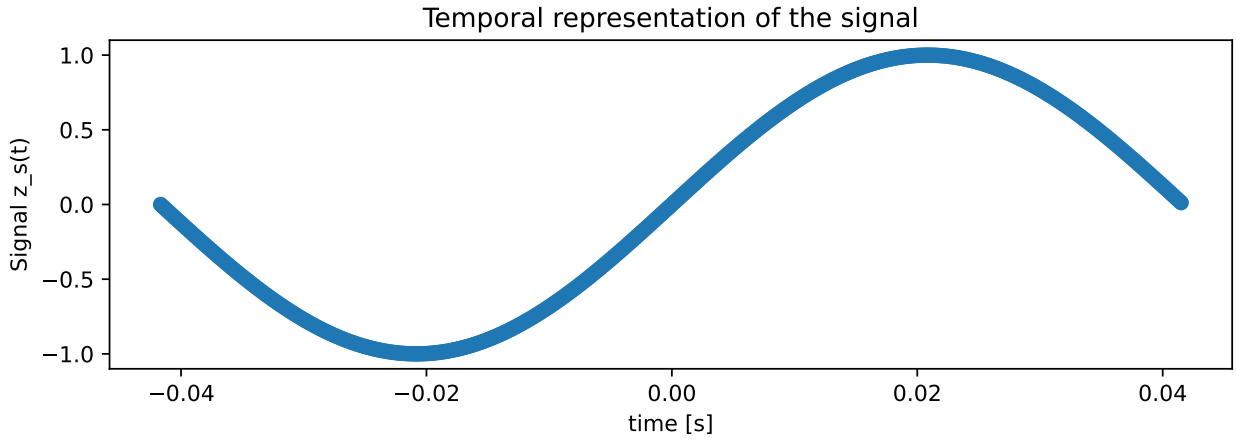
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APPLICATION                                                           %
% Fourier series coefficients of a periodic, non-causal signal          %
% Forcing the non-causality to be considered                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all

% Parameters
```

```
%%%% Displays
figure
subplot(2,1,1), stem(f_n, abs(Z_s_FSC)), xlabel('f_n [Hz]'),...
    ylabel('|Z_s|'), axis([-15, 15,-0.1, 0.5]), title('Zoom in')
subplot(2,1,2), stem(f_n, phases_Z_s_FSC_cl),  xlabel('f_n [Hz]'),...
    ylabel('arg(Z_s) [rad]'), axis([-15, 15, -pi, pi])

figure
subplot(2,1,1), stem(f_n, abs(Z_s_FSC)), xlabel('f_n [Hz]'),...
    ylabel('|Z_s|'), title('Normalized Fourier series coefs.')
subplot(2,1,2), stem(f_n, phases_Z_s_FSC_cl),  xlabel('f_n [Hz]'),...
    ylabel('arg(Z_s) [rad]')

figure
plot(t_s, z_s, 'o-'), xlabel('time [s]'), ylabel('Signal z_s(t)'),...
    title('Temporal representation of the signal')
```

Python / NumPy version:

```
import numpy as np
import matplotlib.pyplot as plt

def phase_with_threshold(Z, threshold=1e-8):
    phi = np.angle(Z) * (np.abs(Z) > threshold)
    return phi * (np.abs(phi) > threshold)

# Parameters
N = 512
f1 = 12.0
Tw = 1 / f1  # windowing duration defined over one period only

# Built-in variables
Ts = Tw / N
fs = 1 / Ts
df = fs / N

# Support vectors
t_s = np.arange(-Tw/2, Tw/2, Ts)
f_n = np.arange(-N//2, N//2) * df

# Building the test discrete-time sampled signal
z_s = np.sin(2 * np.pi * f1 * t_s)

# DFT calculation
Z_s_FSC = np.fft.fftshift(np.fft.fft(np.fft.fftshift(z_s)) / N)
Phi_Z_s_FSC = phase_with_threshold(Z_s_FSC)
```


Zoom in
|Z_s|
f_n [Hz]
arg(Z_s) [rad]
f_n [Hz]
Normalized Fourier series coefs.
|Z_s|
f_n [Hz]
arg(Z_s) [rad]
f_n [Hz]



Conclusion:

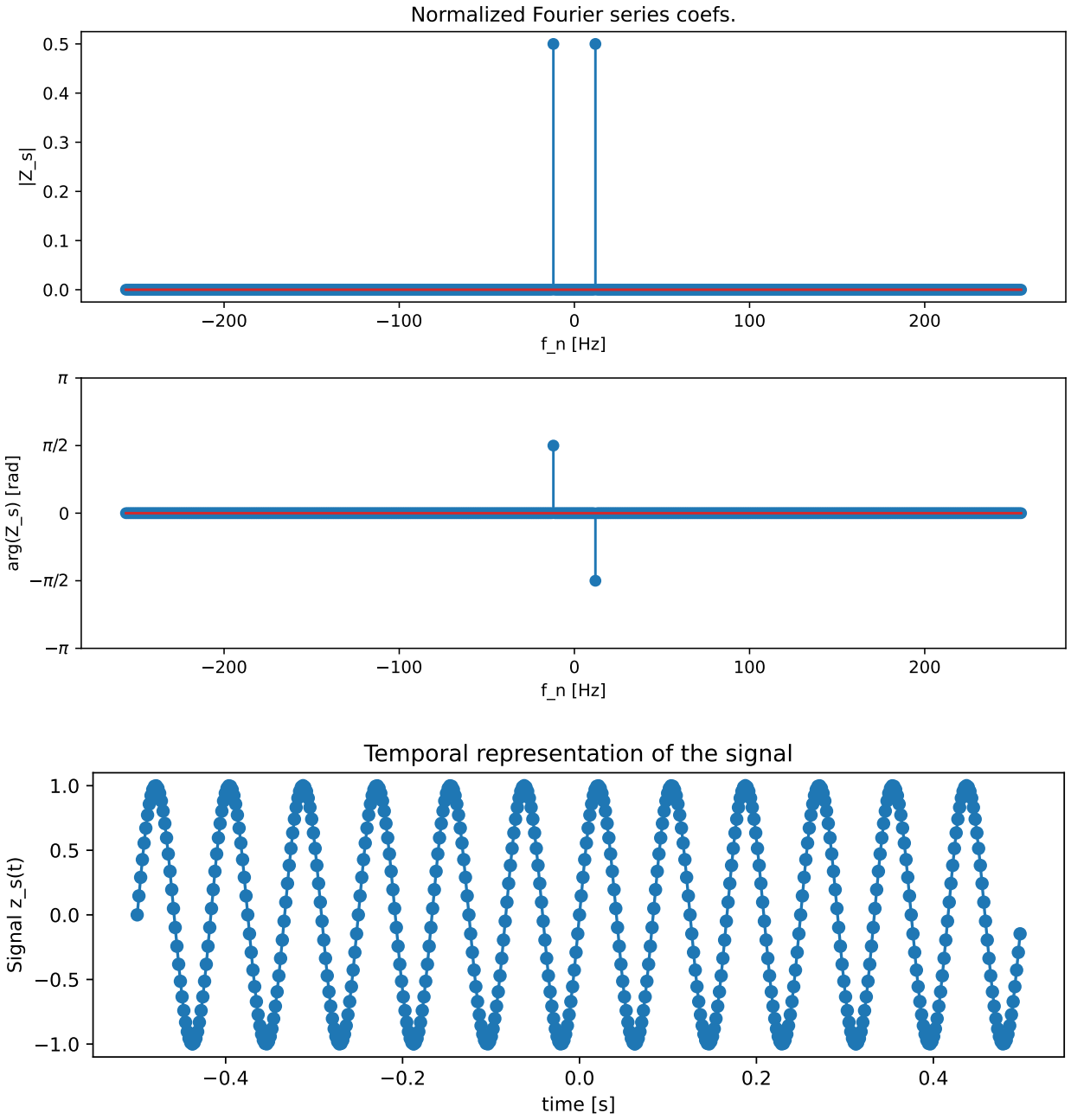
The phase is now represented consistently.

Naturally, applying *"fftshift"* to a well-windowed non-causal signal, as initially defined (see Eq. (43)), does not modify its DFT, as shown in the code below.

7.1.2.2 Illustrative code

MATLAB version:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APPLICATION                                                                  %
% Fourier series coefficients of a periodic, non-causal signal                 %
% Forcing the non-causality to be considered                                   %
% (continued)                                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all

% Parameters
N  = 512;  % number of samples
Tw = 1;   % windowing duration in natural units, i.e. seconds

f1= 12;   % frequency of the cosine waveform signal in Hz

%In-built variables
Ts = Tw/N; % sampling period
fs = 1/Ts; % sampling frequency
df = fs/N; % spectral resolution

% support vectors
t_s= -Tw/2:Ts:Tw/2-Ts; % time support, of duration Tw
f_n= -fs/2:df:fs/2-df; % two-sided frequency support
```



7.2 DFT of a non-periodic signal

We now consider the more generic case of non-periodic signals, hence the use of the FT formalism.

7.2.1 Causal signal

- **Signal 3:** For this application, we consider a causal, time-limited, non-periodic signal, where non-periodicity is caused by random noise. The signal is built as follows. We consider the realistic case of a useful electrical signal perturbed by ambient electrical noise. The useful

signal is assumed to be sinusoidal, with frequency 17 Hz. The noise has two components: a sinusoidal oscillation, with frequency and amplitude arbitrarily set to 50 Hz and 0.1 V, respectively, and a uniform random background component (white noise) whose magnitude is equal to that of the 50 Hz oscillation. The amplitude of the useful signal is five times smaller than that of the 50 Hz oscillation. Thus, the useful signal is "buried in noise".

The signal is windowed over $T_w = 1$ s. The code below shows how powerful the DFT is for detecting the spectral components of a noisy signal.

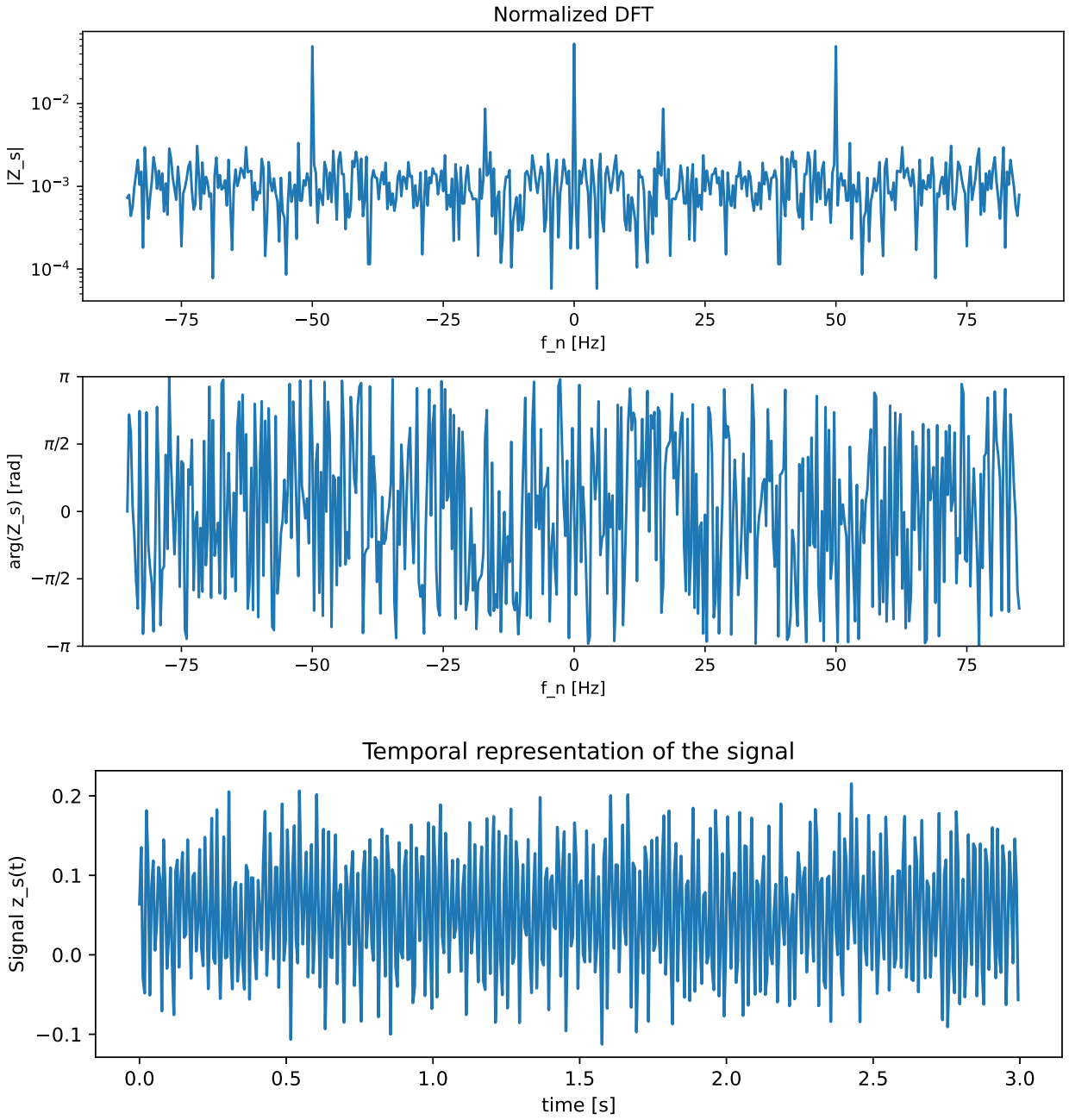
7.2.1.1 Illustrative code

MATLAB version:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APPLICATION                                                                  %
% DFT of a non-periodic signal                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Initializations
% Regular initializations lines in MATLAB
clc
clear all
close all

% Parameters
N  = 512;  % number of samples
Tw = 3;    % windowing duration now defined over one period only

f1 = 50;    % frequency of the 1st signal component in Hz
f2 = 17;    % frequency of the 2nd signal component in Hz
mag_s1 = 0.1;      % magnitude of the 1st signal component, in V
mag_s2 = mag_s1/5; % magnitude of the 2nd signal component, in V

%In-built variables
Ts = Tw/N; % sampling period
fs = 1/Ts; % sampling frequency
df = fs/N; % spectral resolution

noise=rand(1,N)*mag_s1;

% support vectors
n   = 0:N-1;
t_s = n*Ts; % time support, of duration Tw
f_n = -fs/2:df:fs/2-df; % two-sided frequency support
tau = Tw/2;

%%%%% Building the test discrete-time sampled signal
z_s = mag_s2*sin(2*pi*f2*t_s)+mag_s1*sin(2*pi*f1*t_s)+noise; % noisy signal
```

```
# Parameters
N = 512
Tw = 3.0

f1 = 50.0
f2 = 17.0
mag_s1 = 0.1
mag_s2 = mag_s1 / 5

# Built-in variables
Ts = Tw / N
fs = 1 / Ts
df = fs / N

rng = np.random.default_rng(seed=0)
noise = rng.random(N) * mag_s1

# Support vectors
n = np.arange(N)
t_s = n * Ts
f_n = np.arange(-N//2, N//2) * df
tau = Tw / 2

# Building the test discrete-time sampled signal
z_s = (mag_s2 * np.sin(2 * np.pi * f2 * t_s)
       + mag_s1 * np.sin(2 * np.pi * f1 * t_s) + noise)

# DFT calculation
Z_s_DFT = np.fft.fftshift(np.fft.fft(z_s)) / N

Phi_Z_s_DFT = phase_with_threshold(Z_s_DFT)

# Displays
fig, ax = plt.subplots(2, 1, figsize=(9, 6))
ax[0].semilogy(f_n, np.abs(Z_s_DFT))
ax[0].set(xlabel="f_n [Hz]", ylabel="|Z_s|",
          title="Normalized DFT")
ax[1].plot(f_n, Phi_Z_s_DFT)
ax[1].set(xlabel="f_n [Hz]", ylabel="arg(Z_s) [rad]")
ax[1].set_ylim(-np.pi, np.pi)
ax[1].set_yticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
ax[1].set_yticklabels([r"$-\pi$", r"$-\pi/2$", "0", r"$\pi/2$", r"$\pi$"])
fig.tight_layout()

plt.figure(figsize=(8, 3))
plt.plot(t_s, z_s)
plt.xlabel("time [s]")
```

```
plt.ylabel("Signal z_s(t)")
plt.title("Temporal representation of the signal")
plt.tight_layout()
plt.show()
```



Conclusion:

In the two-sided spectrum of the modulus of the DFT, the 50 Hz components of the noise are clearly visible, and the 17 Hz components of the actual signal are visible too, whereas

9 Conclusion and Key Takeaways

This course monograph has introduced the **mathematical foundations**, **practical implementation**, and **interpretation of the Discrete Fourier Transform (DFT)** in the context of signal processing.

The progressive derivation from the continuous-time Fourier transform to the discrete Fourier transform has emphasized the successive effects of:

- time sampling,
- finite-duration observation,
- spectral discretization,
- and numerical implementation.

Particular attention has been paid to the **practical interpretation of DFT spectra**, including:

- spectral resolution,
- frequency-axis construction,
- amplitude normalization,
- one-sided and two-sided spectra,
- phase issues
- spectral leakage,
- windowing effects,
- and the role of `fftshift`.

A **major objective** of this notebook has been to bridge the gap between:

- the mathematical formalism of Fourier analysis,
- and practical numerical implementations using Python/NumPy.

Throughout the document, the **Fourier-series formalism** and the **Fourier-transform formalism** have been systematically distinguished in order to clarify:

- the physical meaning of the computed spectra,
- the interpretation of amplitudes and frequency supports,
- and the correct normalization conventions.

The **implementation-oriented** sections have also highlighted **several practical issues** frequently encountered by students and practitioners, such as:

- causal versus non-causal signal representations,
- frequency centering,
- interpretation of negative frequencies,
- and the construction of physically meaningful spectral representations.

This notebook is **intentionally designed** as both:

- an **introductory pedagogical resource** for students discovering Fourier analysis,
- and a **practical reference document** for signal-processing implementations.

The authors hope that this material will help readers develop a deeper intuition regarding the DFT and its central role in modern signal processing.