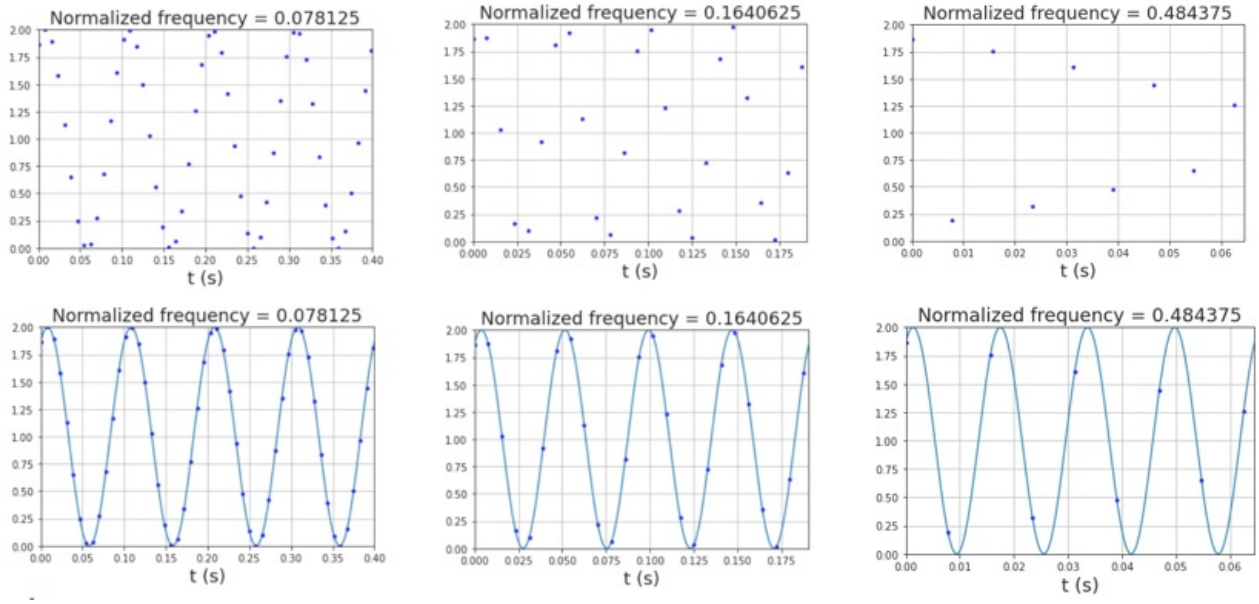
Kernel: Python 3 (system-wide)

Lab session 2A : Discrete-time sinusoids

Introduction

Digital one-dimensional signals most often result from the sampling of an analog signal x(t) using an Analog-to-Digital converter, or from the analytical expression of a continuous function x(t). In both cases, the tranformation from continuous time t to discrete time realizes $x[n] = x(nT_s)$ with a **sampling frequency** $f_s = \frac{1}{T_s}$, which is generally constant for a given signal.

While analog signals take any real values at any time on their domains, digital signals (see Tervo, p.58) are both of discrete time and of discrete values. This double discretization is mandatory if we want to handle signals using computers, which obviously have finite memory amount. In these lab sessions, we will often **use digital signals to represent analog signals**. As an example, the figures below show a discrete-time sinusoid $x[n] = x(nT_s) = Acos(2\pi f_0 nT_s)$ and the corresponding continuous-time signal $x(t) = Acos(2\pi f_0 t)$ it should represent :



Although this association seems natural and obvious in this example, weirder behaviors are frequently obtained which can't be explained without better knowledge of the time sampling process.

In particular :

1. A discrete sinusoid, usually considered as $x[n] = A\cos(2\pi f_0 nT_s)$ does not always look like a continuous sinusoid (see the upper right graph above), although they are totally legitimate to represent an actual analog sinusoid.
2. The same suite of samples extracted from an analog sinusoid may be obtained for several values of f_0, which will accordingly rather be considered as a "frequency parameter". In other words, unlike the analog "usual" sinusoid, there is no one-to-one correspondence between the frequency parameter f_0 and a given discrete sinusoid, except if the frequency parameter f_0 is constrained in a particular interval (*e.g.* $[0; f_s]$ or $[-\frac{f_s}{2}; \frac{f_s}{2}]$).
3. It follows from the previous point that discrete sinusoids "extracted" from an analog sinusoidal signal $x(t)$ do not always show the same "apparent frequency" that the natural frequency f_0 of $x(t)$. For adequately sampled sinusoids (*i.e.* for sufficiently fast sampling) this apparent frequency is bound to a maximum value of $f_0 = \frac{f_s}{2}$. Any sampling made at $f_s < 2f_0$ will result in an apparent frequency lower than f_0 ; these sinusoids are called "undersampled".
4. Discrete sinusoids "extracted" from an analog sinusoidal signal should be sampled with a sampling period which is at least two times smaller than the natural period $T_0 = \frac{1}{f_0}$ of the analog sinusoid as $T_0 = \frac{2}{f_s}$ because $f_0 = \frac{f_s}{2}$ and as we know $T_s = \frac{1}{f_s}$ that means $T_0 = 2T_s, T_s = \frac{T_0}{2}$

Point 1 raises a question : How do we know that a discrete sinusoid, even if looking weird, is actually "legitimate" to represent an analog sinusoid of frequency f_0. By this, we mean that if the discrete sinusoid is converted back to a continuous signal, we obtain the right analog sinusoid with the same frequency f_0. In other words (see The sampling theorem in *dspguide.com*) :

If you can exactly *reconstruct* the analog signal from the samples, you must have done the sampling *properly*. Even if the sampled data appears confusing or incomplete, the key information has been captured if you can reverse the process.

This defines the proper sampling of an analog sinusoidal signal, formalized by the famous ***sampling theorem*** or ***Shannon-Nyquist theorem***, a milestone in the field of digital signal processing (DSP). To grasp the full significance of this theorem requires visualizing the sampling process in the frequency domain. In this lab session, you will discover a part of the meaning and practical consequences of this theorem in an empirical but guided way.

At any moment before, during and after this lab session, you should read carefully the above-cited page devoted to the sampling theorem in the *dspguide.com*. At the end of this lab, you should understand more clearly the developments made in this page.

Lab objectives

This lab is made to familiarize yourself with discrete sinusoids as a particular object of signal processing ; take it as a guided exploration, feel free to experiment with parameters,... Contrary to other labs, this one will not require any programmation ; hence, the code cells are executed in Python 3. The main reason for using Python is the easy integration of widgets in Jupyter notebooks.

Parameters initialization

You had first to execute (`Shift-Enter`) the code cell below to initialize the environment and load the required Python libraries.

In [3]:
```
import numpy as np
from matplotlib import pylab as plt
import wave
import IPython
from scipy import fftpack as fft
import ipywidgets as widgets
from ipywidgets import interact, interactive, fixed, interact_manual
from math import floor
from scipy import signal as sig
%matplotlib inline

# Casting unitary numbers to real numbers will give errors
# because of numerical rounding errors. We therefore disable
# warning messages.
import warnings
# warnings.filterwarnings('ignore')

# Set common figure parameters
newparams = {'axes.labelsize': 12, 'axes.linewidth': 1, 'savefig.dpi': 200,
             'lines.linewidth': 1, 'figure.figsize': (6, 6),
             'ytick.labelsize': 7, 'xtick.labelsize': 10,
             'ytick.major.pad': 5, 'xtick.major.pad': 5,
             'legend.fontsize': 10, 'legend.frameon': True,
             'legend.handlelength': 1.5, 'axes.titlesize': 12,}
plt.rcParams.update(newparams)

def tone(frequency=440., length=1., amplitude=1., sampleRate=44100., soundType='int8'):
    """ Returns a sine function representing a tune with a given frequency.

    :frequency: float/int. Frequency of the tone.
    :length: float/int. Length of the tone in seconds.
    :amplitude: float/int. Amplitude of the tone.
    :sampleRate: float/int. Sampling frequency.
    :soundType: string. Type of the elements in the returned array.
    :returns: float numpy array. Sine function representing the tone.
    """
    t = np.linspace(0,length,floor(length*sampleRate))
    data = amplitude*np.sin(2*np.pi*frequency*t)
    return data.astype(soundType)

# Parameters that are being used in the start of this notebook
sampleRate = 44100     # La fréquence d'échantillonnage, celle d'un CD audio ordinaire
sampwidth = 1          # In bytes. 1 for 8 bit, 2 for 16 bit and 4 for 32 bit
volumePercent = 50     # Volume percentage
nchannels = 1          # Mono. Only mono works for this notebook

# Some dependent variables
shift = 128 if sampwidth == 1 else 0 # The shift of the 8 bit samples, as explained in the section
above.
soundType = 'i' + str(sampwidth)
amplitude = np.iinfo(soundType).min*volumePercent/100.
```

Discrete-time sinusoids

The **discrete frequency of a discrete-time sinusoid** is merely its natural frequency normalized by the sampling frequency, *i.e.* :

$$\nu = \frac{f_0}{f_s}$$

The normalized frequency is sometimes also called reduced frequency or normalized frequency.

Owing to fact that $f_s = \frac{1}{T_s}$, any discrete-time sinusoid may thus be represented in the simple following way, depending only on the indices n, without reference to the actual sampling frequency T_s :

$$x[n] = A\cos(2\pi f_0 n T_s + \phi) = A\cos(2\pi \frac{f_0}{f_s} n + \phi) = A\cos(2\pi \nu n + \phi)$$

Discrete-time sinusoids in this particular form are important objects in digital signal processing (DSP).

The inverse of the discrete frequency gives N, the number of samples per period of the discrete sinusoid. It requires at least two samples per period to reproduce at least the right frequency of the sampled analog sinusoid ;in this case however, depending on the phase, the amplitude of the sampled sinusoid may be smaller than the actual A, and may even be zero.

$$x[n] = A\cos(2\pi \nu n + \phi) = A\cos(2\pi \frac{n}{N} + \phi)$$

The sinusoid sampling simulator

The code cell below shows a section of an analog sinusoid sampled at a **fixed sampling frequency** $f_s = 20Sa/s$. You can choose its fundamental frequency f_0 (between $0Hz$ and $200Hz$) using the upper cursor, with steps of 0.25 Hz. Note that you can easily change the excursion frequency parameters in the relevant *frequency slider* line in the code cell below. Another slider determines the phase at origin ϕ_0, which can vary between 0 and π with steps of $\frac{\pi}{10}$.

You should first visualize various discrete sinusoids by slowly increasing (through the upper slider) the fundamental frequency of the analog sinusoid used for sampling. You will notice that above some frequency, it is not always easy to recognize the shape of a sinusoid. You have the possibility to visualize the analog sinusoid from which the samples of the discrete signal are extracted : for this, you just have to check the `Show continuous sine guide` checkbox.

The discrete frequency parameter ν_0 is also printed just above the graph. When the actual value of ν_0 lies outside its irreducible interval (*i.e.* $\nu_0 > 0.5$), the Shannon-Nyquist condition is not fullfilled, and the analog sinusoid is undersampled ; in this case, the background color appears in yellow.

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from math import floor

# Set common figure parameters
newparams = {'axes.labelsize': 12, 'axes.linewidth': 1, 'savefig.dpi': 200,
             'lines.linewidth': 1, 'figure.figsize': (10, 5),
             'ytick.labelsize': 7, 'xtick.labelsize': 7,
             'ytick.major.pad': 5, 'xtick.major.pad': 5,
             'legend.fontsize': 7, 'legend.frameon': True,
             'legend.handlelength': 1.5, 'axes.titlesize': 7,}
plt.rcParams.update(newparams)

# Constants
A = 1
f0 = 20
te = 0.05
fe = 1 / te
T0 = 1
NP = floor(T0 / te)

tt = np.arange(0, T0, 1 / (25 * fe))

# Function to generate and plot the signal
def sam(f0, cont, ind, phi):  # Added phi as a parameter
    global s
    nu = f0 / fe
    NP = floor(T0 * fe)
    t = np.arange(0, T0 + 1/fe, 1/fe)
    s = A * np.cos(2 * np.pi * f0 * t + phi)  # Use phi in the cosine function
    ss = A * np.cos(2 * np.pi * f0 * tt + phi)

    ax = plt.axes()
    plt.plot(t, s, 'o')
    plt.xlabel('time (s)')

    if cont == 1:
        plt.plot(tt, ss)

    plt.grid()
    plt.ylim(-1.05, 1.05)

    print('Sampling frequency =', fe)
    print("First elements of the discrete sine : ", s[0:10])

    if fe / 2 < f0:
        print('Aliasing !')
        ax.set_facecolor("yellow")
        k = floor(f0 / fe + 0.5)
```

```
        fapp = abs(k * fe - f0)
        ss1 = A * np.cos(2 * np.pi * (fapp) * tt + phi)
        if ind == 1:
            plt.plot(tt, ss1)
        print('Folded (Apparent) frequency =', fapp)
        print('Discrete frequency =', nu)
    else:
        print(' ')
        print('Discrete frequency nu =', nu)
        ax.set_facecolor("white")

# Frequency slider
f0w = widgets.FloatSlider(min=1, max=60, step=0.25, value=1, description="Frequency (Hz)")

# Checkbox for continuous sine guide
f1w = widgets.Checkbox(
    value=False,
    description='Show continuous sine guide',
    disabled=False,
    indent=False
)

# Checkbox for aliasing
f2w = widgets.Checkbox(
    value=False,
    description='Show principal alias',
    disabled=False,
    indent=False
)

# **New Phase Slider**
phi_w = widgets.FloatSlider(min=0, max=np.pi, step=np.pi/10, value=0, description="Phase φ")

# Interactive widget with phase slider
widgets.interact(sam, f0=f0w, cont=f1w, ind=f2w, phi=phi_w)
```

Out[5]: <function __main__.sam(f0, cont, ind, phi)>

I. Sampling an analog sinusoid

Use the simulator above to visualize a section of a discrete sinusoid, obtained from sampling an analog sinusoid of variable frequency f_0 at a fixed sample frequency $f_s = 20Sa/s$.

Q1: What is the maximum frequency f_0^{Max} of the analog sinusoid that you can safely sample in order to get a representative set of samples ?

Q1-R: It should be according to the condition to the Shannon-Nyquist theorem as follow

$f_0^{Max} = \frac{fs}{2} = \frac{20}{2} = 10.00Hz$

Q2: To which discrete frequency ν_{Max} corresponds f_0^{Max} ?

Q2-R: As we know that the discrete frequency $\nu_{Max} = \frac{f_0^{Max}}{fs} = \frac{10.00Hz}{20.00Hz} = 0.5$

I.1 Sampling close to the limit : $f_0 \leq \frac{f_s}{2}$

We will first limit our exploration to analog frequencies smaller to $\frac{f_s}{2}$, corresponding to a correct sampling, with a particular focus on the region close to the Shannon-Nyquist limit. Remember that the background color will shift from white to yellow if the Shannon-Nyquist condition is not fulfilled, corresponding to under sampling the analog sinusoid.

Q3: In which frequency range of discrete frequency ν can you easily recognize the lineshape of a sinusoid considering only the sample (*i.e* without using the "*Show continuous sine guide*" checkbox) ?

Q3-R: upto $\nu = 0.25$

Note : You can increase this frequency range using the "*Show continuous sine guide*" checkbox. Note that the (pseudo)analog sinusoid which appears in orange continuous line is actually also a discrete-time sinusoid, albeit with a sampling frequency large enough so that it appears continuous when traced with the instruction like *e.g.* `plt.plot(tt,ss)` which links adjacent samples with a straight line.

Q4: Place the upper cursor on $f_0 = 10Hz$, and report below the values of the first ten samples.

Q4-R: The values of first ten samples are following

[1. -1. 1. -1. 1. -1. 1. -1. 1. -1.]

Keeping $f_0 = f_{Nyquist} = \frac{f_s}{2} = 10Hz$, change the value of the phase at the origin ϕ_0 of the analog sinusoid, and take a look on the variations of the amplitude.

Q5: Search for the conditions to obtain a series of equal sample values : Give this constant value and the actual phase ϕ_0 at which it is obtained.

Q5-R: At $\phi_0 = \frac{\pi}{2}$ we have all the values of the function close to zero. The the constant value is zero, while the phase value is $\phi_0 = \frac{\pi}{2}$. It is also evident mathematically that $x[n] = cos(2\pi\frac{f_0}{f_s} + \phi_0) = cos(2\pi\frac{10.00}{20.00} + \phi_0) = cos(n\pi + \phi_0)$ for $\phi_0 = \frac{\pi}{2}$ we have $cos(2\pi + \frac{\pi}{2}) = sin(n\pi) = 0$, thus all constant values = 0.

A lesson to remember : Working just at the Shannon-Nyquist limit may lead to a false appreciation of the amplitude of the sampled sinusoid, which falsely appears smaller than its actual value. For a particular phase, it may even appear as a constant value for all the samples. It is thus safer to work well above the shannon-Nyquist limit, by taking the sampling frequency much higher than the Nyquist frequency (*i.e.* much higher than twice the "continuous" frequency f_0)

I.2 Multiple Aliases : $f_0 > \frac{f_s}{2}$

For the remainder of this exercise, you will keep the phase at the origin $\phi = 0$.

Increase the fundamental frequency of the analog sinusoid f_0 above the Nyquist frequency $f_{Ny} = \frac{f_s}{2}$, and examine the variations of the apparent frequencies when $f_{Ny} \leq f_0 \leq f_s$.

Q7: In this frequency range of f_0, how the apparent frequency f_{App} of the sampled discrete-time sinusoid does it evolve ?

Q7-R: The apparent frequency in the region $f_{Ny} \leq f_0 \leq f_s$ seems to decreases until it goes to zero as we increases fundamental frequency f_0 up to f_s. We can show it by the relation $f_{App} = |f_s - f_0|$

Compare the samples obtained for discrete frequencies ν differing by an integer value N (for example $\nu = 0.25, 1.25, 2.25$).

Q8: What do you observe?

Q8-R:They provide the same sample values for $\nu = 0.25, \nu = 1.25, \nu = 2.25$. This is also evident mathematically as

$cos(2\pi\nu n) = cos(2\pi(\nu + N)n) = cos(2\pi\nu n + 2\pi Nn) = cos(2\pi\nu n)$ as $2\pi Nn$ is a multiple of 2π and cosine is an integer.

The apparent frequency of two discrete-time sinusoids differing by some integer is the same, so are the extracted samples. A discrete-time sinusoid of discrete frequency $\nu_0 \in [-0.5; 0.5]$ represents an infinity of analog sinusoids, called its **aliases**, with frequencies $f_k = (k + \nu_0)f_s\ k \in \mathbb{Z}$. All these aliases are present in the spectrum of the discrete-time sinusoid. (For example for $\nu = 0.25$ we have $f_k = (k + 0.25) * (20.00)$, for $k = 0$, $f_1 = 5.00$, for $k = 1$, $f_2 = 25$, for $k = 2$, $f_3 = 45$, Now all these f_1, f_2, f_3 will produce same apperent frequency. Thus these f_1, f_2, f_3 are now the aliases of $\nu = 0.25$.

I.3 "Bad" sampling (undersampling)

When the Shannon-Nyquist condition is not fulfilled, the analog sinusoid will be undersampled, which means that the apparent frequency f_{App} of the sampled discrete-time sinusoid will not correspond to the actual frequency of the sampled analog sine.

A discrete-time sinusoid, or any discrete-time signal, provided it is correctly sampled ($f_s \geq \frac{f_{Max}}{2}$) allows to recover the initial anaolg signal, usually through low-pass filtering.

In contrary, an undersampled discrete-time sinusoid ($\nu = (k + \nu_0) > 0.5, k \in \mathbb{Z}_0$) will generally not allow to recover the initial signal, but rather a sinusoid corresponding to its principal alias, *i.e.* of an apparent frequency $f_{App} = \nu_0$. ; this corresponds to a **spectral folding** towards lower frequencies (in french "*Repliement spectral*").

In the simulator, remember that the background color will shift from white to yellow if the Shannon-Nyquist condition is not fulfilled, corresponding to undersampling the analog sinusoid. By checking the button "*Show principal alias*", the principal alias will appear in green. In undersampling conditions, the samples do belong to both the analog sinusoid and the principal alias, located at a frequency $f_s - f_0$ when $f_0 \in [\frac{f_s}{2}; f_s]$.

Whenever an analog signal of any shape is sampled, spectral folding may be harmful, because a folded component will appear at a lower frequency, where it can interfere with other components naturally present in the analog signal, which will appear distorted, and not faithful to the original.

in practice, in analog to digital converters working at a fixed sampling frequency f_s, the analog input signal is first of all filtered by a low-pass (analog) filter with a cut-off frequency $f_c = \frac{f_s}{2}$, which will cancel or attenuate the frequencies larger than $\frac{f_s}{2}$. This low-pass filter, called a guard filter, insures that no spectral component will undergo a spectral folding, thereby keeping the original signal unperturbed in the frequency range up to $\frac{f_s}{2}$.

In [0]: