Lab Session 3

Basile Dubois Bonnaire

Ignat Sabaev & Antoine Millon

Exercice 1

We have $f(x) = e^{i2\pi F_0 x}$ and a sampling step $a = \frac{1}{F_e}$ where $F_e = 32$.
Given the expression of f the fourier transform should give us a peak at F_0.

1. Reduction of $S_N(\lambda)$

$S_N(\lambda)$ is a geometric sum from $-N$ to $N-1$. Doing the appropriate factorisation gives us

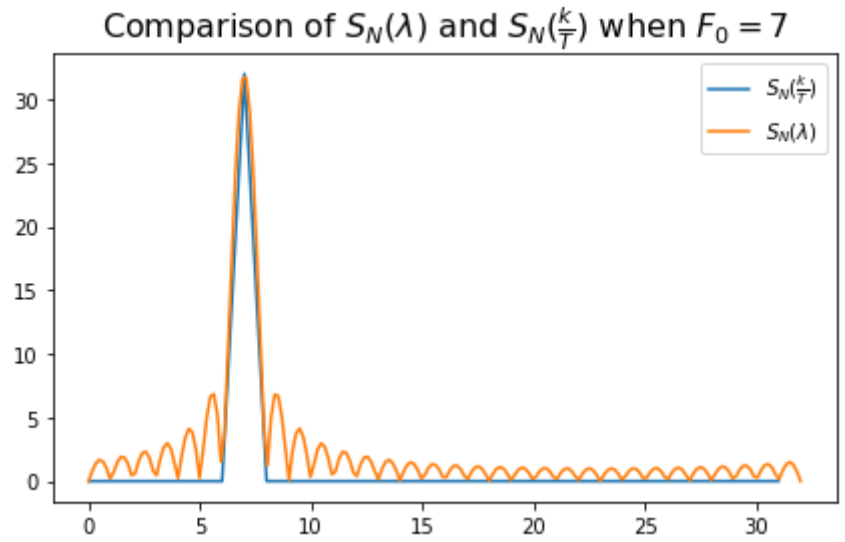
$$
\begin{aligned}
S_N(\lambda) &= e^{-i2\pi Na(F_0-\lambda)} \sum_{n=0}^{2N-1} e^{i2\pi na(F_0-\lambda)} \\
&= e^{-i2\pi Na(F_0-\lambda)} \frac{1 - e^{i2\pi 2Na(F_0-\lambda)}}{1 - e^{i2\pi a(F_0-\lambda)}} \\
&= \frac{e^{i2\pi Na(F_0-\lambda)} - e^{i2\pi Na(F_0-\lambda)}}{e^{i\pi a(F_0-\lambda)} - e^{i\pi a(F_0-\lambda)}} \frac{2i}{2i} e^{i\pi(F_0-\lambda)a} \\
&= \frac{\sin(2\pi Na(F_0-\lambda))}{\sin(\pi a(F_0-\lambda))} e^{i\pi(F_0-\lambda)a}
\end{aligned}
$$

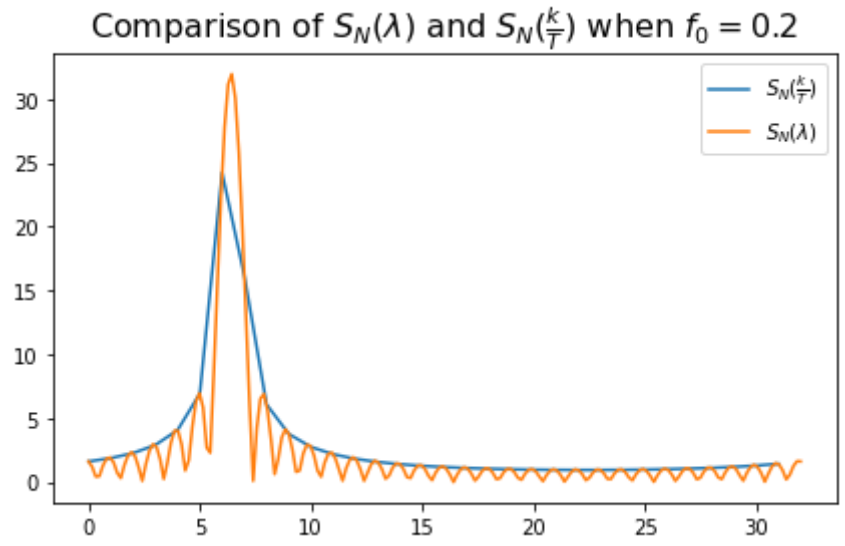
And then introducing $a = \frac{1}{F_e}$ and $f_0 = \frac{F_0}{F_e}$ We get

$$
S_N(\lambda) = \frac{\sin(2\pi N(f_0 - \frac{\lambda}{F_e}))}{\sin(\pi(f_0 - \frac{\lambda}{F_e}))} e^{i\pi(f_0 - \frac{\lambda}{F_e})}
$$

In [2]:

```python
# Comparison between approximated and analytical solutions
Fe = 32
N = 16

SNlambda = lambda l, f0, Fe : np.sin(2*np.pi * N * (f0 - l/Fe)) * np.exp(-1j*np.pi*(f0-l/Fe)) / np.sin(np.pi * (f0 - l/

# Calculation of SN(k/T) using fft when F0=7 :

f = lambda x, F0 : np.exp(2j*np.pi*F0*x)

F0 = 7
temp_signal = f(np.arange(-N, N)/Fe, F0)

SNk = np.fft.fft(temp_signal)

## plots

X = np.linspace(0, Fe, 200)

plt.plot(np.arange(0, 2*N) / (2*N/Fe), np.abs(SNk), label=r"$S_N(\frac{k}{T})$")
plt.plot(X, np.abs(SNlambda(X, F0/Fe, Fe)), label=r"$S_N(\lambda)$")
plt.title(r"Comparison of $S_N(\lambda)$ and $S_N(\frac{k}{T})$ when $F_0 = 7$", fontsize=16)
plt.legend()
plt.tight_layout()
plt. show()

# Calculation of SN(k/T) using fft when f0=0.2 :
f0 = 0.2

F0 = f0 * Fe
temp_signal = f(np.arange(-N, N)/Fe, F0)

SNk = np.fft.fft(temp_signal)

## plots

plt.plot(np.arange(0, 2*N) / (2*N/Fe), np.abs(SNk), label=r"$S_N(\frac{k}{T})$")
plt.plot(X, np.abs(SNlambda(X, f0, Fe)), label=r"$S_N(\lambda)$")
plt.title(r"Comparison of $S_N(\lambda)$ and $S_N(\frac{k}{T})$ when $f_0 = 0.2$", fontsize=16)
plt.legend()
plt.tight_layout()
plt. show()
```





- With F_0 the approximation $S_N(k/T)$ is flat, equal to 0 exept around 7 where a peak appears. The theoretical value $S_N(\lambda)$ oscillates before and after the peak.
- With f_0 the approximation better approximates the theoretical value by always getting the peaks of the oscillation.

Exercice 2

In this exercice we consider the discrete signals $f = \sum_{n\in\mathbb{Z}} f_n \delta_{na}$ and $d = \frac{1}{4}(\delta_0 + \delta_1 + \delta_2 + \delta_3)$.

We study the convolution $g = d \star f = \sum_{n\in\mathbb{Z}} g_n \delta_{na}$.

1. Expression of g_n

The convolution product on distributions gives us

$$< d \star f, \phi >= \frac{1}{4} \sum_{k=0}^{3} \sum_{n\in\mathbb{Z}} f_n \phi((k+n)a)$$

Which implies

$$\begin{aligned} d \star f &= \frac{1}{4} \sum_{k=0}^{3} \sum_{n\in\mathbb{Z}} f_n \delta_{(k+n)a} \\ &= \frac{1}{4} (\sum_{n\in\mathbb{Z}} f_n \delta_{na} + \sum_{n\in\mathbb{Z}} f_n \delta_{(1+n)a} + \sum_{n\in\mathbb{Z}} f_n \delta_{(2+n)a} + \sum_{n\in\mathbb{Z}} f_n \delta_{(3+n)a} \\ &= \sum_{n\in\mathbb{Z}} \underbrace{\frac{1}{4}(f_n + f_{n-1} + f_{n-2} + f_{n-3})}_{g_n} \delta_n \end{aligned}$$

So $g_n = \frac{1}{4}(f_n + f_{n-1} + f_{n-2} + f_{n-3})$. By the N-periodicity of f_n, the N-periodicity of g_n is immediate.

2. Computation of g using DFT and IDFT

The fourier transform of a convolution is the product of the fourier transforms of the operands i.e. $\widehat{d \star f} = \hat{d}\,\hat{f}$.

Thus to obtain g using DFT we calculate the DFT of d and f separately then apply the inverse transform on the product $\hat{d}\,\hat{f}$.

Coresponding program

In [3]:

```
# Definition of the function to calculate the convolution between two function
def convolution(a, b):
    """
    Applies the convolution product a*b and returns it by first calculating the DFT of a and b,
```

```
    multiplying them together and returning the inverse DFT.

    Arguments :
        a : ndarray, first operand of the convolution
        b : ndarray, second operand of the convolution
    """
    a_hat = np.fft.fft(a)
    b_hat = np.fft.fft(b)
    c = np.fft.ifft(a_hat * b_hat)

    return c
```

3. Application

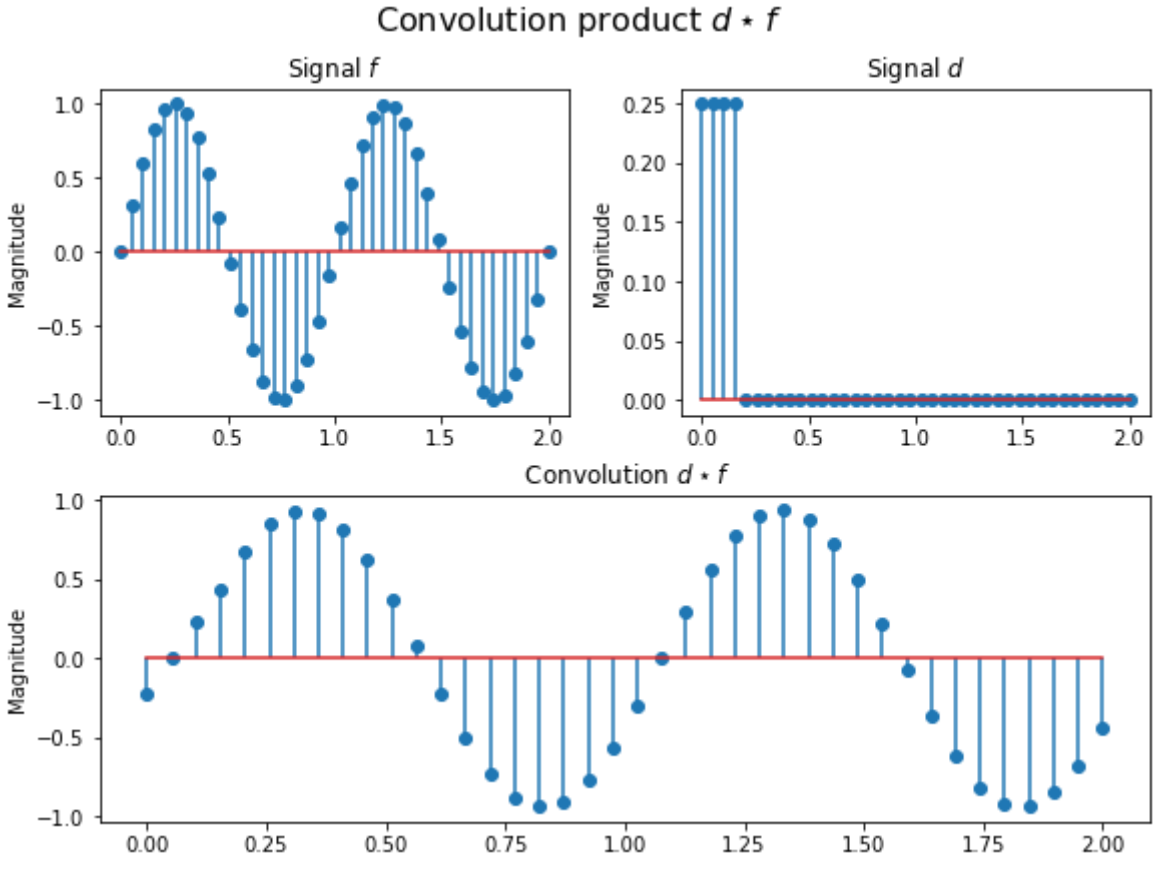
We consider $f(x) = \sin(2\pi x)$ on the interval $[0, 2[$ sampled at a 20 Hz frequency.

The discrete signal obtained can thus be written $\sum_{n=0}^{39} f(na)\delta_{na}$ where $a = \frac{1}{20}$.

d remains the same as before.

In [4]:

```
# Application of the previous program

# sampling rate
a = 1/20

# sampling of [0,2[
X = np.linspace(0,2,int(2/a))

# definition of f
f = lambda x : np.sin(2*np.pi*x)

# definition of d
# no matter the sampling, d will always be one fourth on the first 4 elements
d = np.zeros(int(2/a))
d[0:4]=1/4

g = convolution(d, f(X))

# Plotting shenanigans
fig = plt.figure(constrained_layout = True)
fig.set_size_inches(8,6)
spec = fig.add_gridspec(ncols=2, nrows=2, figure = fig)
f_ax1 = fig.add_subplot(spec[0,0])
f_ax1.set(title=r"Signal $f$", ylabel="Magnitude")

f_ax2 = fig.add_subplot(spec[0,1])
f_ax2.set(title=r"Signal $d$", ylabel="Magnitude")

f_ax3 = fig.add_subplot(spec[1,:])
f_ax3.set(title=r" Convolution $d \star f$", ylabel="Magnitude")

f_ax1.stem(X, f(X))
f_ax2.stem(X, d)
f_ax3.stem(X,g.real)

fig.suptitle("Convolution product $d\star f$", fontsize=16)
plt.show()
```



4. Deconvolution

Using the same property we used in **2.2** and supposing $\hat{d} \neq 0$ everywhere, we have

$$\hat{f} = \frac{\hat{g}}{\hat{d}}$$

And f can be recovered using the inverse Fourier transform.

If $\hat{d}$ vanishes at some points we have a problem as $\frac{\hat{g}}{\hat{d}}$ is undefined. One way to work with such a hurdle is to replace the zero value with something small but different from 0. This idea is expanded below with the introduction of the inverse filter.

deconvolution code

In [5]:
```
def deconvolution(d, g, c) :
    """
    if g = f*d with f a signal, returns f using the dft method

    Parameters:
        d = ndarray : one of the operand of the convolution
        g = ndarray : the result of the convolution
        c = float   : cutoff point for the inverse filter
    Returns :
        f = ndarray : the second operand of the convolution
    """

    d_hat = np.fft.fft(d)
    g_hat = np.fft.fft(g)

    # applies the inverse filter to d_hat with precision c
    d_hat_inv = inverse(d_hat, c)

    f = np.fft.ifft(g_hat * d_hat_inv)

    return f
```

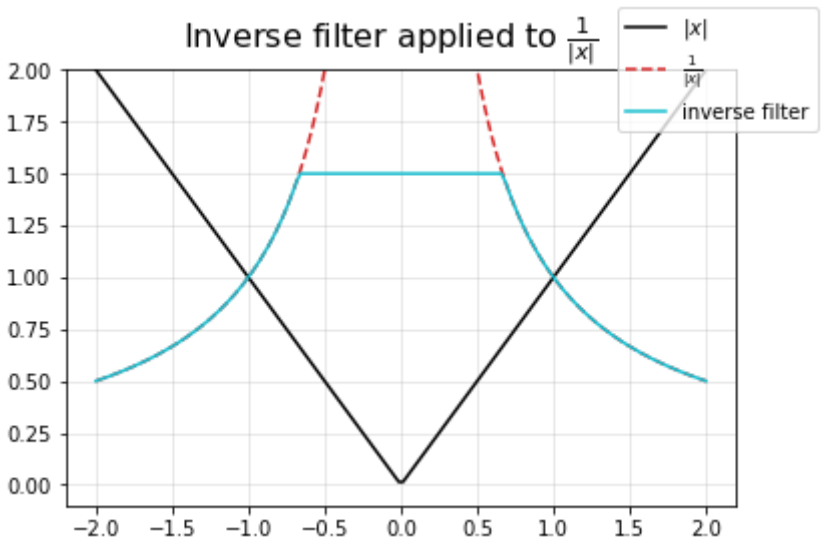
5. Inverse filter

Inverse filter implementation

In [6]:
```
def inverse(f, c) :
    """
    Inverse filter meant to avoid undefined values when dividing.

    Parameters:
        f = ndarray : the function to inverse.
        c = float   : cutoff point. When |f| gets below this value, 1/c is returned.

    Returns :
        f_inv = ndarray : 1/f if |f| > c, 1/c otherwise.
    """

    N = len(f)
    f_inv = np.zeros(N, dtype=complex)

    for i in range(N) :
        if abs(f[i]) > c :
            f_inv[i] = 1/f[i]
        else :
            f_inv[i] = 1/c

    return f_inv
```

Illustration

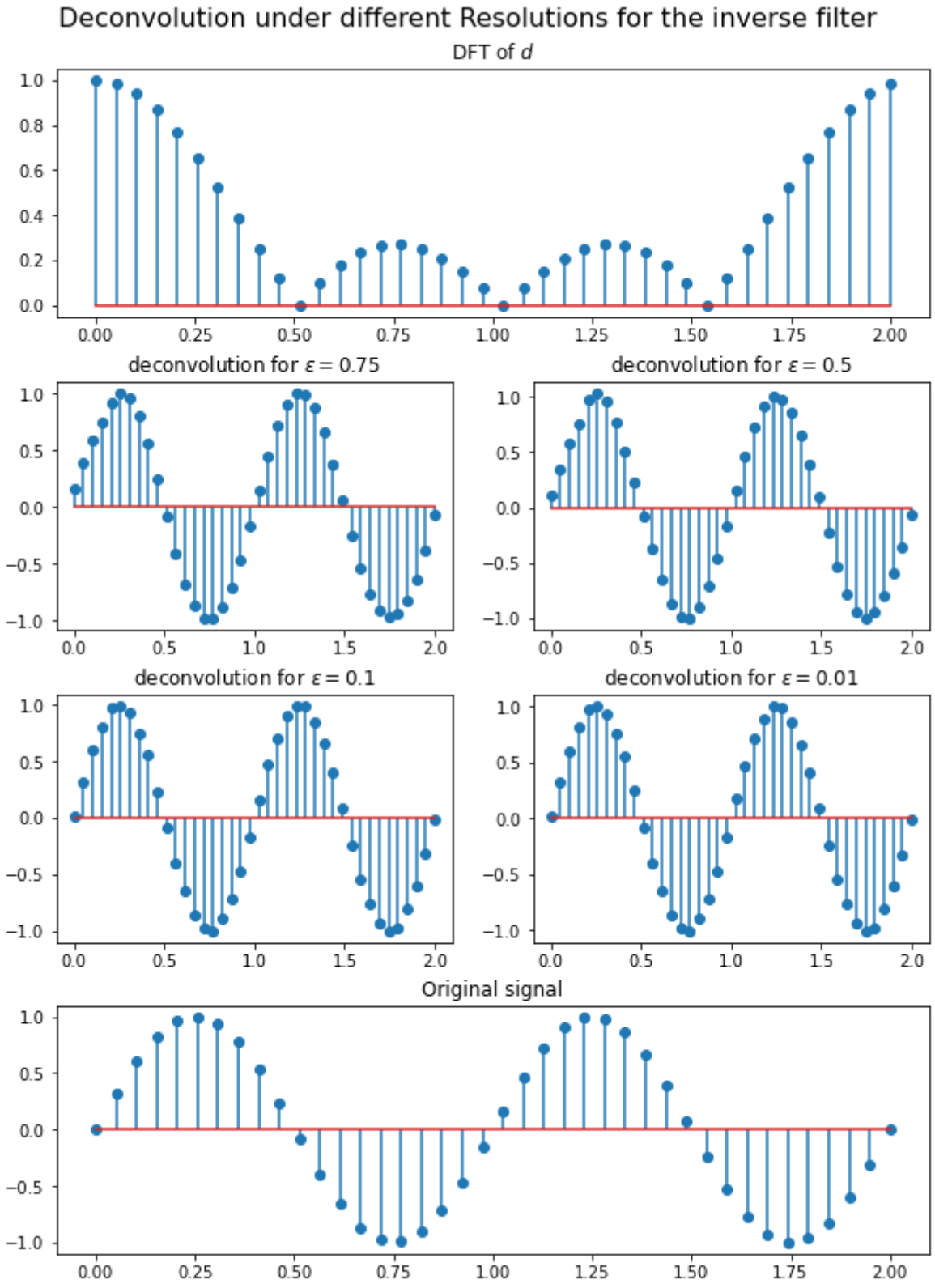
In [47]:
```
# Illustration of the inverse filter using the absolute value function

X2 = np.linspace(-2, 2, 150)
abs_inv = inverse(abs(X2), 1/1.5)
real_inv = inverse(abs(X2), 0.1)

fig2, ax = plt.subplots()

ax.plot(X2, abs(X2), 'k-', label= r"$|x|$")
ax.plot(X2, real_inv, 'C3--', label= r"$\frac{1}{|x|}$")
ax.plot(X2, abs_inv, 'C9-', label= "inverse filter")
ax.set_ylim(-0.1, 2)
ax.grid(alpha=.35)
fig2.suptitle(r"Inverse filter applied to $\frac{1}{|x|}$", fontsize=16)

fig2.legend()
plt.show()
```



Recovery of f using the inverse filter

In [41]:

```
# plotting shenanigans ---------------------------------------------
fig3 = plt.figure(constrained_layout=True)
fig3.set_size_inches(8,11)
fig3.suptitle("Deconvolution under different Resolutions for the inverse filter", fontsize=16)
spec2 = fig3.add_gridspec(4,2)

## Plot of the original signal
f_ax = fig3.add_subplot(spec2[3,:])
f_ax.stem(X, f(X))
f_ax.set_title("Original signal")

## Plot of the dft of d, to see the zeros
d_hat_ax = fig3.add_subplot(spec2[0,:])
d_hat_ax.stem(X, abs(np.fft.fft(d)))
d_hat_ax.set_title("DFT of $d$")

ax1 = fig3.add_subplot(spec2[1,0])
ax2 = fig3.add_subplot(spec2[1,1])
ax3 = fig3.add_subplot(spec2[2,0])
ax4 = fig3.add_subplot(spec2[2,1])
axes = np.array([ax1, ax2, ax3, ax4])

# Actual Work ------------------------------------------------------

F = np.zeros((4, len(X)), dtype=complex)
errors = [.75, .5, .1, 1E-2]

for i in range(4):
    F[i] = deconvolution(d, g, errors[i])
    axes[i].stem(X, F[i].real)
    axes[i].set_title("deconvolution for $\epsilon = $" + str(errors[i]))

plt.show()
```



There are three zeroes in the DFT of d. The inverse filter is needed (although, the true values in `d_hat` are not exactly zero, they get very close and we might want to avoid the overhead of the inverse calculation or risk of overflow for very small values).
We can see a discrepancy in the beginning of the signal when ϵ is high. The first values are higher than they should be (Particularly clear for the first value which is 0 originally). This quickly disolves once ϵ gets small enough.

Exercice 3

We consider the filter (h_n) and its transfer function, $H(\lambda) = \sum_{n\in\mathbb{Z}} h_n e^{-i2\pi n\lambda}$.

1 Calculation of H

H is a 1-periodic function so it can be expressed via its fourier serie : $H(\lambda) = \sum_{n\in\mathbb{Z}} c_n e^{i2\pi n\lambda}$ where the c_n are the fourier coefficiants of H.

It comes then that $c_n = h_{-n}$. We have

$$\begin{aligned} c_n &= \int_{-\frac{1}{2}}^{\frac{1}{2}} H(x) e^{-i2\pi xn}\, dx \\ &= \int_{-\frac{1}{4}}^{\frac{1}{4}} e^{-i2\pi xn}\, dx \\ &= \frac{1}{-2\pi ni}(e^{-\frac{i2\pi n}{4}} - e^{\frac{i2\pi n}{4}}) \\ &= \frac{\sin(\frac{\pi}{2}n)}{\pi n} \\ &= \frac{1}{2}\,\mathrm{sinc}(\frac{\pi}{2}n) \end{aligned}$$

So $h_n = \frac{1}{2}\,\mathrm{sinc}(-\frac{\pi}{2}n) = -\frac{1}{2}\,\mathrm{sinc}(\frac{\pi}{2}n)$

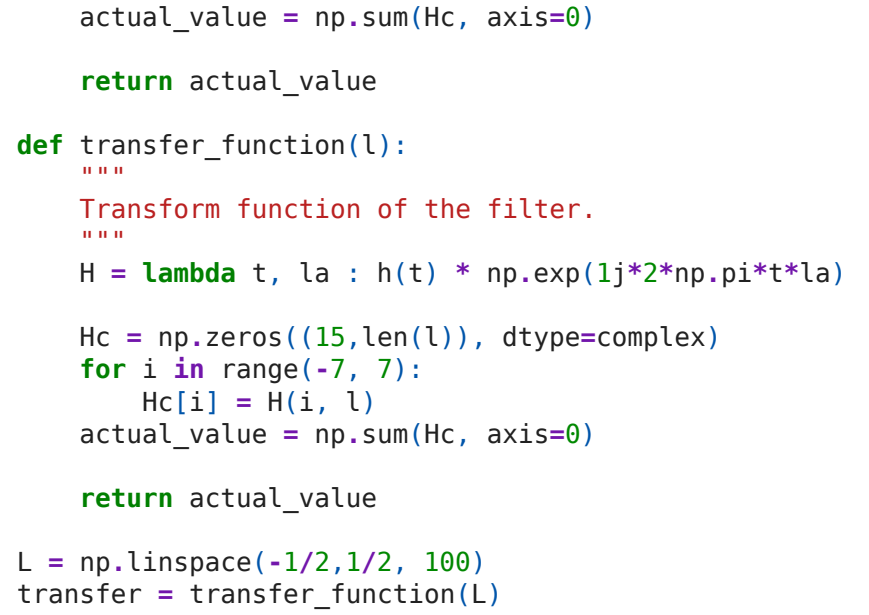
We now truncate the filter To obtain N finite values. Thus two cases unearth : N odd or N even.

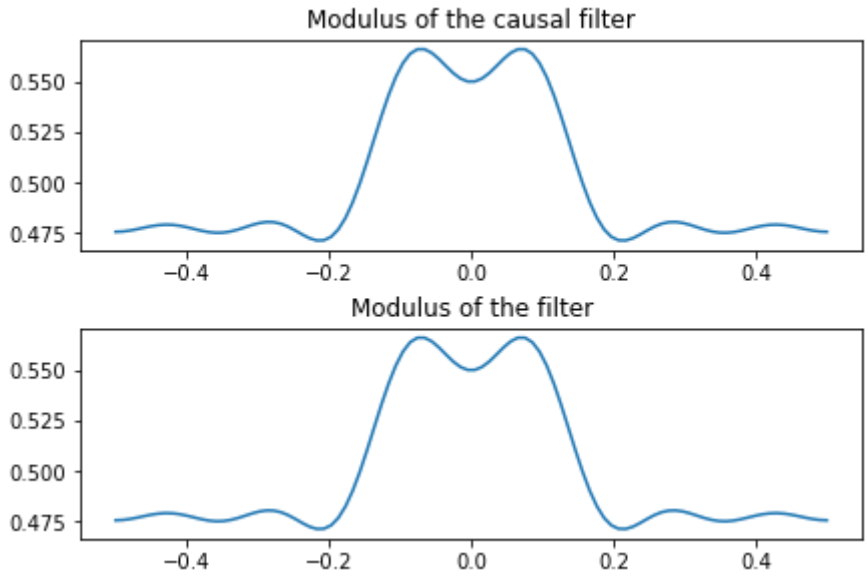
2. N odd

In this case we choose $N = 15$. We have $H(\lambda) = \sum_{n=-7}^{7} h_n e^{-i2\pi n\lambda}$.

However, this filter is not causal as its support is not in $[0, +\infty[$ and takes value from future event. A causal version would be $\sum_{0}^{14} h_{n-7}\delta_n$. Taking its fourier transform we get $e^{i2\pi 7\lambda} \sum_{n=-7}^{7} h_n e^{i2\pi n\lambda}$ which corresponds to a shift of magnitude 7 in frequency.

In [45]:
```
#calculates the h_n
h = lambda n : -1/2 * np.sinc(np.pi * n * 2)


fig_phase, ( ax_phase_causal, ax_phase ) = plt.subplots(2,1, constrained_layout=True)
fig_abs, ( ax_abs_causal, ax_abs ) = plt.subplots(2,1, constrained_layout=True)


def transfer_function_causal(l):
    """
    Transform function of the causal filter.
    """
    H = lambda t, la : h(t-7) * np.exp(1j*2*np.pi*t*la)

    Hc = np.zeros((15,len(l)), dtype=complex)
    for i in range(0, 14):
        Hc[i] = H(i, l)
    actual_value = np.sum(Hc, axis=0)

    return actual_value

def transfer_function(l):
    """
    Transform function of the filter.
    """
    H = lambda t, la : h(t) * np.exp(1j*2*np.pi*t*la)

    Hc = np.zeros((15,len(l)), dtype=complex)
    for i in range(-7, 7):
        Hc[i] = H(i, l)
    actual_value = np.sum(Hc, axis=0)

    return actual_value

L = np.linspace(-1/2,1/2, 100)
transfer = transfer_function(L)
transfer_causal = transfer_function_causal(L)

ax_phase_causal.plot(L, np.angle(transfer_causal))
ax_phase_causal.set_title("Phase of the causal filter")
ax_phase.plot(L, np.angle(transfer))
ax_phase.set_title("Phase of the original filter")

ax_abs_causal.plot(L, np.abs(transfer_causal))
ax_abs_causal.set_title("Modulus of the causal filter")

ax_abs.plot(L, np.abs(transfer))
ax_abs.set_title("Modulus of the filter")
print('')
```





We can see that the phase of the causal filter is now linear while the modulus remains unchanged.

3. N is even

We have

$$\begin{aligned}\tilde{h_n} &= \int_{-\frac{1}{2}}^{\frac{1}{2}} H(x) e^{i\pi x(2n-1)}\, dx \\ &= \int_{-\frac{1}{4}}^{\frac{1}{4}} e^{i\pi x(2n-1)}\, dx \\ &= \frac{1}{\pi(2n-1)i}(e^{\frac{i\pi(2n-1)}{4}} - e^{\frac{-i2\pi(2n-1)}{4}}) \\ &= \frac{\sin(\frac{\pi}{4}(2n-1))}{2\pi(2n-1)} \\ &= \frac{1}{2}\,\mathrm{sinc}(\frac{\pi}{4}(2n-1))\end{aligned}$$

In [46]:

```
# necessary options and imports
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```