Lab session 3

Approximation of a spectrum, filter analysis

report by Grebennikova Anna & Kseniia Ovchinnikova

Université Grenoble Alpes, UGA

Exercise 1.Approximation of a spectrum, filter analysis

Let f be a discrete signal corresponding to the samples $(f(na))_{n\in\mathbb{Z}}$, supposed to be slowly increasing, its Fourier transform reads:

$$\hat{f}(\lambda) = \sum_{n=-\infty}^{+\infty} f(na)e^{-2i\pi n\lambda}.$$

Assume that the signal is observed only on the interval $[-Na, (N-1)a]$, the approximative spectrum corresponds to:

$$S_N(\lambda) = \sum_{n=-N}^{N-1} f(na)e^{-2i\pi n\lambda}.$$

Considering $\lambda_k = \frac{k}{2Na} = \frac{k}{T}$, i.e.:

$$S_N\left(\frac{k}{T}\right) = \sum_{n=-N}^{N-1} f(na)\omega_{2N}^{-nk},$$

with $\omega_q^p = e^{2i\pi\frac{p}{q}}$ which can be computed by means of DFT.
Let us consider $f(na)$ with $f(x) = e^{2i\pi F_0 x}$, with a sampling step $a = \frac{1}{F_e}$.

Question 1

Let us define $f_0 = \frac{F_0}{F_e}$ and assume $N = 16$. We will give the expression of $S_N(\lambda)$ with respect to f_0. Firstly we substitute $f(x) = e^{2i\pi F_0 x}(f(na) = e^{2i\pi F_0 na})$ into expression for $S_N(\lambda)$:

$$S_N(\lambda) = \sum_{n=-N}^{N-1} e^{2i\pi na(F_0-\lambda)}.$$

Knowing that $a = \frac{1}{F_e}$ and using $f_0 = \frac{F_0}{F_e}$ we can rewrite the expression:

$$S_N(\lambda) = \sum_{n=-N}^{N-1} e^{2i\pi n\frac{1}{F_e}(F_0-\lambda)} = \sum_{n=-N}^{N-1} e^{2i\pi n(\frac{F_0}{F_e}-\frac{\lambda}{F_e})} = \sum_{n=-N}^{N-1} e^{2i\pi n(f_0-a\lambda)}.$$

For simplification of expressions let us define $t = \pi(f_0 - \lambda a)$. Now we calculate this sum as the sum of geometric sequence $S_N = \frac{b_1(1-q^{2N})}{1-q}$, where $b_1 = e^{-2iNt}$, $q = e^{2it}, q^{2N} = e^{4iNt}$:

$$S_N(\lambda) = e^{-2iNt}\frac{1-e^{4iNt}}{1-e^{2it}} = e^{-2iNt}\frac{e^{2iNt}(e^{-2iNt}-e^{2iNt})}{e^{it}(e^{-it}-e^{it})} = e^{-it}\frac{sin(2Nt)}{sin(t)}.$$

We substitute $t = \pi(f_0 - \lambda a)$:

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

So the modulus of $S_N(\lambda)$ equals:

$$|S_N(\lambda)| = \left|\frac{sin(2N\pi(f_0-\lambda a))}{sin(\pi(f_0-\lambda a))}\right|.$$

And if $N = 16$:

$$|S_{16}(\lambda)| = \left|\frac{sin(32\pi(f_0-\lambda a))}{sin(\pi(f_0-\lambda a))}\right|.$$

Let us also note that it's easy to see that there exists a point of discontinuity where $f_0 - \lambda a = 0$. Knowing that $a = \frac{1}{F_e}$ and $f_0 = \frac{F_0}{F_e}$ we can rewrite that as $\frac{F_0}{F_e} - \lambda\frac{1}{F_e} = F_0 - \lambda = 0$. So we can write limit at this point as follows:

$$\lim_{\lambda\to F_0}|S_N(\lambda)| = \lim_{\lambda\to F_0}\left|\frac{sin(2N\pi(\frac{F_0}{F_e}-\lambda\frac{1}{F_e}))}{sin(\pi(\frac{F_0}{F_e}-\lambda\frac{1}{F_e}))}\right| = \left|\frac{2N\pi(\frac{F_0}{F_e}-\lambda\frac{1}{F_e})}{\pi(\frac{F_0}{F_e}-\lambda\frac{1}{F_e})}\right| = 2N.$$

So on practice we should observe a peak equal to 2N at the point where $\lambda = F_0$.

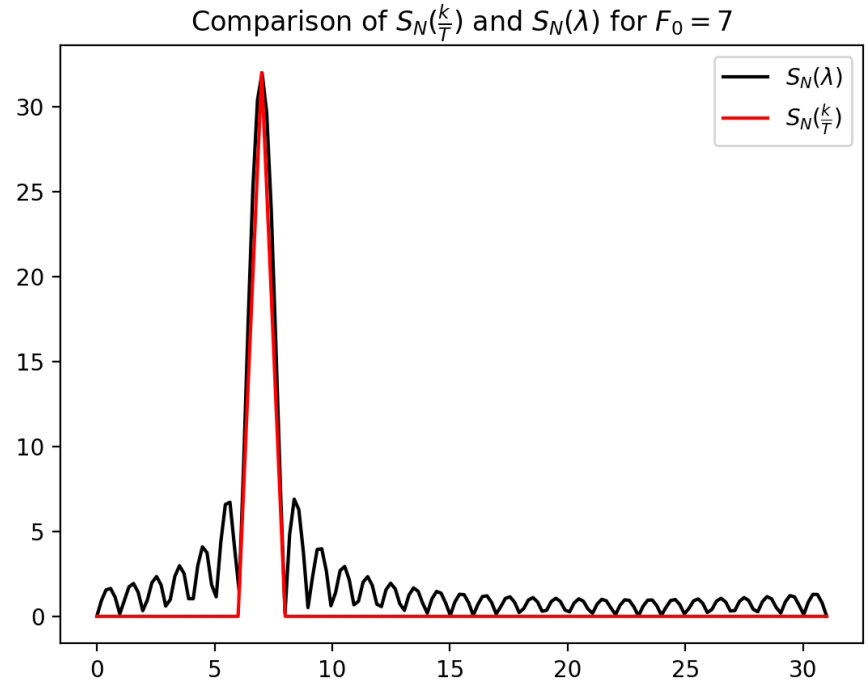
Question 2

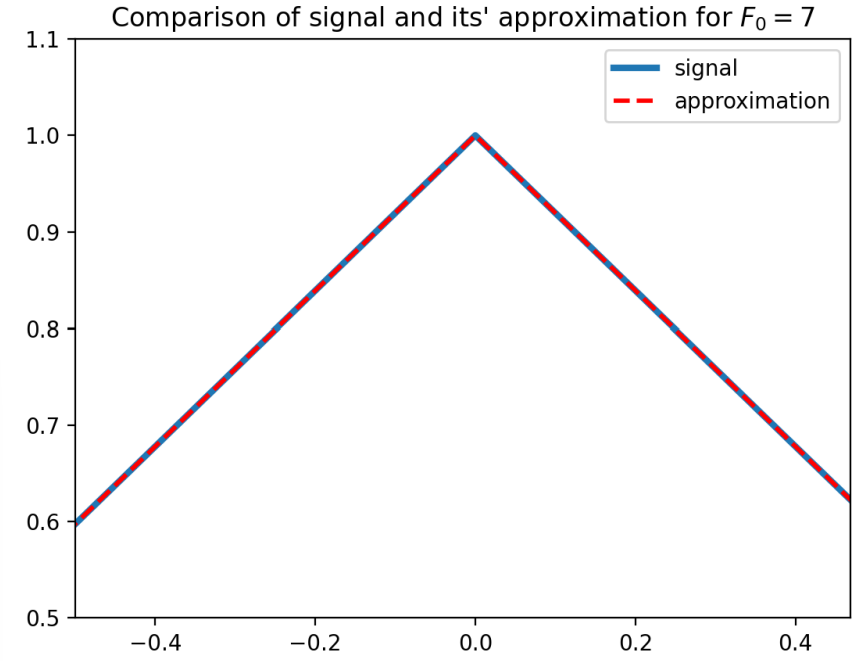
With $F_e = 32$ we compute $S_N\left(\frac{k}{T}\right)$ when $F_0 = 7$ and compare with $S_N(\lambda)$. We also plot the graph of signal and it's approximation with the use of $S_N\left(\frac{k}{T}\right)$ to be sure of our approximation.

We write a program:

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

#We define constants
N = 16
Fe = 32
F0 = 7
f0 = F0/Fe
a = 1/Fe

#We calculate sn(lambda)
l = np.linspace(0, 2*N - 1, 10*N)
sn_l_abs = np.zeros(len(l))
sn_l_abs = abs(np.sin(2*np.pi*N*(f0-l*a))/(np.sin(np.pi*(f0-l*a))))

#We plot sn(lambda)
plt.plot(l, sn_l_abs, label = r'$S_N(\lambda)$', color = 'k')

#We calculate sn(k/T)
ngrid = np.linspace(-N, N-1, 2*N)
f = np.exp(2j*np.pi*f0*ngrid)
sn_kT_abs = np.zeros(len(ngrid))
sn_kT_abs = abs(np.fft.fft(f))
ax = np.linspace(0, 2*N - 1, 2*N)

#We plot sn(k/T)
plt.plot(ax, sn_kT_abs, label = r'$S_N(\frac{k}{T})$', color = 'r')
plt.legend()
plt.title(r'Comparison of $S_N(\frac{k}{T})$ and $S_N(\lambda)$ for $F_0=7$')

#we plot signal
plt.figure(2)
plt.plot(ngrid, f, label = 'signal', linewidth = 3)

#We calculate approximation of the signal
ck_shift = -np.fft.fftshift(np.fft.fft(f)/(2*N))
F = 0
for i in range(-N, N):
  F += ck_shift[i+N]*np.exp(2j*np.pi*i*ngrid*a)

#We plot approximation of the signal
plt.plot(ngrid,F,'--', color = 'r', label = 'approximation', linewidth = 2)
plt.xlim(-N*a,(N-1)*a)
plt.ylim(0.5,1.1)
plt.title("Comparison of signal and its' approximation for $F_0=7$");
plt.legend()
plt.show()

```

Our program prints the following graphs:





As we stated in the **Question 1**, we observe peak equal to $2N = 2\dot{1}6 = 32$ at the point $\lambda = F_0 = 7$ and when $f_0 = 0.2$. Compare with $S_N(\lambda)$ in each case and give an interpretation.

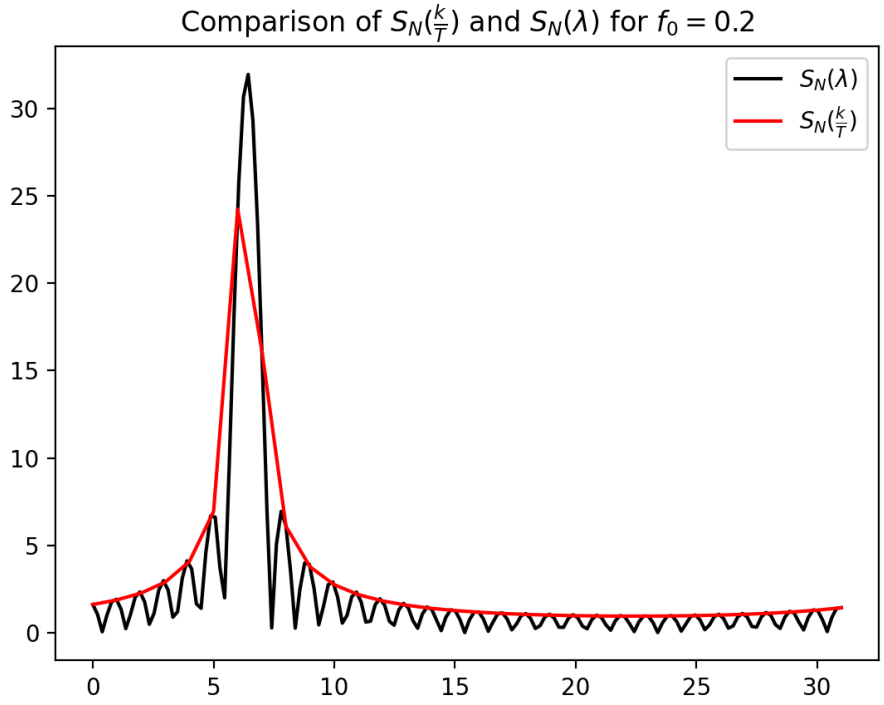
For the case when $f_0 = 0.2$ we use the above stated code with the only change that when defining a constants instead of lines 7-8 we use the following ones:

```
f0 = 0.2
F0 = Fe*f0
```

And instead of line 29 we use the following one:

```
plt.title(r'Comparison of $S_N(\frac{k}{T})$ and $S_N(\lambda)$ for $f_0=0.2$')
```

Our program now prints the following graph:



For the theoretically computed $S_N(\lambda)$ we observe peak equal to $2N = 2 \cdot 16 = 32$ at the point $\lambda = F_0 = f_0 \cdot F_e = 0.2 \cdot 32 = 6.4$ as it should be. As for the $S_N(\frac{k}{T})$ we observe certain issues - because now the point of discontinuity $\lambda = F_0$ is not integer and DFT uses only integer points. So for $S_N(\frac{k}{T})$ peak happens in the closest integer point $\lambda = 6$.

Exercise 2. On the use of DFT for filtering

Question 1

Let us consider the discrete signal $f = \sum_{n\in\mathbb{Z}} f_n \delta_{na}$, where (f_n) is periodic with period N, and the discrete signal $d = \sum_{n=0}^{3} \frac{1}{4}\delta_{na}$. Let g be defined as:

$$g = d * f = \sum_{n\in\mathbb{Z}} g_n \delta_{na}$$

By definition, the circular convolution between the sequences d and f, both periodic with period N, is given by:

$$g_n = (d * f)_n = \sum_{k=0}^{N-1} d_k f_{n-k} \mod N$$

In our case, f is periodic, and d has only four non-zero coordinates:

$$d = \sum_{n=0}^{N-1} d_n \delta_{na} = \sum_{k=0}^{3} d_k \delta_{ka}, \quad \text{where } d_k = \frac{1}{4}$$

Thus, the final formula for g_n looks like:

$$g_n = \sum_{k=0}^{N-1} \frac{1}{4} f_{n-k} \mod N$$

So, we conclude that g_n is a sum of the periodic function f with period N, and the function g itself is a sum of the coordinates g_n. Therefore, g has period N because it is a sum of periodic functions f_n.

Question 2

One of the properties of convolution and the Fourier transform is:

$$DFT(d * f) = DFT(d) \times DFT(f)$$

where the operation $\times$ represents element-wise multiplication.

By applying the DFT to the convolution of functions, we obtain the coefficients $\tilde{g}_k$ of the function g in the Fourier domain.

To compute g, we first need to compute $DFT(d)$ and $DFT(f)$, then multiply the resulting transforms element-wise. After that, performing the inverse Fourier transform on the coefficients $\tilde{g}_k$ will yield the function g.

The corresponding program:

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

def func(x):
  return np.sin(2*np.pi*1*x)

def delta(x,n,a):
  if x == n*a:
    return 1
  else:
    return 0

def function_f(x_i,n,a):
  sum = 0
  for j in range(n):
    sum += func(j*a)*delta(x_i,j,a)
  return sum

def function_d(x_i,n,a):
  sum = 0
  for j in range(4):
    sum += 1/4*delta(x_i,j,a)
  return sum

# number of point
n = 40
# the value of a
a = 1/20
# the arange of x for the value of funtions
x = np.arange(0,2,a)


# initialization of fuctions d and f
f = []
d = []
for i in x:
  f.append(function_f(i,n,a))
  d.append(function_d(i,n,a))

# dft of f_k and d_k
f_k = np.fft.fft(f)
d_k = np.fft.fft(d)

# using the property of DFT of convolution fuction
g_k = f_k * d_k

```

```
# inverse dft to get the values of g
g = np.fft.ifft(g_k)

# plot the absolute value of coefficients and the real part of g function

plt.figure()
plt.bar(x, np.abs(g_k), label = "Calculated coefficients g_k", width = 0.01, color = "b")
plt.legend()
plt.show()

plt.figure()
plt.plot(x, g.real, label="The function g after inverse DFT", color="r")
plt.legend()
plt.show()
```

Question 3

We consider a sinusoid f with a frequency of 1 Hz defined on the interval $[0, 2[$ with $a = \frac{1}{20}$, i.e., there are 40 points and 40 corresponding values of the function f. Thus, we obtain the discrete distribution

$$f = \sum_{n=0}^{39} f_n \delta_{na}, \quad \text{where } f_n = f(na).$$

Using the corresponding program, we computed the coefficients for g after using DFT for functions d and f.

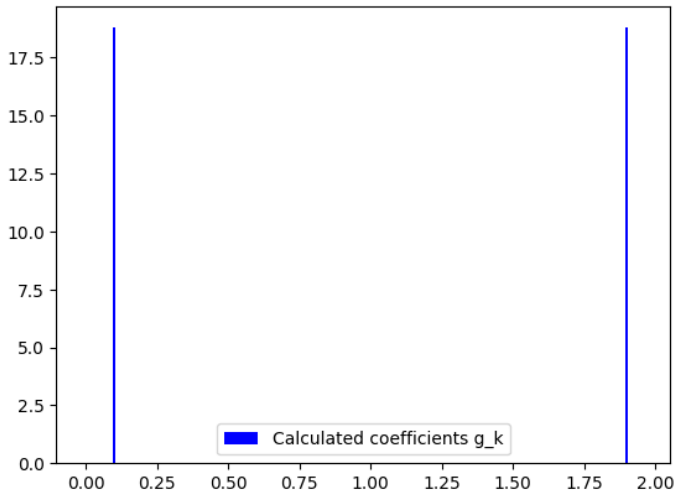


Fig. 1: Calculated coefficients g_k

The result of inverse DFT performed to coefficints g_k is showed on graph below:

Question 4

Now we want to retrieve f from g (deconvolution) using DFT and its inverse.
First, we need to use again the property:

$$DFT(d * f) = DFT(d) \times DFT(f) \Leftrightarrow DFT(g) = DFT(d) \times DFT(f)$$
$$\Leftrightarrow DFT(f) = \frac{DFT(g)}{DFT(d)}$$

As we can see, in the denominator there are the coefficients $DFT(d)$, which can vanish at certain indices. Thus, for these indices, the coefficient $DFT(f)$ will become infinite.

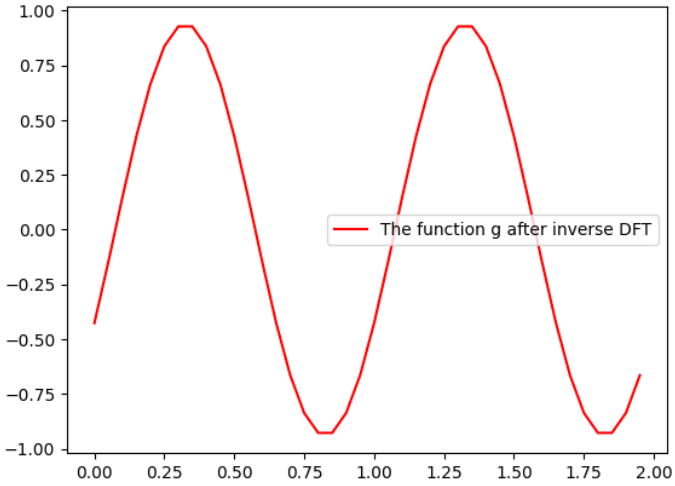


Fig. 2: The function g after inverse DFT

If one of the DFT coefficients $\tilde{g}_k$ is infinite, this can render the function g practically unusable, as its values will either become too large or undefined at many points. Therefore, for physically meaningful computations, the function f needs to be properly defined, avoiding singularities that lead to infinite DFT coefficients.

The corresponding program to calculate the deconvolution of f:

```
# using DFT again for function g and d
G_k = np.fft.fft(g)

D_k = np.fft.fft(d)

# use property of convolution function
F_k = G_k/D_k

# inverse dft to get the values of f
f_dft = np.fft.ifft(F_k)

plt.figure()
plt.plot(x,f,label="sin(2*np.pi*1*x))", color="b")
plt.plot(x, f_dft.real, label="result of DFT", color="r")
plt.legend()
plt.show()
```

We can observe some problems in the corresponding graphs, which were plotted as the result of the deconvolution of the function f.

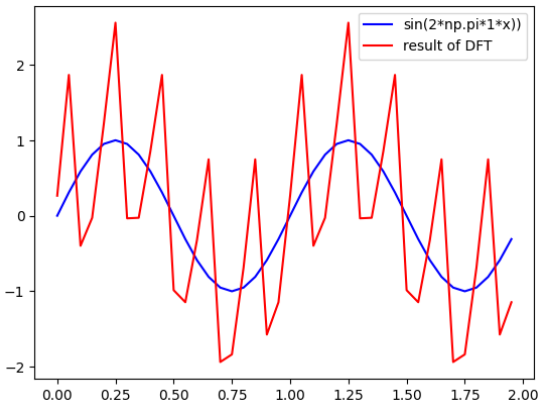


Fig. 3: The function f after using deconvolution without proving coefficients D_k

Question 5

To solve this problem, we should modify the inverse filter as follows:

$$InvD_k = \begin{cases} \frac{1}{D_k}, & \text{if } |D_k| > \epsilon \\ \frac{1}{\epsilon}, & \text{otherwise} \end{cases}$$

This modification helps manage coefficients that vanish for D_k.
The corresponding program:

```
def inverse_Dk(Dk,eps):
  if np.abs(Dk) > eps:
    return 1/Dk
  else:
    return 1/eps
```

The improved result is:

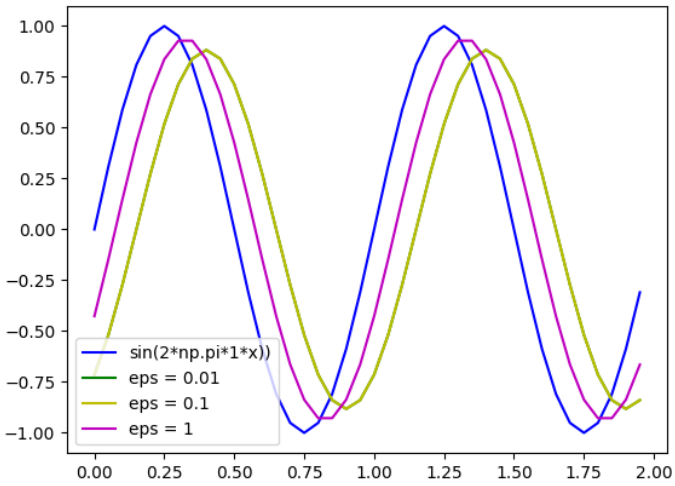


Fig. 4: The function f after using deconvolution inverse coefficients D_k

Now, the function f is smoother and seems to resemble a sinusoid. We observe a shift in the function because we neglect the actual coefficients D_k during the inverse transformation and approximate them using ϵ.