Image and signal processing: Computer exercise 3
CE03G02T05

SID-LAKHDAR Riyane, PRULIERE Valentin

15/11/2015

Abstract

This report summarize, explains and refers to the answers we have designed for the third "Image and Signal Processing" computer exercise.

1 Introduction

The aim of the current computer exercise is to have a first theoretical and experimental approach with two systems often used in signal processing: The Fourier transform (F, and its inverse function F^{-1}) and the convolution.
The idea behind using these systems is to create a function space where the signals have properties which make them easier to study. However, to use a property P on a signal in the output space, we need to explicitly identify how to transcript this property in the input space.

Thus, in the following study, we will try to show the main properties of the output signals of the two systems. We will also try to transcript properties such as periodicity or time scaling from the output space of these systems to properties in them input space. Finally, we will try to consider the time complexity of each system and see how to improve it using the mathematical properties of the two systems.

In this report, we have tried to separate as much as possible our theoretical researches from the output results of our programs, by implementing them regardless of the theoretical expectations.
In this purposes, we have designed the two following sections of this report:

- "Material & Methods": In this section, we described the theoretical methods used to answer to our different questions. We provided the expected algorithms, their main principles or their python implementation.
- "Experiments": In this section, we gave the experimental results of our experiments. We tried to explain them, and see whether they are consistent with the theoretical expectations or not.

2 Material & Methods

2.1 Fourier transform

2.1.1 Definition

The Fourier transform of uni-dimensional function (usually of time) decomposes this input signal into the frequencies that make it up. This mathematical operator maps a function from the real space (depending on time) with a complex function (of frequency). The main interest of this transform is that the absolute value of the transformed function represents the amount of cycles frequencies present in the original signal and complex argument of the transformed function is the phase offset of the basic sinusoid in that frequencies. Thus, analyzing the time and frequency behavior of signals can be made easier by this transform. ([1]).

2.1.2 Formal definition and implementation on discrete time depending signals

For a given discrete uni-dimension signal s containing the N samples s[0], ... s[N-1], the uni-dimensional discrete Fourier transform of s is defined for each input t of s as: $F(s)[t] = \sum_{k=0}^{N-1} s[k] * \exp^{-2i\pi k \frac{t}{N}}$.
To compute the Fourier transform of a discrete time-dependent signal, we have implemented the python function (fig 1). We also have used the same template to implement the inverse function of the Fourier transform (Fig 2)

```python
def myDFT1D(signal):
  length   = len(signal)
  i        = complex(0,1)
  res      = []
  for muInd in range(0, length):
      currRes  = 0
      for xInd in range(0, length):
          currRes  += signal[xInd] *  np.exp(-i * 2 * np.pi * muInd * xInd
               / length)
      res.append(currRes)
  return res
```

Figure 1: Python code to compute Fourier transform of a time depending discrete signal.

The proper functioning of these two functions will be shown in the Experiments section using some input signals and the theoretically known transform (such as an impulse or a Gaussian signal). We will also check if some basic properties of the Fourier transform are respected by our programs (such as $f(t) = FT^{-1}(FT(f(x))))$.

2.1.3 Advantages and properties

Thanks to the uni-dimension Fourier transform previously defined, we are able to:

[1][ref: https://en.wikipedia.org/wiki/Fourier_transform]

```
def myInvDFT1D(ft):
    length  = len(ft)
    i       = complex(0,1)
    res     = []
    for tInd in range(0, length):
        currRes  = 0
        for muInd in range(0, length):
            currRes += ft[muInd] * np.exp(i * 2 * np.pi * tInd * muInd /
                length)
        res.append(currRes / length)
    return res
```

Figure 2: Python code to compute the inverse function of the Fourier transform.

- Make a mapping between a given uni-dimensional signal in the real domain and a function of the complex domain.
- This map exists for any uni-dimensional input signal ([2]) and is unique.

The first interest of this transform is that the transformed function is a complex function. Thus, it has many forms which can be used depending on the studied case:

- A Eulerian form (composed of sinus and a cosinus function) which is more suitable for studying the periodic behavior of our signal over time or frequency. As we know the bounds of the sinus and cosinus functions, this form is also very useful to determine the bounds of our signal's components.
- An exponential form (composed of an amplitude and a phase) which allows to write multiplication signals thanks to simple phase additions.

Secondly, the Fourier transform is a linear application. Thus, study a signal $s(x) = \alpha_0 s_0(x) + \alpha_1 s_1(x)$ through its Fourier transform is equivalent to study the Fourier transform of $s_0 and s_1$ which may be easier. This is the main property we will use to study convolutions in the following sections.

Finally the time (and frequency) property we can notice on the transformed function may be directly transcribed on time (and frequency) properties for the initial real function, using mapping that we will show for specific signals.

2.2 First use of the Fourier transform: discrete impulse signal

2.2.1 Formal expression of the Fourier transform

Let consider the impulse function

$$\delta_a : x \to \delta(x-a) = \begin{cases} 1; & \text{if } x = a \\ 0; & \text{otherwise} \end{cases}$$

[2]Here we suppose that this signal is an integrable functions, or Lebesgue-measurable: $\int_{-\infty}^{\infty} |f(x)| dx < \infty$

Using the definition of the Fourier transform, we have:

$$\begin{aligned}
F(\delta_a)[\omega] = \delta_a[\omega] &= \sum_{k=-\infty}^{+\infty} \delta_a[k] * \exp^{-2i\pi\omega k} \\
&= \sum_{k=-\infty, k\neq a}^{+\infty} \delta_a[k] * \exp^{-2i\pi\omega k} + \delta_a[a] * \exp^{-2i\pi\omega a} \\
&= \exp^{-2i\pi\omega a}
\end{aligned} \tag{1}$$

Using this expression, we can deduce the Fourier transform of the impulse signal using the following forms:
The Eulerian form of the exponential transform: $\delta_a[\omega] = cos(-2i\pi\omega a) + isin(-2i\pi\omega a)$. Thus

$$(Re\Delta_a)[\omega] = cos(-2\pi a\omega) = cos(2\pi a\omega)$$
$$(Im\Delta_a)[\omega] = sin(-2\pi a\omega) = -sin(2\pi a\omega)$$

We can also use the exponential form of the Fourier transform to deduce

$$(Mag\Delta_a)[\omega] = 1$$
$$(Phase\Delta_a)[\omega] = (-2\pi a\omega) mod 2\pi$$

To have an experimental approach of this results, we have implemented the python function given at the figure Fig 3 which generates an impulse signal δ_a. In the next section, we will use this function to experimentally check the previous results.

```python
def generateDelta(IdxSampleWhereImpuls, numberOfSamples):
    res = [0 for j in xrange(numberOfSamples)]
    res[IdxSampleWhereImpuls] = 1
    return res
```

Figure 3: Python code to generate an impulse signal δ_a.

2.2.2 Impact on the decomposition of the signal: periodicity

Using the previous expressions of the Fourier transform, we can notice that the real and the imaginary part of the Fourier transform are periodic with the same period = a. Thus, a time shifting in the input impulse function (changing the value of a) results in a time scaling (frequency change) in the transformed function.

2.3 Convolution and derivative expressions

In the previous section, we have studied how time shifting and time scaling of an input signal may influence the output signal of a system (Fourier transform system). Let's now study an other system: the convolution system(s). The main topic of this section will be to determine the impact of the length of causal signals on them transform. We will also see how and why to use a Fourier transform to compute a convolution and describe a signal. Finally, we will see the limits of this method (in the Experiments section).

2.3.1 Linear and circular convolution definition

The linear and circular convolutions of two causal input function x and y, of length $N_x and N_y$ are defined by:

$$z_{lin}[k] = \sum_{l=0}^{k} x[l] * y[k-l]$$

$$z_{circ}[k] = \sum_{l=0}^{k} x[l] * y[(k-l) mod M]$$

The M that appears in circular convolution's equation will be the parameter that we will variate to show the properties of this convolutions.
To compute the circular convolution of two input signals, we have designed the python program given by the figure Fig 4

```
def my1DCircularConvolution(x, y, M):
    Nx  = len(x)
    Ny  = len(y)
    # Creates a M size copy of x and y where filled with zero at the end
    x_M = copy_and_fill_with_zero(x, M)
    y_M = copy_and_fill_with_zero(y, M)
    z   = []
    for k in xrange(M):
        resK = 0
        for l in xrange(M):
                        resK += x_M[l] * y_M[(k-l)%M]
                z.append(resK)
    return z
```

Figure 4: Python code to generate a circular convolution.

2.3.2 Circular convolution using Fourier transform

The program Fig 4 computes a circular convolution with a complexity $O(N^2)$. But we know that the circular convolution may be computed using the Fourier transform as follows $z_{circ}[k] = F^{-1}(F(x[k])[n]*F(y[k])[n])$. As we know that the Fourier transform of a signal of length N may be computed with a complexity O(N log(N)), we can provide the program figure Fig 5 which will compute a circular convolution in a more efficient way than the one Figure Fig 4

```
def my1DCircularConvolution_fourier(x, y, M):
    # Creates a M size copy of x and y where filled with zero at the end
    x_M  = copy_and_fill_with_zero(x, M)
    y_M  = copy_and_fill_with_zero(y, M)
    TF_x = myDFT1D(x_M)
    TF_y = myDFT1D(y_M)
    return myInvDFT1D(myProd(TF_x, TF_y))
```

Figure 5: Python code to efficiently generate a circular convolution using a Fourier transform.

3 Experiments

3.1 Fourier transform and inverse

In this section, we will consider the programs we designed to compute the Fourier transform and its inverse functions (Fig 1 and Fig 2). We will try to show their proper functioning using some theoretically known properties on the Fourier transform applied on Gaussian function and an impulse function.

3.1.1 Check basic property of the Fourier transform

The first Fourier transform property we will use is: for each uni-dimension signal f and for each input of f, $f(t) = FT^{-1}(FT(f(x)))$.

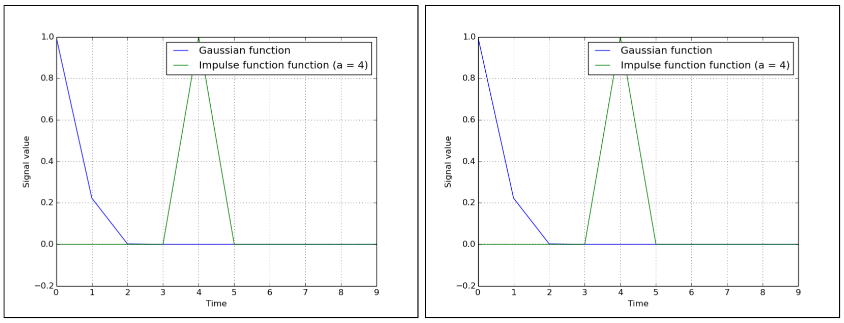

Figure 6: Gaussian and impulse signal on the left. $FT^{-1}(FT(f(x)))$ on the right

We can notice on the figure Fig 6 that our program respect this property for the input signals $f(x) = \exp^{1.5*x^2}$ (Gaussian) and $f(x) = 1$ if $x = 4$ and 0 otherwise.

An other way to check the proper functioning of our program is to uses the theoretically known result: $F(gaussian)(\alpha) = F(\exp^{-A*x^2})(\alpha) = \exp^{\frac{-\pi^2*\alpha^2}{A}} * \sqrt{\frac{\pi}{A}}$ (the proof may be obtained by simply applying the definition of the Fourier transform to the Gaussian function). The figure Fig 7 shows the Fourier transform of a Gaussian function obtained by our program. We can notice on this figure the matching between the Fourier transform computed by our program and the expected function.

3.1.2 Property of the Fourier transform on discrete impulse signals

Using the program defined previously, we will verify some properties on the Fourier transform of an impulse signal.
First of all, the two first plots of the figure Fig 8 represents the Fourier transform of the impulse signal where the non null value is in 0. It confirms that if the non null value of the impulse is in 0, the Fourier transform of the signal is the constant integer 1. By changing the value of the non null index from 0 to 4 or to 8 (four last plots of the figure Fig8), we can also confirm the good behavior

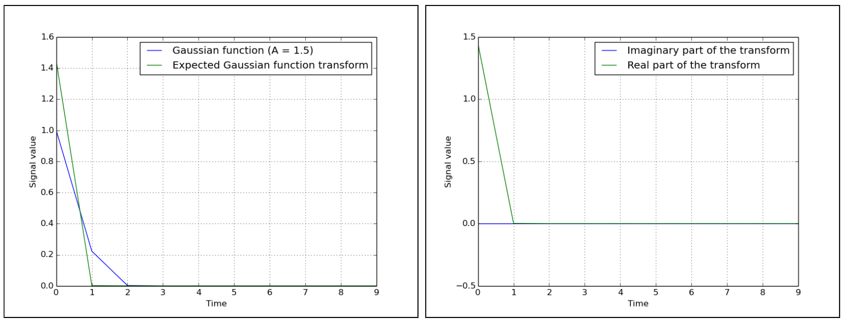

Figure 7: Theoretical and computed transform of a Gaussian.

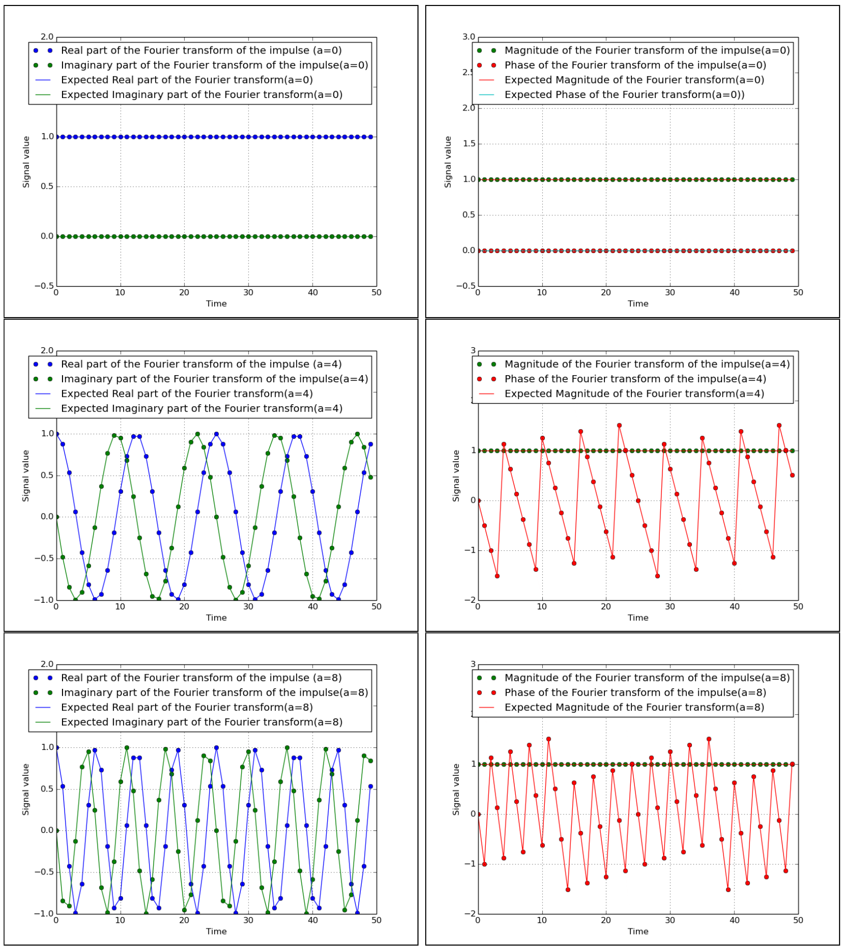

Figure 8: Fourier transform of $\delta_0, \delta_4 and \delta_8$.

of our program by comparing the computed value of the transform (dots) to the theoretically expected values(curves). All this figures also allow us to confirm the periodic property of the Fourier transform of the impulse signal. Finally, this figures allow us to confirm the link between the time shifting of the input impulse signal and the time scaling of the transform signal (the period of the transformed signal equals the shifting a of the initial signal δ_0).

3.2 Convolution and Fourier transform

3.2.1 Use of the output signal length M

Let consider the two signals x = 5,1,2,6,4, and y=1,2,3. Depending on the value of the length M of the output signal, the linear and circular convolution of x and y are:

- $M = N_x + N_y - 1$:

k	0	1	2	3	4	5	6
z_{lin}	5	11	19	13	22	26	12

- $M = N_x + N_y - 1$:

k	0	1	2	3	4	5	6
z_{circ}	5	11	19	13	22	26	12

- $M = max(N_x, N_y)$:

k	0	1	2	3	4
z_{circ}	31	23	19	13	22

3.2.2 Equivalence between Fourier transform and linear convolution

The figure fig 9 (left plot) allows us to see that our experimental results match the results we have calculated in the previous section.

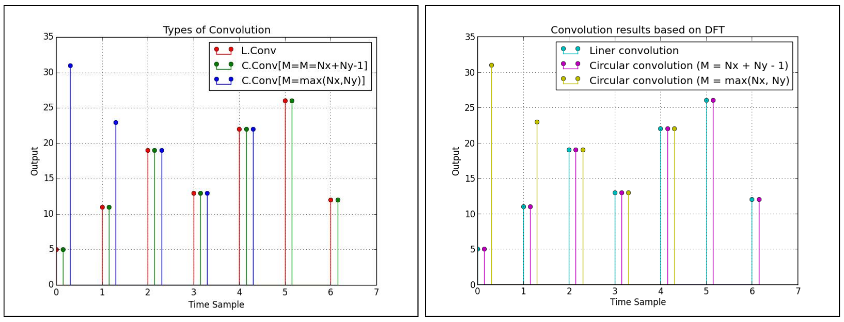

Figure 9: Linear and circular convolution and them Fourier transform.

It also shows that the linear convolution is equivalent to the circular convolution when $M = N_x + N_y - 1$

4 Conclusion

During this computer exercise, we have described two ways to analyze the time and frequency behavior of a signal. Thus, we can now conclude about the advantages of using the Fourier transform for analyzing an impulse signal, and its interesting time complexity compared to the convolution.