Image Restoration by the Inverse Fourier Transform

1 Introduction

In this report, we would first introduce the theoretical elaboration for the Inverse Fourier Transform in terms of image restoration and processing. Then, we would introduce the generalizations and some of the practical applications using Python to explain how Fourier Transform and its inverse works in real life in terms of image processing. Next, we would state disadvantages of Fourier Transform.

2 What is Image Restoration by the Inverse Fourier Transform?

According to K. Acharjya [1], the objective of image restoration is to restore a degraded or distorted image such as image with addictive intensity noise, to its original content and quality.

Refer to V. Hlaváč [2], in terms of image processing, we would input the image and do 2 separate flows. A brief logic flow of image processing would be demonstrated in Figure 1. The first flow would be filtration in spatial filter. It is a linear combination of the input image with coefficients of filter with the operation of convolution. The second flow is that we first do direct transformation (i.e. Fourier Transform in this context). Then, we do filtration in frequency domain (i.e. in the course, we saw different types of filtering such as Gaussian, high-pass, low-pass such and such, they are classified as Frequency Domain.). Next, we do inverse transformation. Finally, we combine these two flows to output the modified image.
From A. Zisserman [3], an observed image can be modelled by linear shift invariant (LSI) equation:

$$g(x,y) = \iint f(x',y')h(x-x',y-y')dx'dy' + n(x,y), \tag{1}$$

where $g(x,y)$ is the spatial domain, integral is convolution, $f(x',y')$ is the original image we would like to restore, $h(x-x',y-y')$ is the convolution or point spread function (PSF), $n(x,y)$ is the addictive noise. In other words, from A. Zisserman [3], we are trying to estimate inputting image f that we would like to expect from the observed degraded image g. The image restoration flow is demonstrated in Figure 2. According to CosmoStat [4], PSF describes how an imaging system responds to an unresolved point source. In other words, the PSF gives a measure of the amount of blurring that is added to any given object as a result of imperfections in the optics.

2.1 Fourier Transform for Convolution and Inverse Fourier Transform

Next, we would apply Fourier Transform for Convolution into LSI such that Inverse Fourier Transform exists. With reference to J. M. Buhmann [5], proof of Fourier Transform for Convolution combined with LSI are as follows:
Suppose that F is Fourier Transform and ignore the component $n(x,y)$, which is a limitation of Fourier Transform and would be discussed in session 4. Given that convolution is

$$g(x,y) = (f*h)(x,y) = \iint f(x',y')h(x-x',y-y')dx'dy'. \tag{2}$$

We calculate Fourier Transform of $g(\hat{x})$:

$$\begin{aligned} g(x,y) = F[g(\hat{x,y})] &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x',y')h(x-x',y-y')\exp{(-i2\pi ux)}\exp{(-i2\pi uy)}dx'dy' \\ &= \int_{-\infty}^{\infty} f(x',y')\int_{-\infty}^{\infty} h(x-x',y-y')\exp{(-i2\pi ux)}\exp{(-i2\pi uy)}dx'dy' \\ &= \int_{-\infty}^{\infty} \hat{h}(u)f(x',y')\exp{(-i2\pi ux')}\exp{(-i2\pi uy')}dx'dy' \\ &= \hat{h}(u)\hat{f}(u) \end{aligned}$$

, where $\hat{f}(u)$ is the Inverse Fourier Transform $f(x',y')$. Hence we have equation:

$$g(x,y) = \hat{h}(u)\hat{f}(u) \tag{3}$$

Therefore, we can apply the Fourier Transform with convolution to to obtain the image we would like to restore with Inverse Fourier Transform.

3 Generalizations and Application

In this section, we would like to introduce two applied methods for the theoretical Fourier Transform and its inverse to attain image restoration of the degraded images in the real world. The two applied methods are Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT).

3.1 Discrete Fourier Transform (DFT)

Refer to notes from Stephen Roberts [6], the Discrete Fourier Transform (DFT) is the equivalent of the continuous Fourier Transform for signals known only at instants separated by sample times. (i.e. a finite sequence of data).
The Discrete Fourier Transform is modelled as follows from Stephen Roberts [5]:
Let $f(n)$ be an input signal, which is the source of data, where $n = 0, ..., N-1$. Let N samples be denoted $f[0], f[1], f[2], ..., f[k], ..., f[N-1]$. Then the Discrete Fourier Transform is:

$$F[n] = \frac{1}{N}\sum_{k=0}^{N-1} f[k]\exp{(-\frac{2\pi ink}{N})}, \tag{4}$$

and the inverse Discrete Fourier Transform is:

$$f[k] = \frac{1}{N}\sum_{n=1}^{N-1} F[n]\exp{(\frac{2\pi ink}{N})}. \tag{5}$$

3.2 Fast Fourier Transform (FFT)

Refer to S. Roberts [6], since the time taken to evaluate a DFT on a digital computer depends principally on the number of multiplications involved, it is time-consuming to take operations for these actions and hence we would introduce a more efficient algorithm to perform Fourier Transform, which is the Fast Fourier Transform (FFT). Refer to V. Hlaváč [2], the main idea of FFT is that the length N in DFT can be expressed as sum of length $\frac{N}{2}$ of two DFTs, the first one consists of

odd samples and the second one contains even samples.
According to S. Roberts [6], Fast Fourier Transform is:

$$F[n] = \frac{1}{N} \sum_{k=0}^{N-1} f[k] W_N^{nk}, \tag{6}$$

where $nk \in \mathbb{Z}$ repeats from various combinations of k and n, W_N^{nk} is a periodic function with only distinct N values.

3.3 Examples and process flow for image restoration with FFT

Now, we would examine some examples using Python with reference to C. Chen [7], combining with low-pass filter, and high-pass filter , that we have learnt in our course, to perform FFT with `np.fft` package and use `cv2` package to load the picture.
We would examine `'poppy.jpg'` in Figure 3 from course worksheet as an example for the low-pass and high-pass filters using FFT with reference to pictures in Figure 4 and 5 respectively.
First, we would perform FFT to transform image to the frequency. Next, we would visualize and centralize the zero-frequency component, then apply filter frequencies (low-pass and high-pass), and then decentralize, and finally use Inverse Fourier Transform to generate data and obtain the modified image. Python code is demonstrated in Appendix 1.

3.3.1 Low-Pass filter and High-Pass filter

From T. C. O'Haver [9], low-pass filter is a filter that does not affect low frequency components to pass through but does minimize or eliminate for high frequency components and is usually used for smoothing and highlight low gradient areas to have a blurring or smoothing edge effects in the image. high-pass filter is a filter that allows high frequency components to pass through but minimize or eliminate low frequency components and is usually used for sharpening and highlight high gradient areas such as edges.

4 Limitation

Earlier, we do assume that to ignore the addictive noise $n(x, y)$ in Fourier Transformation, and this is one of the drawbacks for Fourier Transformation as it does not process addictive noise. In addition, the Fourier Transformation would notify us when there is a discontinuity but it does not tell us where the discontinuity is. Furthermore, according to tutorialspoint [8], Fourier transform is only applicable to periodic signals and that is not applicable when signal is non-periodic nor aperiodic.

5 Appendix

5.1 Python code with reference to C. Chen [7]

```
# import packages for picture loading, data processing and graph plotting
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt,exp
import cv2
import facets_improc as ip

# Define distance function to locate the points in the frequency domain
def distance(point1,point2):
    return sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)

# Define Low Pass filter function
def idealFilterLP(D0,imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2,cols/2)
    for x in range(cols):
        for y in range(rows):
            if distance((y,x),center) < D0:
                base[y,x] = 1
    return base
# Define High Pass filter function
def idealFilterHP(D0,imgShape):
    base = np.ones(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2,cols/2)
    for x in range(cols):
        for y in range(rows):
            if distance((y,x),center) < D0:
                base[y,x] = 0
    return base

# Begin FFT with low-pass filter
plt.figure(figsize=(7*5, 5*5), constrained_layout=False)
# Load the original picture
img = cv2.imread("poppy.jpg", 0)
plt.subplot(131), plt.imshow(img, "gray"), plt.title("Original Image")
# Perform FFT
original = np.fft.fft2(img)
plt.subplot(132), plt.imshow(np.log(1+np.abs(original)), "gray"), plt.title("Spectrum")
# Perform centralization
center = np.fft.fftshift(original)
plt.subplot(133), plt.imshow(np.log(1+np.abs(center)), "gray"), plt.title("Centered Spectrum")
```

```
# plot the graph
plt.gcf()
# save the modified image
plt.savefig('LP_poppy1.jpg')

plt.figure(figsize=(7*5, 5*5), constrained_layout=False)
# Perform centralization with Low Pass filter
LowPassCenter = center * idealFilterLP(50,img.shape)
plt.subplot(231), plt.imshow(np.log(1+np.abs(LowPassCenter)), "gray"),
plt.title("Centered Spectrum multiply Low Pass Filter")
# Perform decentralization
LowPass = np.fft.ifftshift(LowPassCenter)
plt.subplot(232), plt.imshow(np.log(1+np.abs(LowPass)), "gray"), plt.title("Decentralize")
# Perfrom inverse FFT to obtain the modified image
inverse_LowPass = np.fft.ifft2(LowPass)
plt.subplot(233), plt.imshow(np.abs(inverse_LowPass), "gray"), plt.title("Processed Image")

# plot the graph
plt.gcf()
# save the modified image
plt.savefig('LP_poppy2.jpg')

# Begin FFT with High-pass filter
plt.figure(figsize=(7*5, 5*5), constrained_layout=False)
# Load the original image
img = cv2.imread("poppy.jpg", 0)
plt.subplot(131), plt.imshow(img, "gray"), plt.title("Original Image")
# Perform FFT
original = np.fft.fft2(img)
plt.subplot(132), plt.imshow(np.log(1+np.abs(original)), "gray"), plt.title("Spectrum")
# Perform centralization
center = np.fft.fftshift(original)
plt.subplot(133), plt.imshow(np.log(1+np.abs(center)), "gray"), plt.title("Centered Spectrum")

# plot the graph
plt.gcf()
# save the modified image
plt.savefig('HP_poppy1.jpg')

# Perform centralization with Low Pass filter
plt.figure(figsize=(7*5, 5*5), constrained_layout=False)
HighPassCenter = center * idealFilterHP(50,img.shape)
plt.subplot(231), plt.imshow(np.log(1+np.abs(HighPassCenter)), "gray"),
plt.title("Centered Spectrum multiply Low Pass Filter")
# Perform decentralization
HighPass = np.fft.ifftshift(HighPassCenter)
plt.subplot(232), plt.imshow(np.log(1+np.abs(HighPass)), "gray"), plt.title("Decentralize")
# Perfrom inverse FFT to obtain the modified image
```

```
inverse_HighPass = np.fft.ifft2(HighPass)
plt.subplot(233), plt.imshow(np.abs(inverse_HighPass), "gray"), plt.title("Processed Image")

# plot the graph
plt.gcf()
# save the modified image
plt.savefig('HP_poppy2.jpg')
```

5.2 Figures

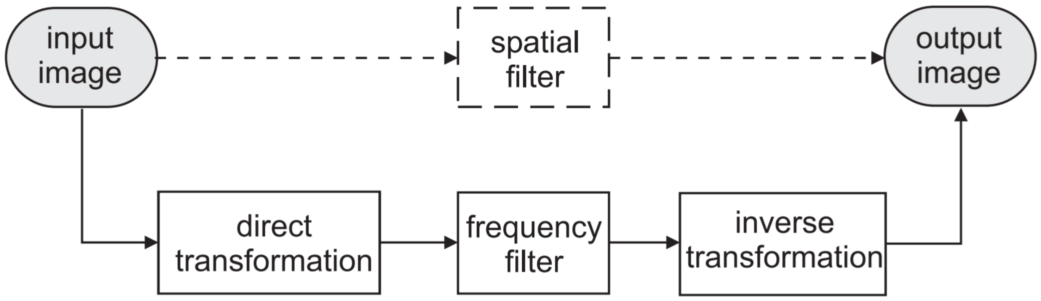


Figure 1: Flow of Image Processing and filtering from V. Hlaváč [2]

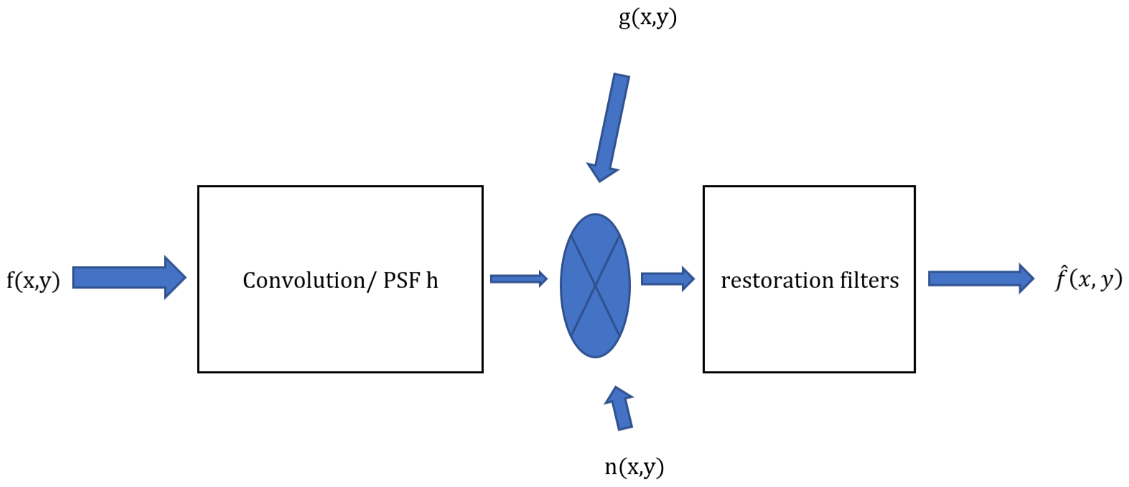


Figure 2: Flow of Image Processing and filtering from V. Hlaváč [2]

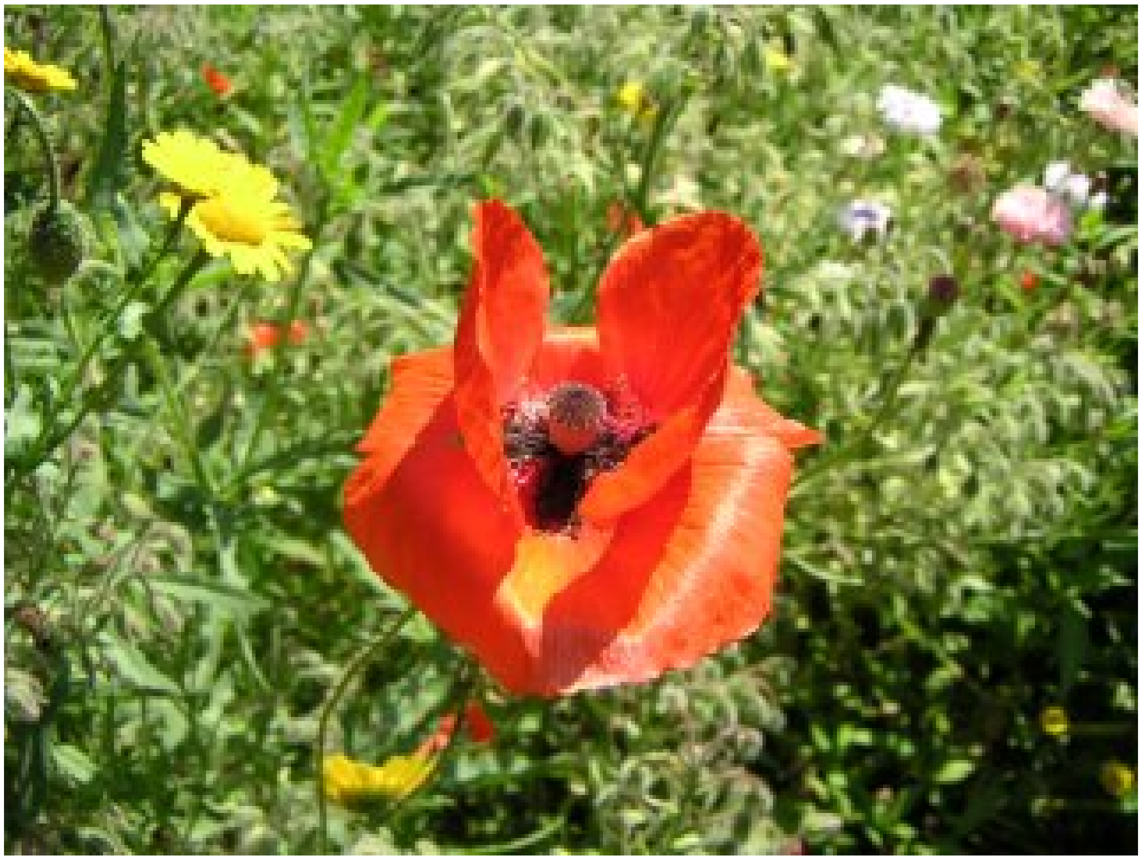
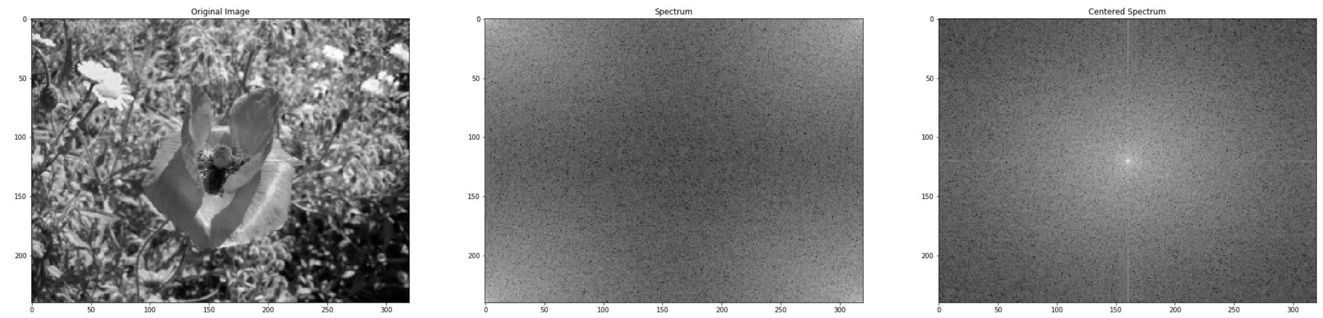
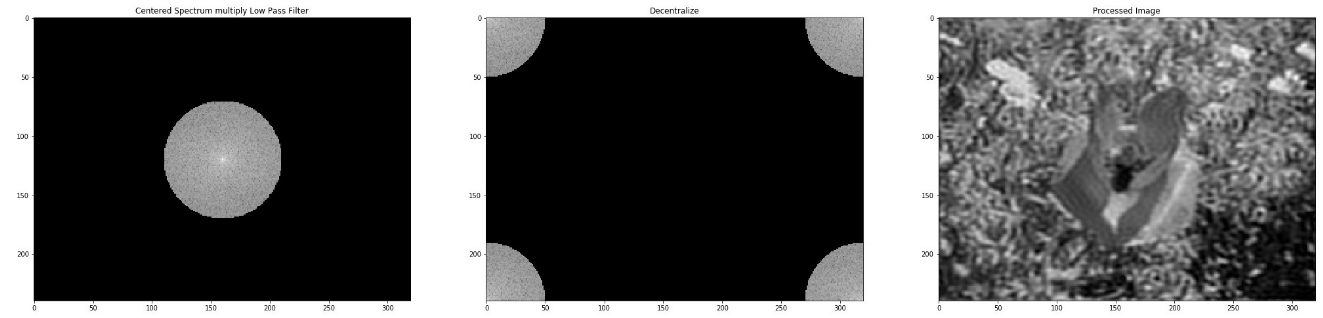


Figure 3: Original image

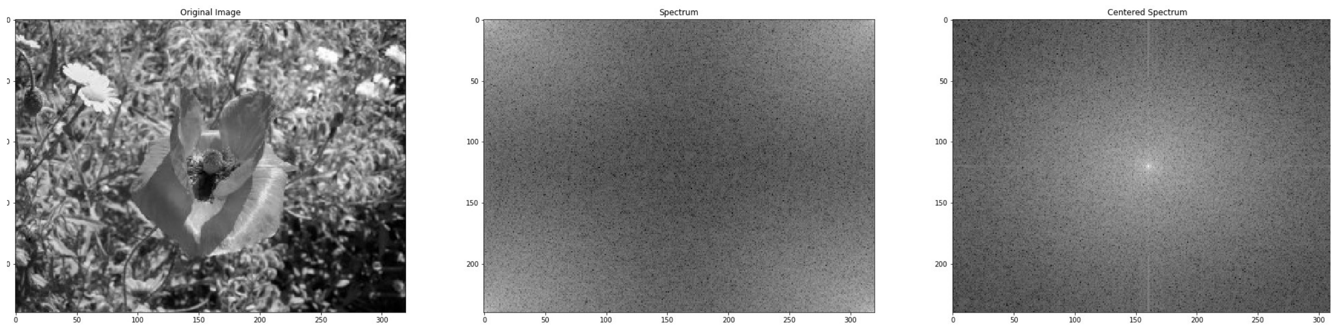


(a) (From left to right) (1) Original image, (2) Spectrum of FFT, (3) Centralized Spectrum

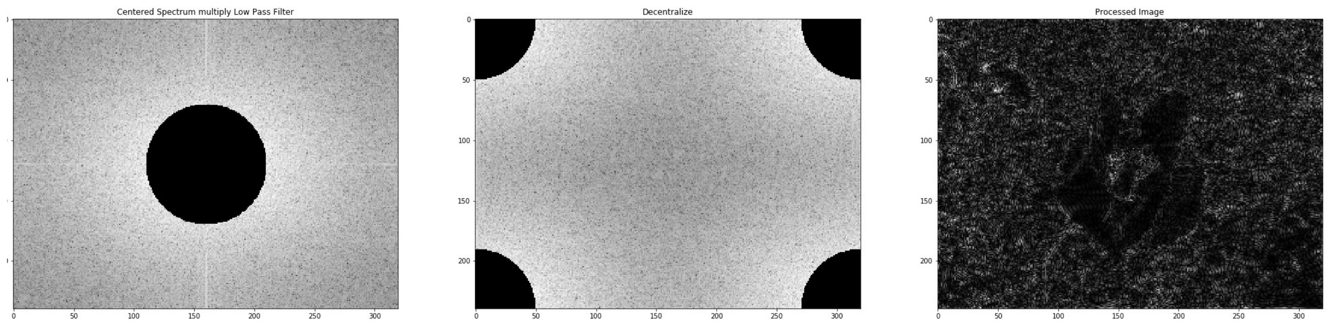


(b) (From left to right) (1) Centralized Spectrum with Low Pass Filter, (2) Decentralization, (3) Processed image with inverse FFT

Figure 4: Fast Fourier Transform with Low Pass Filter [7]



(a) (From left to right) (1) Original image, (2) Spectrum of FFT, (3) Centralized Spectrum



(b) (From left to right) (1) Centralized Spectrum with High Pass Filter, (2) Decentralization, (3) Processed image with inverse FFT

Figure 5: Fast Fourier Transform with High Pass Filter [7]

References

[1] K. Acharjya, *Image Restoration*, https://bit.ly/34dendc/
[Accessed 19/08/2020.]

[2] V. Hlaváč, *Fourier transform in 1D and in 2D*,
http://www.robots.ox.ac.uk/ sjrob/Teaching/SP/l7.pdf/
[Accessed 19/08/2020.]

[3] A. Zisserman, *Image Restoration*,
http://www.robots.ox.ac.uk/ az/lectures/ia/lect3.pdf/
[Accessed 19/08/2020.]

[4] CosmoStat, *PSF Estimation and Image Restoration*, https://www.cosmostat.org/research-topics/psf/
[Accessed 19/08/2020.]

[5] J. M. Buhmann, *Image Processing and Computer Vision*,
http://people.inf.ethz.ch/pomarc/courses/VisualComputing/viscompMP01.pdf/
[Accessed 19/08/2020.]

[6] S. Roberts, *The Discrete Fourier Transform*, http://www.robots.ox.ac.uk/ sjrob/Teaching/SP/l7.pdf/
[Accessed 19/08/2020.]

[7] C. Chen, *Digital Image Processing using Fourier Transform in Python*,
https://bit.ly/324Dzj7/
[Accessed 19/08/2020.]

[8] Tutorialspoint, *Fourier transforms*, https://bit.ly/3g7Wtuw
[Accessed 19/08/2020.]

[9] T. C. O'Haver, *A Pragmatic Introduction to Signal Processing*,
https://terpconnect.umd.edu/ toh/spectrum/Smoothing.html/
[Accessed 19/08/2020.]