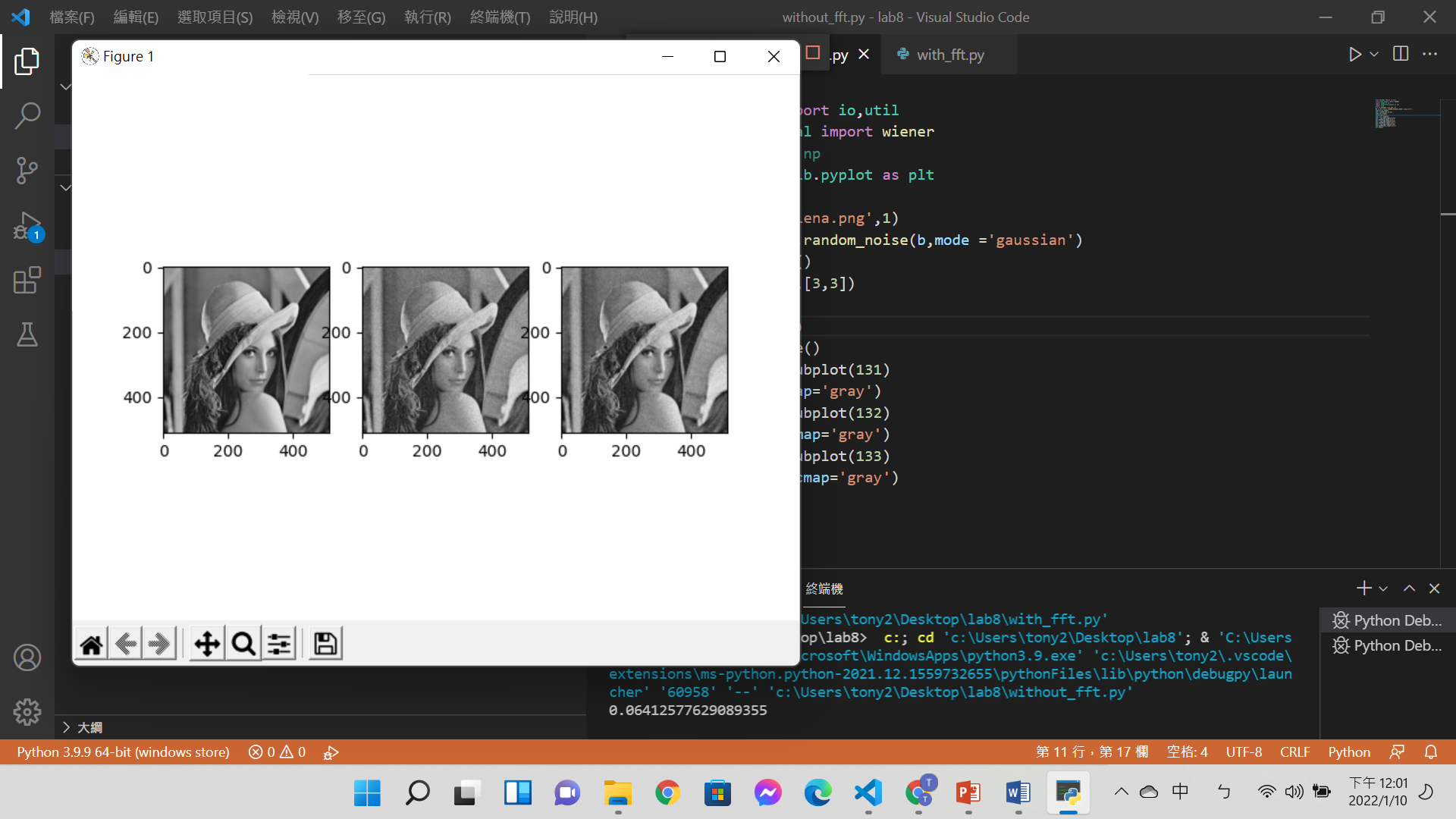
* + 製作一Python程式，不使用FFT製作濾波器

花費0.06412577629089355秒



Code:

from skimage import io,util

from scipy.signal import wiener

import numpy as np

import matplotlib.pyplot as plt

import time

b = io.imread('Lena.png',1)

bn = util.noise.random\_noise(b,mode ='gaussian')

start=time.time()

brn = wiener(bn,[3,3])

end=time.time()

print(end-start)

fig = plt.figure()

ax1 = fig.add\_subplot(131)

ax1.imshow(b,cmap='gray')

ax2 = fig.add\_subplot(132)

ax2.imshow(bn,cmap='gray')

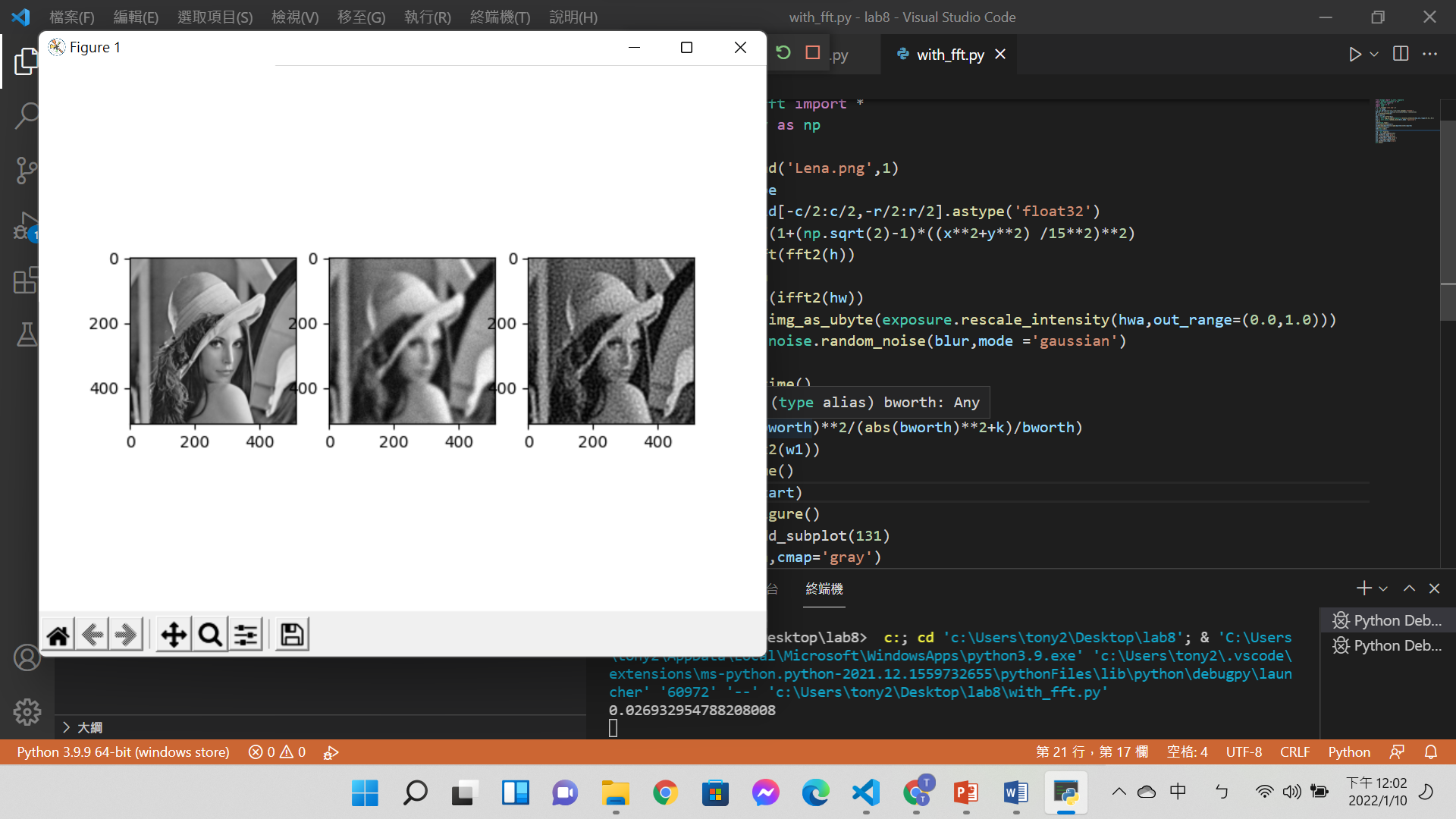
ax3 = fig.add\_subplot(133)

ax3.imshow(brn,cmap='gray')

plt.show()

* + 製作一Python程式，使用FFT製作濾波器

花費時間0.026932954788208008秒



Code:  
from skimage import io,util, exposure

import matplotlib.pyplot as plt

from numpy.fft import \*

import numpy as np

import time

h = io.imread('Lena.png',1)

r,c = h.shape

x,y =np.mgrid[-c/2:c/2,-r/2:r/2].astype('float32')

bworth = 1./(1+(np.sqrt(2)-1)\*((x\*\*2+y\*\*2) /15\*\*2)\*\*2)

hf = fftshift(fft2(h))

hw=hf\*bworth

hwa = np.abs(ifft2(hw))

blur = util.img\_as\_ubyte(exposure.rescale\_intensity(hwa,out\_range=(0.0,1.0)))

blur = util.noise.random\_noise(blur,mode ='gaussian')

k=0.01

start=time.time()

bf=fftshift(fft2(blur))

w1=bf\*(abs(bworth)\*\*2/(abs(bworth)\*\*2+k)/bworth)

wla=abs(ifft2(w1))

end=time.time()

print(end-start)

fig = plt.figure()

ax3 = fig.add\_subplot(131)

ax3.imshow(h,cmap='gray')

ax1 = fig.add\_subplot(132)

ax1.imshow(blur,cmap='gray')

ax2 = fig.add\_subplot(133)

ax2.imshow(wla,cmap='gray')

plt.show()

* + 比較上述兩種做法的執行結果及執行速度

從以上的結果來看，使用傅立葉轉換復原影像是比用傳統方式的濾波器快上很多的。

心得

僅管我在使用fft製作濾波器的實習項目中先用butterworth 濾波器之後再使用高斯雜訊，但復原結果還是不錯，且使用fft濾波器還是比傳統濾波器快上許多