Lab 2: 2-D Random Processes

Course Title: Image Processing I (Spring 2022)

Course Number: ECE 63700

Instructor: Prof. Charles A. Bouman

Author: **Zhankun Luo**

1. Power Spectral Density of an Image

1.1. gray scale image `img04g.tif`

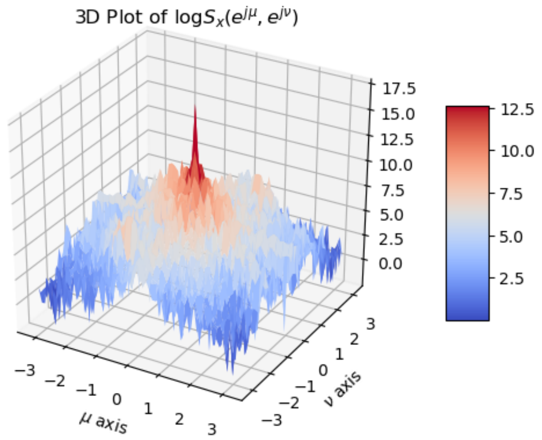
solution

Input Gray Scale Image **img04g.tif**

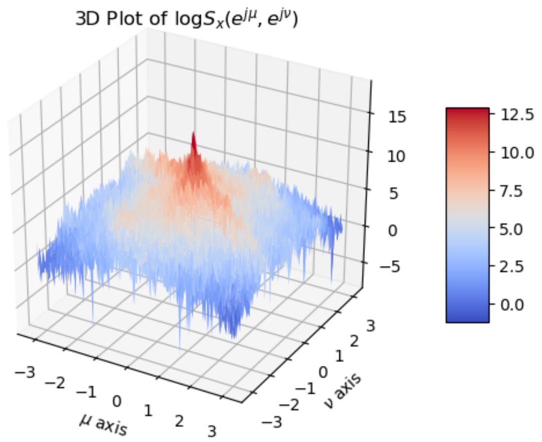
1.2. power spectral density for block 64×64, 128×128, 256×256

solution

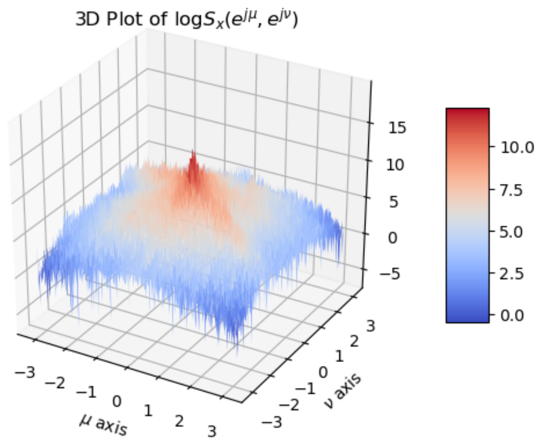
The mesh 3D plots for block sizes of 64×64, 128×128, and 256×256 are shown below



Power Spectral Density for Block Sizes of 64×64



Power Spectral Density for Block Sizes of 128×128

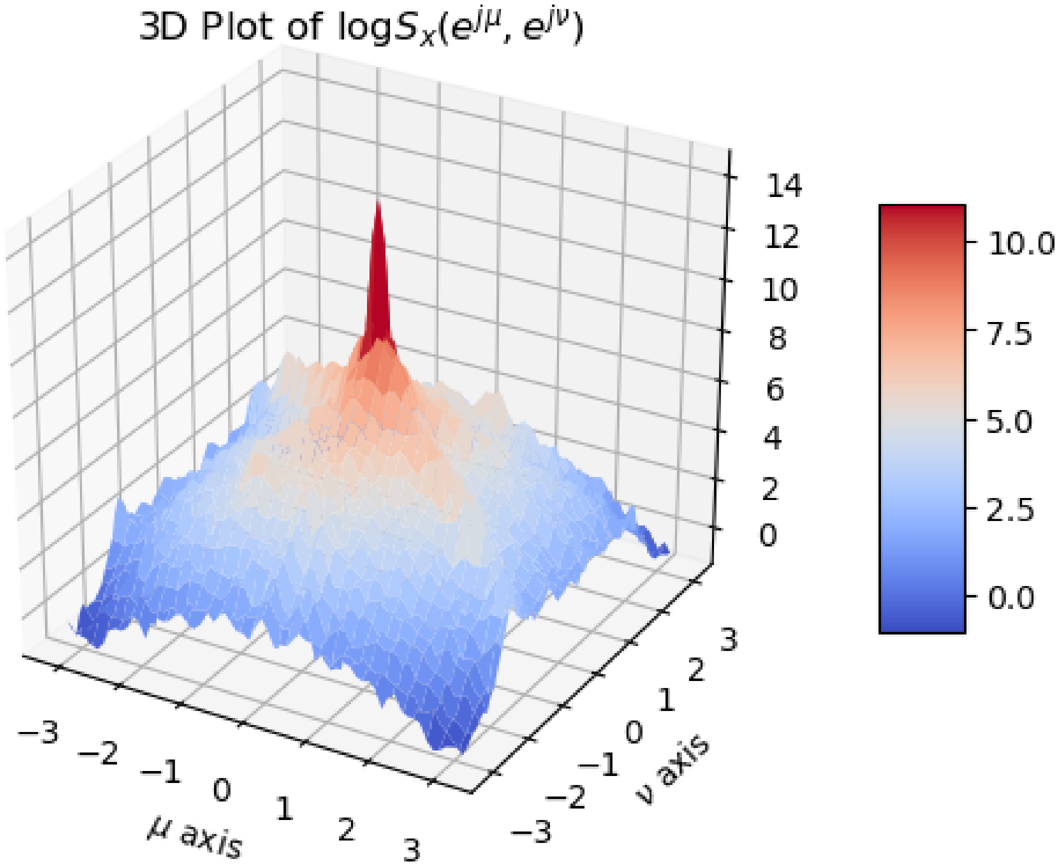


Power Spectral Density for Block Sizes of 256×256

1.3. improved power spectral density estimate

solution

The improved power spectral density estimate with 25 non-overlapping image windows of size 64 ×64 is shown below



Better Power Spectral Density with 25 Non-overlapping Windows of Size 64x64

3.5. Python function: `BetterSpecAnal(x)`

solution

The function `BetterSpecAnal(x)` can compute the improved power spectral density estimate with 25 non-overlapping image windows, and export the mesh plot

```
def BetterSpecAnal(x: np.ndarray,
                   size_block: int = 64,
                   num_window: int = 5,
                   symbol='\log S_x') \
    -> Tuple[np.ndarray, Figure]:
    H, W = x.shape
    h_center, w_center, width_block, width_id \
        = H//2, W//2, size_block//2, num_window//2
    h_corner0, w_corner0 \
        = h_center - width_block, w_center - width_block
    list_id = list(range(-width_id,  -width_id+num_window))
    list_pair_offset = list((i*size_block,j*size_block) \
                            for i in list_id for j in list_id)
    window = hamming(size_block).reshape(-1, 1)
    window = window @ window.T
    Z = np.zeros((size_block, size_block))
    for dh, dw in list_pair_offset:
        h_corner, w_corner = h_corner0 + dh, w_corner0 + dw
        z = window * x[h_corner: h_corner+size_block, \
                       w_corner: w_corner+size_block]
        # Compute the power spectrum for the region
        Z += square(abs(fft2(z)) / size_block)
    Z /= len(list_pair_offset)
    # Use fftshift to move the zero frequencies to the center
    Z = fftshift(Z)
    # Compute the logarithm of Power Spectrum
    log_Z = log(Z)
    # Plot 3-D mesh plot and label x and y axises
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x = np.linspace(-pi, pi, num=size_block)
    X, Y = np.meshgrid(x, x)
    cp = ax.plot_surface(X, Y, log_Z, cmap=plt.cm.coolwarm)
    ax.set_xlabel(r'$\mu$ axis')
```

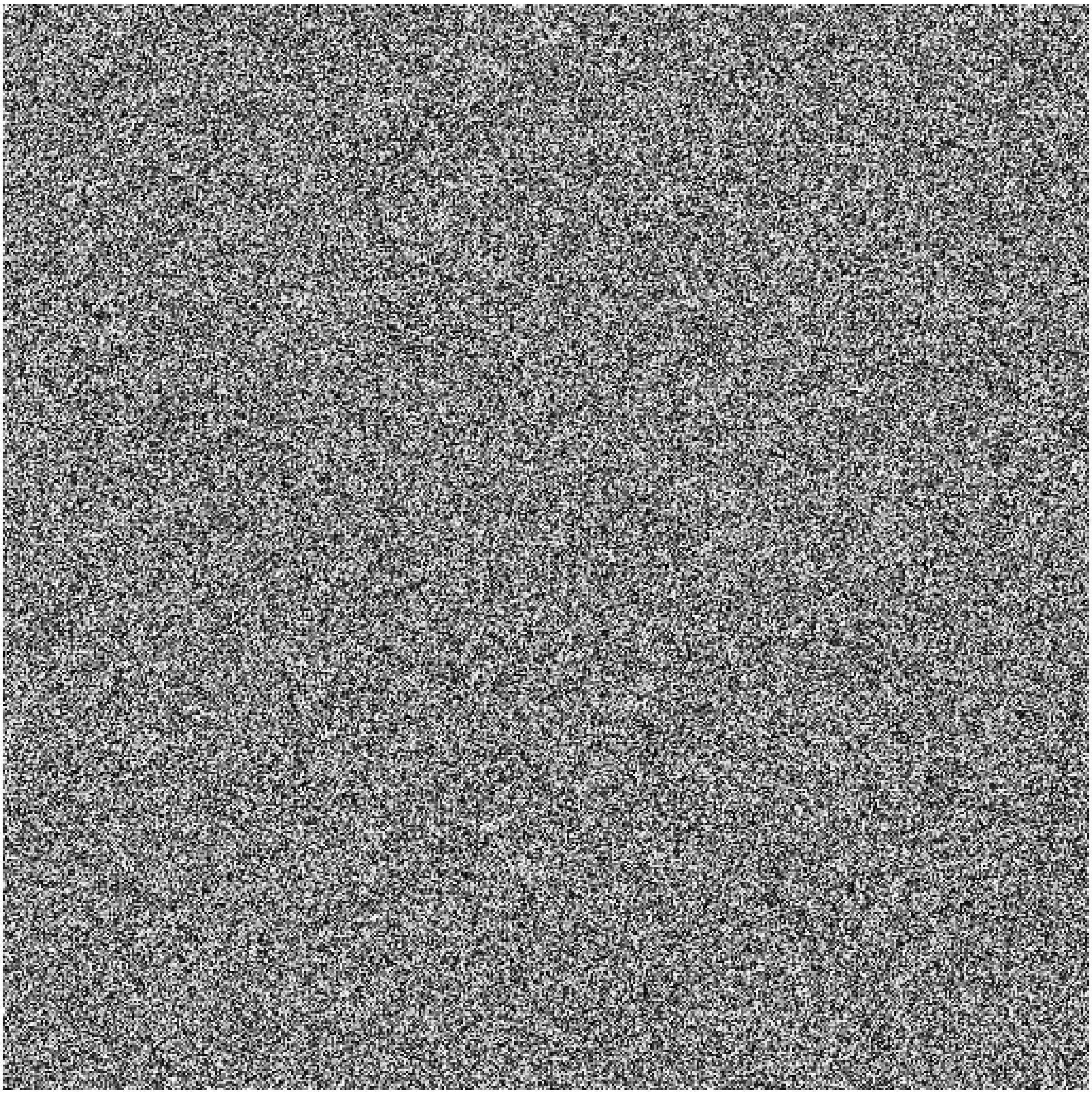
```
    ax.set_ylabel(r'$\nu$ axis')
    str_title = r'3D Plot of $' + symbol \
        + r'(e^{j \mu}, e^{j \nu})$'
    ax.set_title(str_title)
    fig.colorbar(cp, shrink=0.5, aspect=5)
    return Z, fig
```

2. Power Spectral Density of a 2-D AR Process

2.1. image `255(x + 0.5)`

solution

The scaled image `x_scaled=255*(x+0.5)` is shown below



Scaled Image 255(x+0.5)

2.2. image `y + 127`

solution

We filter the image `x` to produce the image `y` using an IIR low-pass filter with transfer function

$$H\left(z_1, z_2\right) = \frac{3}{1 - 0.99 z_1^{-1} - 0.99 z_2^{-1} + 0.9801 z_1^{-1} z_2^{-1}}$$

The corresponding difference function is

$$y(m,n) = 3x(m,n) + 0.99(y(m-1,n) + y(m,n-1)) - 0.9801 y(m-1,n-1)$$

The image $y + 127$ is shown below

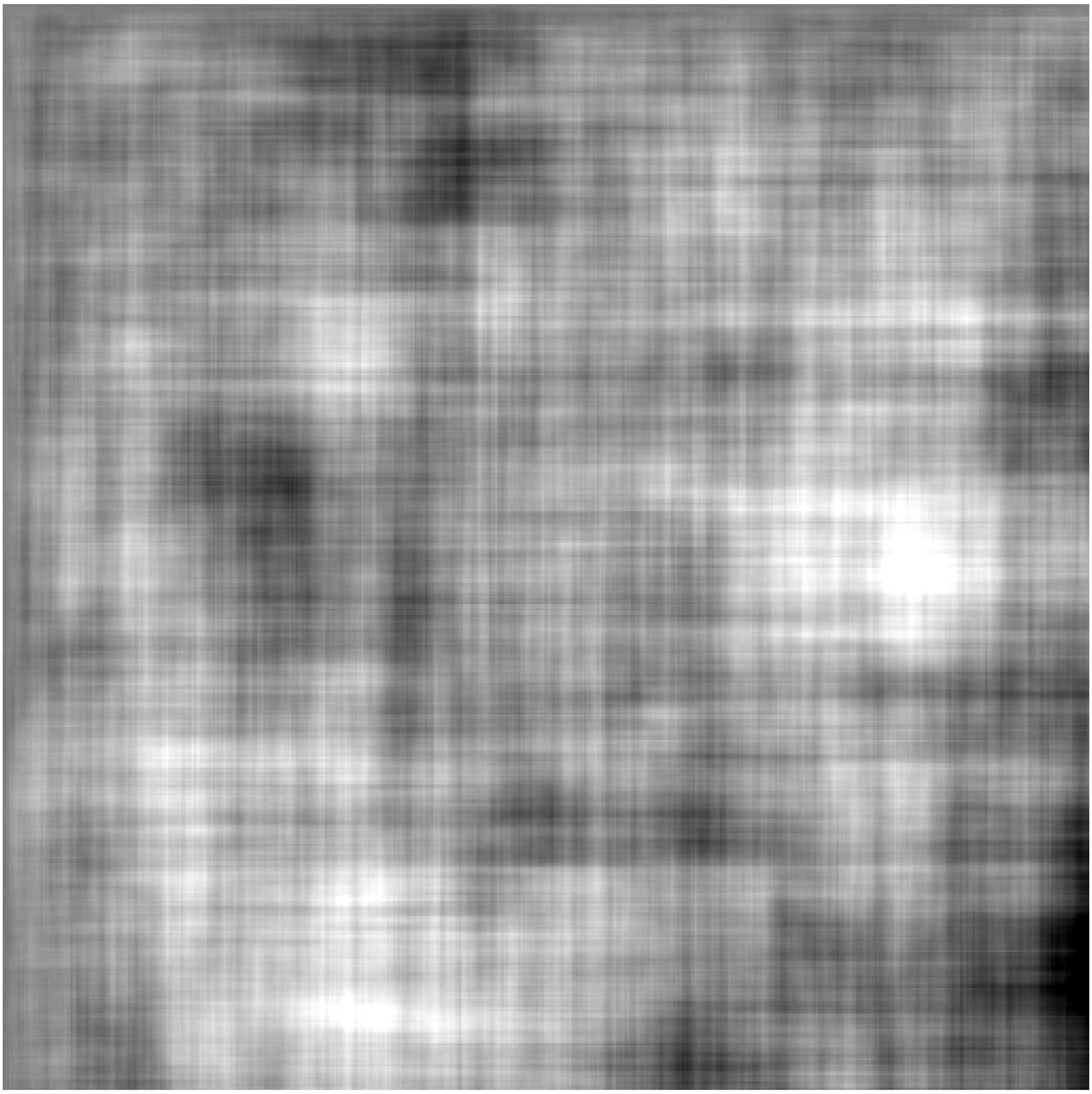


Image y+127 with IIR Low-pass Filter

2.3. mesh plot of the function $\log S_y\left(e^{j\mu}, e^{j\nu}\right)$

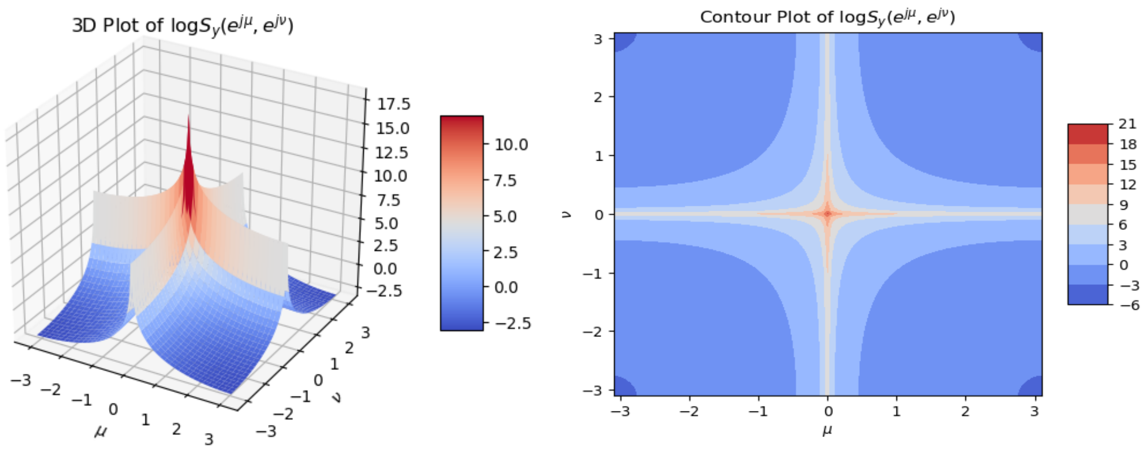
solution

The theoretical power spectrum for `y` is, if we treat `x` as a "white noise"

$$S_y = \left|H\left(e^{j\mu}, e^{j\nu}\right)\right|^2 S_x, \quad \text{where } S_x \approx \mathbb{E}[x^2] = \frac{0.5^2}{3} = \frac{1}{12}$$

$$\log S_y \mid_{(\mu,\nu)=(0,0)} = 2 \log H \mid_{(\mu,\nu)=(0,0)} + \log S_x = 2 \times \log \frac{3}{(1-0.99)^2} + \log \frac{1}{12} = 18.13$$

The left figure below is a 3D mesh plot for the theoretical log power spectrum of `y` $\log S_y\left(e^{j\mu}, e^{j\nu}\right)$, the right figure is a contour plot for $\log S_y\left(e^{j\mu}, e^{j\nu}\right)$



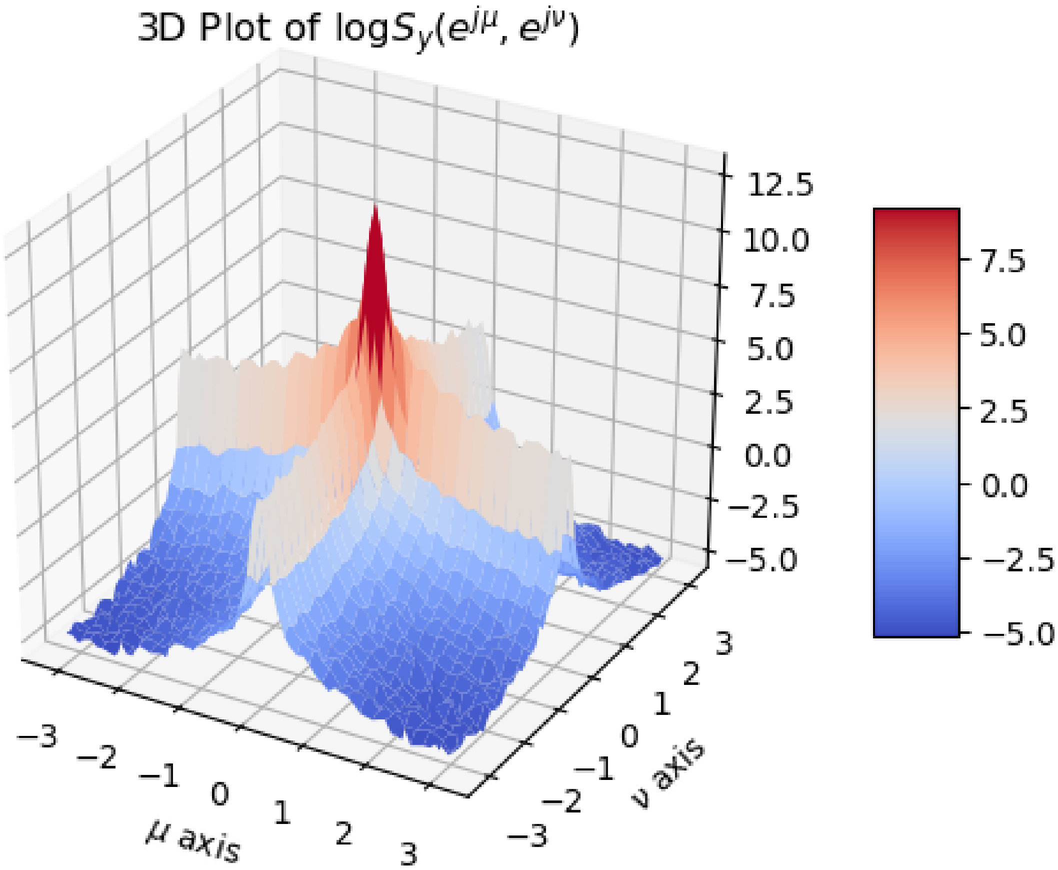
3D Mesh(left) and Contour(right) Plots for Theoretical Log Power Spectrum of y

2.4. mesh plot of the log of estimated power spectral density of `y` using `BetterSpecAnal(y)`

solution

We generate the experimental mesh plot of the log of estimated power spectral density of `y` using `BetterSpecAnal(y)`

The left figure below is a 3D mesh plot for the experimental log power spectrum of `y` $\log S_y\left(e^{j\mu}, e^{j\nu}\right)$, the right figure is a contour plot for $\log S_y\left(e^{j\mu}, e^{j\nu}\right)$



3D Mesh(left) and Contour(right) Plots for Experimental Log Power Spectrum of y

2.5. Python function: `filter_IIR_lowpass(x)`

solution

The function `filter_IIR_lowpass(x)` for IIR low pass filter $H(z_1, z_2)$

```
def filter_IIR_lowpass(x: np.ndarray, pole: float,
                       amplitude: float=1) -> np.ndarray:
    H, W = x.shape
    a00, b01, b11 = square(amplitude*(1-pole)), -pole, pole*pole
    reg, y = np.zeros(W), np.zeros(x.shape)
    y[0][0] = a00 * x[0][0]
    for col in range(1, W):
        y[0, col] = a00 * x[0, col] - b01 * y[0, col-1]
    for row in range(1, H):
        y[row, 0] = a00 * x[row, 0] - b01 * y[row-1, 0]
    for row in range(1, H):
        reg[1:] = b01 * y[row-1, 1:] + b11 * y[row-1, :-1]
        for col in range(1, W):
            y[row, col] = a00 * x[row, col]
                - b01 * y[row, col-1] - reg[col]
    return y
```

Appendix

Code to compute power spectrum $S\left(e^{j\mu}, e^{j\nu}\right)$: `spec_anal.py`

spec_anal.py

```
import numpy as np
from numpy import abs, square, pi, log, hamming
from numpy.fft import fft2, fftshift
import matplotlib.pyplot as plt
from  matplotlib.figure import Figure
from typing import Tuple

def SpecAnal(x: np.ndarray,
             size_block: int = 64,
             symbol='\log S_x')
    -> Tuple[np.ndarray, Figure]:
    H, W = x.shape
    h_center, w_center, width_block \
        = H//2, W//2, size_block//2
    h_corner, w_corner \
        = h_center - width_block, w_center - width_block
    z = x[h_corner: h_corner+size_block,
          w_corner: w_corner+size_block]
    # Compute the power spectrum for the NxN region.
    Z = square(abs(fft2(z)) / size_block)
    # Use fftshift to move the zero frequencies to the center
    Z = fftshift(Z)
    log_Z = log(Z)
    # Plot 3-D mesh plot and label the x and y
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x = np.linspace(-pi, pi, num=size_block)
    X, Y = np.meshgrid(x, x)
    cp = ax.plot_surface(X, Y, log_Z, cmap=plt.cm.coolwarm)
    ax.set_xlabel(r'$\mu$ axis')
    ax.set_ylabel(r'$\nu$ axis')
    str_title = r'3D Plot of $' + symbol \
        + r'(e^{j \mu}, e^{j \nu})$'
    ax.set_title(str_title)
```

```
    fig.colorbar(cp, shrink=0.5, aspect=5)
    return Z, fig

def BetterSpecAnal(x: np.ndarray,
                   size_block: int = 64,
                   num_window: int = 5,
                   symbol='\log S_x') \
    -> Tuple[np.ndarray, Figure]:
    H, W = x.shape
    h_center, w_center, width_block, width_id \
        = H//2, W//2, size_block//2, num_window//2
    h_corner0, w_corner0 \
        = h_center - width_block, w_center - width_block
    list_id = list(range(-width_id,  -width_id+num_window))
    list_pair_offset = list((i*size_block,j*size_block) \
                            for i in list_id for j in list_id)
    window = hamming(size_block).reshape(-1, 1)
    window = window @ window.T
    Z = np.zeros((size_block, size_block))
    for dh, dw in list_pair_offset:
        h_corner, w_corner = h_corner0 + dh, w_corner0 + dw
        z = window * x[h_corner: h_corner+size_block,
                       w_corner: w_corner+size_block]
        # Compute the power spectrum for the NxN region.
        Z += square(abs(fft2(z)) / size_block)
    Z /= len(list_pair_offset)
    # Use fftshift to move the zero frequencies to the center
    Z = fftshift(Z)
    log_Z = log(Z)
    # Plot 3-D mesh plot and label the x and y axises
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x = np.linspace(-pi, pi, num=size_block)
    X, Y = np.meshgrid(x, x)
    cp = ax.plot_surface(X, Y, log_Z, cmap=plt.cm.coolwarm)
    ax.set_xlabel(r'$\mu$ axis')
    ax.set_ylabel(r'$\nu$ axis')
    str_title = r'3D Plot of $' + symbol \
        + r'(e^{j \mu}, e^{j \nu})$'
    ax.set_title(str_title)
    fig.colorbar(cp, shrink=0.5, aspect=5)
    return Z, fig
```

Code to plot power spectrum $\log S\left(e^{j\mu}, e^{j\nu}\right)$: `utils.py`

utils.py

```
from typing import Tuple, Union
import matplotlib.pyplot as plt
import numpy as np
from numpy import square
from math import pi, sin, cos, sqrt, log

def memoize(f):
   cache = {}
   def memoizedFunction(*args):
      if args not in cache:
         cache[args] = f(*args)
      return cache[args]
   memoizedFunction.cache = cache
   return memoizedFunction

sin = memoize(sin)
cos = memoize(cos)

def calc_spectrum_1d(coef: Union[dict, Tuple[dict, dict]],
                     list_omega: list) -> Tuple[list, list]:
    if isinstance(coef, dict):
        return calc_spectrum_1d_dict(coef, list_omega)
    else:
        return calc_spectrum_1d_tuple(coef, list_omega)

def calc_spectrum_1d_dict(dict_coef: dict,
                          list_omega: list) -> Tuple[list, list]:
    func_re = lambda omega: sum([coef * cos( omega * n )
                                 for n, coef in dict_coef.items()])
    func_im = lambda omega: -sum([coef * sin( omega * n )
                                  for n, coef in dict_coef.items()])
    list_re = list( map(func_re, list_omega) )
    list_im = list( map(func_im, list_omega) )
    return list_re, list_im

def calc_spectrum_1d_tuple(tuple_coef: Tuple[dict, dict],
                           list_omega) -> Tuple[list, list]:
```

```
    list_re_num, list_im_num \
        = calc_spectrum_1d_dict(tuple_coef[0], list_omega)
    list_re_den, list_im_den \
        = calc_spectrum_1d_dict(tuple_coef[1], list_omega)
    list_re = [(a1*a2 + b1*b2) / (a2*a2 + b2*b2)
               for a1, b1, a2, b2 in
               list(zip(list_re_num, list_im_num,
                        list_re_den, list_im_den))]
    list_im = [(-a1*b2 + a2*b1) / (a2*a2 + b2*b2)
               for a1, b1, a2, b2 in
               list(zip(list_re_num, list_im_num,
                        list_re_den, list_im_den))]
    return list_re, list_im

def calc_omega_1d(num_point: int = 32) -> list:
    return [pi * i / num_point for i in range(num_point)]

def calc_mag_1d(list_re: list, list_im: list) -> list:
    return [sqrt(re*re+im*im) for re, im
            in list(zip(list_re, list_im))]

def get_mag_1d(coef: Union[dict, Tuple[dict, dict]],
               num_point: int = 32) -> Tuple[list, list]:
    list_omega = calc_omega_1d(num_point)
    list_re, list_im = calc_spectrum_1d(coef, list_omega)
    list_mag = calc_mag_1d(list_re, list_im)
    list_omega_neg = [-omega for omega in list_omega[-1:0:-1]]
    return list(reversed(list_mag))[:-1] + list_mag, \
        list_omega_neg + list_omega

def expand_1d_2d(a: list, b: list) -> np.ndarray:
    a = np.array(a).reshape((-1, 1))
    b = np.array(b).reshape((1, -1))
    return a @ b

def plot_3d(list_mag: Union[list, np.ndarray],
            list_omega: list, symbol='H') -> None:
    X, Y = np.meshgrid(list_omega, list_omega)
    Z = expand_1d_2d(list_mag, list_mag) \
        if isinstance(list_mag, list) else list_mag
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    cp = ax.plot_surface(X, Y, Z, cmap=plt.cm.coolwarm) # lab 2
```

```
    fig.colorbar(cp, shrink=0.5, aspect=5) # lab 2
    str_title = r'3D Plot of $' + symbol \
        + r'(e^{j \mu}, e^{j \nu})$' # lab 2
    ax.set_title(str_title)
    ax.set_xlabel(r'$\mu$')
    ax.set_ylabel(r'$\nu$')

def plot_contour(list_mag: Union[list, np.ndarray],
                 list_omega: list, symbol='H') -> None:
    X, Y = np.meshgrid(list_omega, list_omega)
    Z = expand_1d_2d(list_mag, list_mag) \
        if isinstance(list_mag, list) else list_mag
    fig, ax=plt.subplots(1, 1)
    cp = ax.contourf(X, Y, Z, cmap=plt.cm.coolwarm) # lab 2
    fig.colorbar(cp, shrink=0.5, aspect=5) # lab 2
    str_title = r'Contour Plot of $' + symbol \
        + r'(e^{j \mu}, e^{j \nu})$' # lab 2
    ax.set_title(str_title)
    ax.set_xlabel(r'$\mu$')
    ax.set_ylabel(r'$\nu$')

def filter_IIR_lowpass(x: np.ndarray, pole: float,
                       amplitude: float=1) -> np.ndarray:
    H, W = x.shape
    a00, b01, b11 = square(amplitude*(1-pole)), -pole, pole*pole
    reg, y = np.zeros(W), np.zeros(x.shape)
    y[0][0] = a00 * x[0][0]
    for col in range(1, W):
        y[0, col] = a00 * x[0, col] - b01 * y[0, col-1]
    for row in range(1, H):
        y[row, 0] = a00 * x[row, 0] - b01 * y[row-1, 0]
    for row in range(1, H):
        reg[1:] = b01 * y[row-1, 1:] + b11 * y[row-1, :-1]
        for col in range(1, W):
            y[row, col] = a00 * x[row, col]
                - b01 * y[row, col-1] - reg[col]
    return y
```

Codes for solutions

solution to section 1: `soln_1.py`

```python
import sys
from os.path import dirname
sys.path.insert(0, dirname(dirname(__file__)))
from src.spec_anal import SpecAnal, BetterSpecAnal
import numpy as np
from PIL import Image

if __name__ == '__main__':
    x = np.array(Image.open('resource/img04g.tif'))
    list_size_block = [64, 128, 256]
    list_order = ['a', 'b', 'c']
    for size_block, order in list(zip(list_size_block, list_order)):
        power_spectrum1, fig1 = SpecAnal(x, size_block) # 5^2 windows
        fig1.savefig(f'./result/fig_1_2%s.png'%order,
                     bbox_inches='tight')
    power_spectrum2, fig2 \
        = BetterSpecAnal(x, size_block=64, num_window=5)
    fig2.savefig(f'./result/fig_1_3.png', bbox_inches='tight')
```

solution to section 2: `soln_2.py`

```
import sys
from os.path import dirname
sys.path.insert(0, dirname(dirname(__file__)))
from PIL import Image
import numpy as np
from numpy import sqrt, square, log
from numpy.random import uniform
import matplotlib.pyplot as plt
from src.spec_anal import BetterSpecAnal
from src.utils import get_mag_1d, expand_1d_2d, plot_3d, plot_contour
from src.utils import filter_IIR_lowpass

if __name__ == '__main__':
    amplitude_x = 0.5
    variance_x = square(amplitude_x) / 3
    x = uniform(low=-amplitude_x, high=amplitude_x, size=(512, 512))
    x_scaled = 255 * (x + 0.5)
    img_scaled = Image.fromarray(x_scaled.astype(np.uint8))
    img_scaled.save('result/fig_2_1.png')
    # filter x with h
    # H(z) = sqrt(3) / (1 - 0.99 z^{-1})
    pole_h = 0.99
    amplitude_h = sqrt(3) / (1-pole_h)
    h = ({0: amplitude_h * (1-pole_h) }, {0: 1, 1: -pole_h})
    y = filter_IIR_lowpass(x, pole_h, amplitude_h)
    y_offset = y + 127
    y_offset[y_offset < 0] = 0
    y_offset[y_offset > 255] = 255
    img_y = Image.fromarray(y_offset.astype(np.uint8))
    img_y.save('result/fig_2_2.png')
    # power spectrum of y, theoretical
    size_block = 64
    list_mag, list_omega = get_mag_1d(h, num_point=size_block)
    list_mag = expand_1d_2d(list_mag, list_mag) # |H(z1, z2)|
    log_power_spectrum1 = log(square(list_mag)) # |S_h|=|H(z1, z2)|^2
    # |S_y| = |S_h| |S_x|, S_x = E[x^2] = Var[x^2]
    log_power_spectrum1 += log(variance_x)
    plot_3d(log_power_spectrum1, list_omega, symbol='\log S_y')
    plt.savefig("./result/fig_2_3a.png", bbox_inches='tight')
```

```
    plt.show()
    plot_contour(log_power_spectrum1, list_omega, symbol='\log S_y')
    plt.savefig("./result/fig_2_3b.png", bbox_inches='tight')
    plt.show()
    # power spectrum of y, empirical
    log_power_spectrum2, fig \
        = BetterSpecAnal(y, size_block=size_block,
                         num_window=5, symbol='\log S_y')
    fig.savefig(f'./result/fig_2_4.png', bbox_inches='tight')
```