ECE637 Digital Image Processing I
Laboratory work 2:
2-D Random Processes

Boris Diner

February 3, 2021

1 Power Spectral Density of an Image

In this section, the power spectral density of a gray scale image is analyzed. The code in py-file *SpecAnal.py* estimates the power spectral density of an image by computing the logarithm of the normalized energy spectrum over different block sizes of an image, 64×64, 128×128, and 256×256.

1.1 The gray scale image *img04g.tif*

The image to be analyzed is shown in Figure 1.

Figure 1: The gray scale image *img04g.tif* for which the power spectral density is estimated

1.2 Plots of the power spectral density for different block sizes

It can be observed from figures 2-4 that the power spectral density estimate remains noisy independently of the block size. Also, it should be noted that the peak of the power spectral density is becoming smoother as the block size is getting bigger.

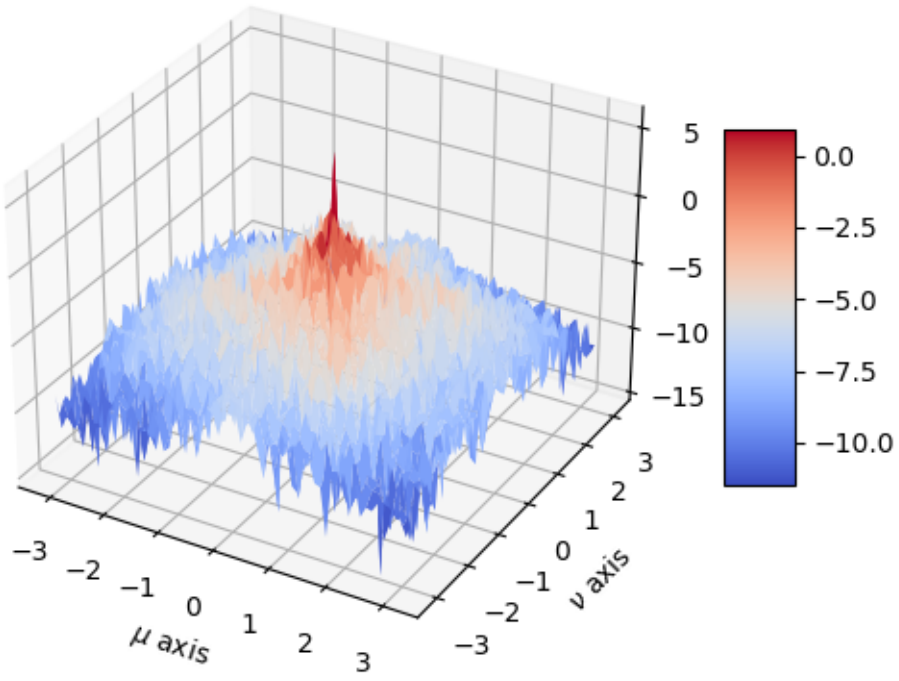

Figure 2: The power spectral density of the image *img04g.tif* with the block size 64×64

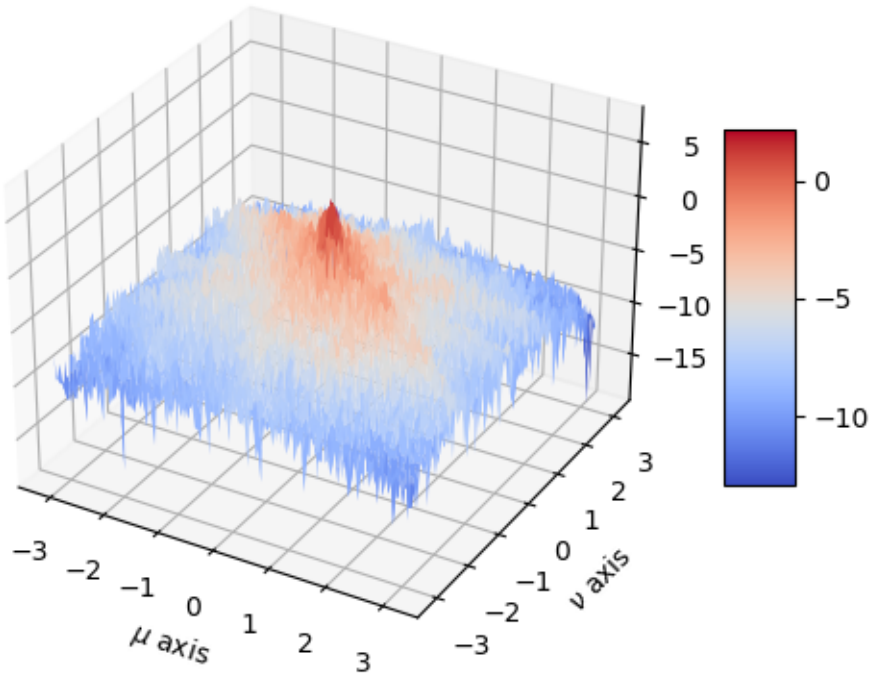

Figure 3: The power spectral density of the image *img04g.tif* with the block size 128×128

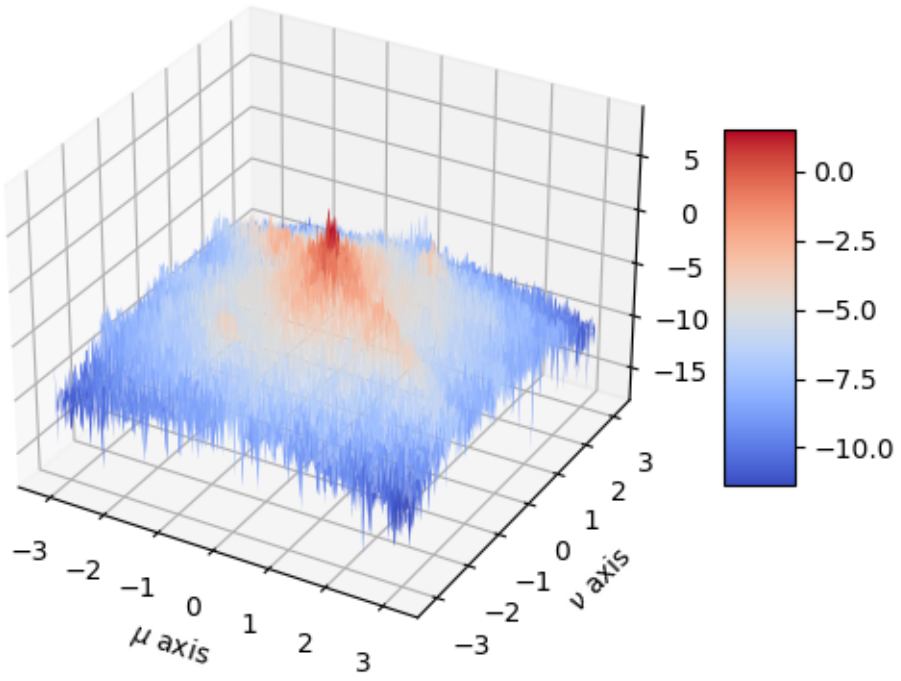

Figure 4: The power spectral density of the image *img04g.tif* with the block size 256×256

1.3 The improved power spectral density estimate

Dividing the initial image into 25 non-overlapping windows, multiplying each of the windows by a 2-D separable Hamming window, computing the sqaured DFT magnitude for each window and then averaging over all 25 windows, we get the following result (Figure 5).

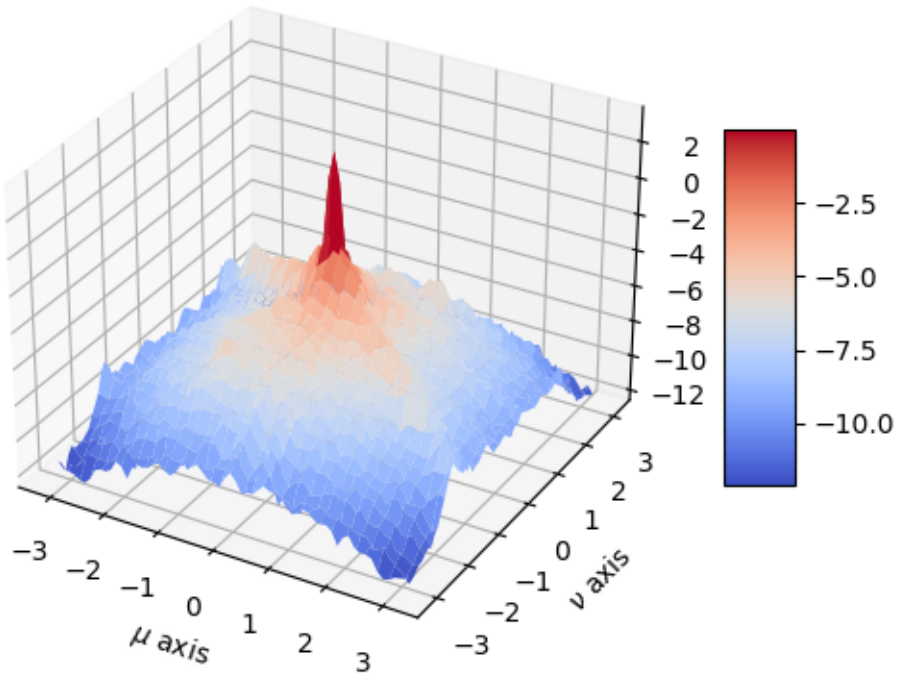

Figure 5: The estimate of the power spectral density of *img04g.tif* by averaging the results across 25 windows

1.4 Python code for BetterSpecAnal.py

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

def BetterSpecAnal(x):
  # The number of pixels in a chunk (one direction)
  N = 64

  # The number of windows in horiztontal and vertical directions
  NUM_WIN = 5

  # Create the 2-D Hamming window
```

```
  W = np.outer(np.hamming(N), np.hamming(N))

  # Set the coordinates of the initial position to iterate over
                                    NUM_WINxNUM_WIN windows
  h_start, v_start = np.subtract(im.size,(NUM_WIN*N, NUM_WIN*N)) //
                                    2

  # Initialize the values of the FT with zeros
  Z = np.zeros((N,N))

  # Iterate over NUM_WINxNUM_WIN windows
  for v_chunk in range(NUM_WIN):
    for h_chunk in range(NUM_WIN):

      z = x[v_start+v_chunk*N:v_start+(v_chunk+1)*N, h_start+h_chunk
                                    *N:h_start+(h_chunk+1)*N]
      Z += (1/N)**2*np.abs(np.fft.fft2(z*W))**2

  # Normalize the result by the number of windows
  Z /= NUM_WIN * NUM_WIN

  # Use fftshift to move the zero frequencies to the center of the
                                    plot.
  Z = np.fft.fftshift(Z)

  # Compute the logarithm of the Power Spectrum.
  Zabs = np.log(Z)

  return Zabs


# Read in a gray scale TIFF image.
im = Image.open('img04g.tif')
print('Read img04.tif.')
print('Image size: ', im.size)

# Display image object by PIL.
im.show(title='image')

# Import Image Data into Numpy array.
# The matrix x contains a 2-D array of 8-bit gray scale values.
x = np.array(im)
print('Data type: ', x.dtype)

# Display numpy array by matplotlib.
plt.imshow(x, cmap=plt.cm.gray)
plt.title('Image')

# Set colorbar location. [left, bottom, width, height].
```

```
cax = plt.axes([0.9, 0.15, 0.04, 0.7])
plt.colorbar(cax=cax)
plt.show()

x = np.double(x)/255.0

N = 64

Zabs = BetterSpecAnal(x)

# Plot the result using a 3-D mesh plot and label the x and y axises
                                    properly.
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
a = b = np.linspace(-np.pi, np.pi, num = N)
X, Y = np.meshgrid(a, b)

surf = ax.plot_surface(X, Y, Zabs, cmap=plt.cm.coolwarm)

ax.set_xlabel('$\\mu$ axis')
ax.set_ylabel('$\\nu$ axis')
ax.set_zlabel('Z Label')

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

2 Power Spectral Density of a 2-D AR Process

2.1 The image $255 \cdot (x + 0.5)$

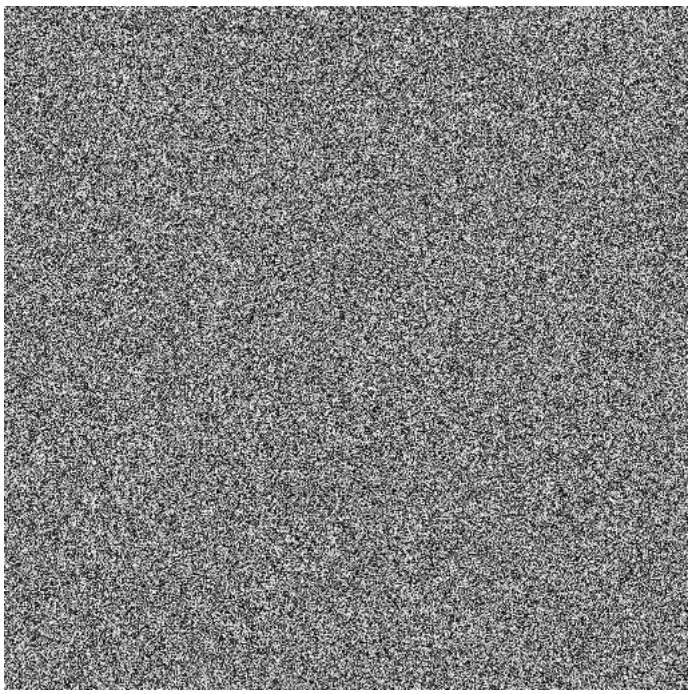

Figure 6: The random image *rand_ img.tif* generated as independent random numbers each uniformly distirbuted on the interval $[-0.5, 0.5]$

2.2 The image $(y + 127)$

We are given that

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

By definition,

$$H(z_1, z_2) \triangleq \frac{Y(z_1, z_2)}{X(z_1, z_2)}$$

Therefore,

$$3X(z_1, z_2) = Y(z_1, z_2) - 0.99z_1^{-1}Y(z_1, z_2) \\ - 0.99z_2^{-1}Y(z_1, z_2) + 0.9801z_1^{-1}z_2^{-1}Y(z_1, z_2)$$

$$Y(z_1, z_2) = 3X(z_1, z_2) + 0.99z_1^{-1}Y(z_1, z_2) \\ + 0.99z_2^{-1}Y(z_1, z_2) - 0.9801z_1^{-1}z_2^{-1}Y(z_1, z_2)$$

Applying the inverse Z-transform, we get

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

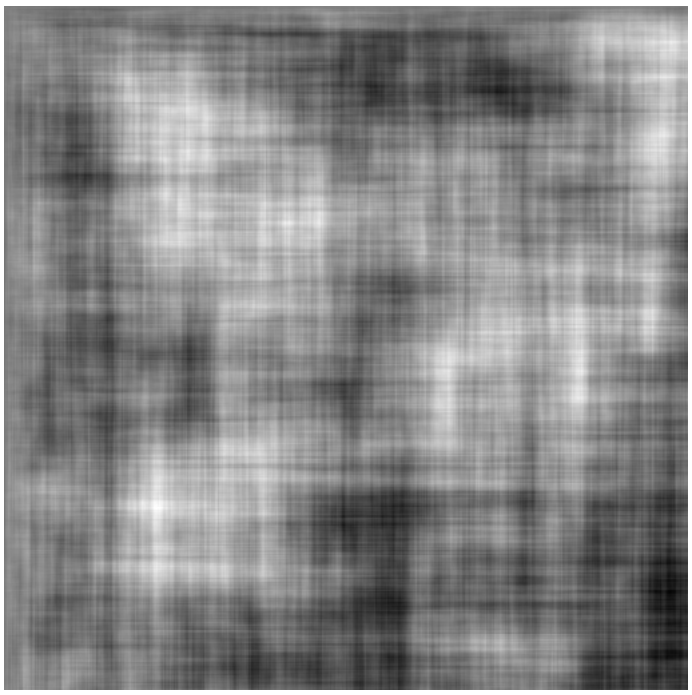

Figure 7: The filtered random image $y + 127$ (*rand_ img_ filtered.tif*)

2.3 Python code to generate and filter the random image

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Set the size of an image
size = (512, 512)

# Create a new grayscale image with the given size.
rand_img = Image.new('L', size)

# Generate a random image in accordance with the uniform
                                    distribution on [-0.5, 0.5]
x = np.random.uniform(-0.5, 0.5, size)

# Scale the image
x_scaled = 255 * (x + 0.5)

# Display x_scaled by matplotlib.
plt.imshow(x_scaled.astype('uint8'), cmap=plt.cm.gray)
plt.title('A randomly generated image')
plt.show()

# Save the image x_scaled
rand_img = Image.fromarray(x_scaled.astype('uint8'))
rand_img.save('rand_img.tif')


# Filter the image
y = np.zeros((size[0]+1, size[1]+1))

for m in range(size[0]):
  for n in range(size[1]):
    y[m+1][n+1] = 3*x[m][n]+0.99*y[m][n+1]+0.99*y[m+1][n]-0.9801*y[m
                                    ][n]

y = y[1:,1:]
y += 127
y = np.clip(y, 0, 255)

# Display the filtered image by matplotlib.
plt.imshow(y.astype('uint8'), cmap=plt.cm.gray)
plt.title('The filtered image')
plt.show()

# Save the filtered image
```

```
rand_img_filtered = Image.fromarray(y.astype('uint8'))
rand_img_filtered.save('rand_img_filtered.tif')
```

2.4 The mesh plot of $\log S_y(e^{j\mu}, e^{j\nu})$

We have a 2-D linear system

$$Y(m,n) = h(m,n) * X(m,n)$$

where $X(m,n)$ is a 2-D wide sense stationary random process.

The power spectral density of $Y(m,n)$ can be derived from the following relation

$$S_y(e^{j\mu}, e^{j\nu}) = |H(e^{j\mu}, e^{j\nu})|^2 S_x(e^{j\mu}, e^{j\nu}) \tag{2}$$

The transfer function of the linear system we are considering is specified by equation 1.

Also, we are given independent identically distributed uniform random variables on the interval $[-0.5, 0.5]$ with distribution $U(0, 1/12)$. Therefore, $X(m,n)$ is wide sense stationary with

$$\mu(m,n) = E[X(m,n)] = 0$$
$$R_x(k,l) = E[X(0,0)X(k,l)] = \sigma_x^2 \delta(k,l) = \frac{1}{12}\delta(k,l)$$

Then the power spectral density of the random process $X(m,n)$

$$S_x(e^{j\mu}, e^{j\nu}) = \text{DSFT}\{R_x(k,l)\} = \frac{1}{12}$$

Plugging $z_1 = e^{j\mu}$ and $z_2 = e^{j\nu}$ in 1 and using 2, we find that the power spectral density of $Y(m,n)$

$$S_y(e^{j\mu}, e^{j\nu}) = \frac{1}{12} \cdot \left| \frac{3}{1 - 0.99e^{-j\mu} - 0.99e^{-j\nu} + 0.9801e^{-j\mu}e^{-j\nu}} \right|^2$$

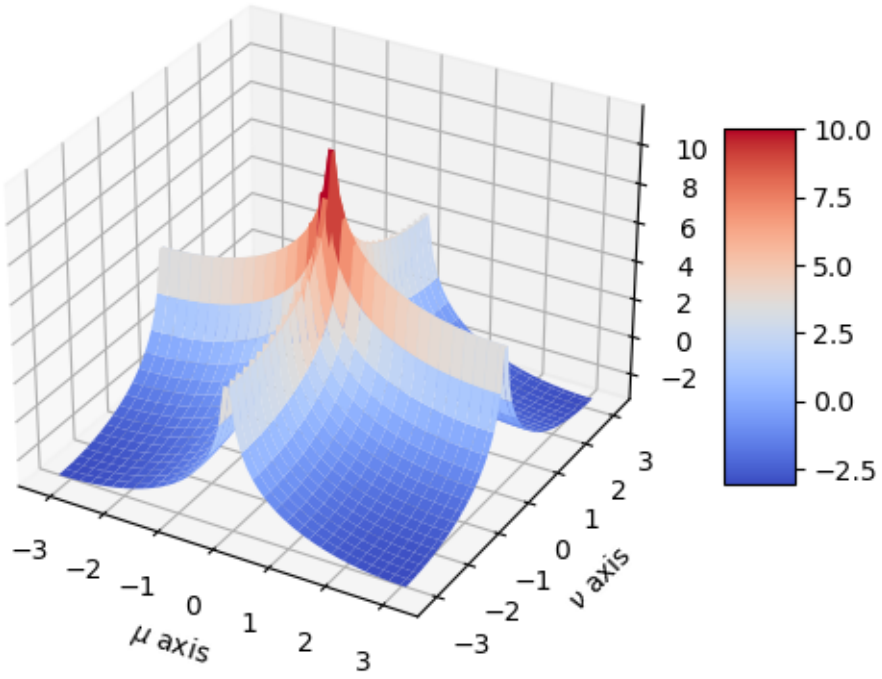

Figure 8: The mesh plot of $\log S_y(e^{j\mu}, e^{j\nu})$

2.5 Python code to plot $\log S_y(e^{j\mu}, e^{j\nu})$

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

PSD_X = 1/12
```

```
N = 64

a = b = np.linspace(-np.pi, np.pi, num = N)
X, Y = np.meshgrid(a, b)

psd_y = PSD_X * abs(3/((1-0.99*np.exp(-1j*X)-0.99*np.exp(-1j*Y)+0.
                                    9801*np.exp(-1j*X)*np.exp(-1j*Y)))
                                    )**2

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(X, Y, np.log(psd_y), cmap=plt.cm.coolwarm)

ax.set_xlabel('$\\mu$ axis')
ax.set_ylabel('$\\nu$ axis')
ax.set_zlabel('Z Label')

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

2.6 The mesh plot of the estimated power spectral density of y

Applying the function *BetterSpecAnal(x)* described in subsection Python code for BetterSpecAnal.py to the filtered random image *rand_img_filtered.tif*, we get the following result (Figure 9).

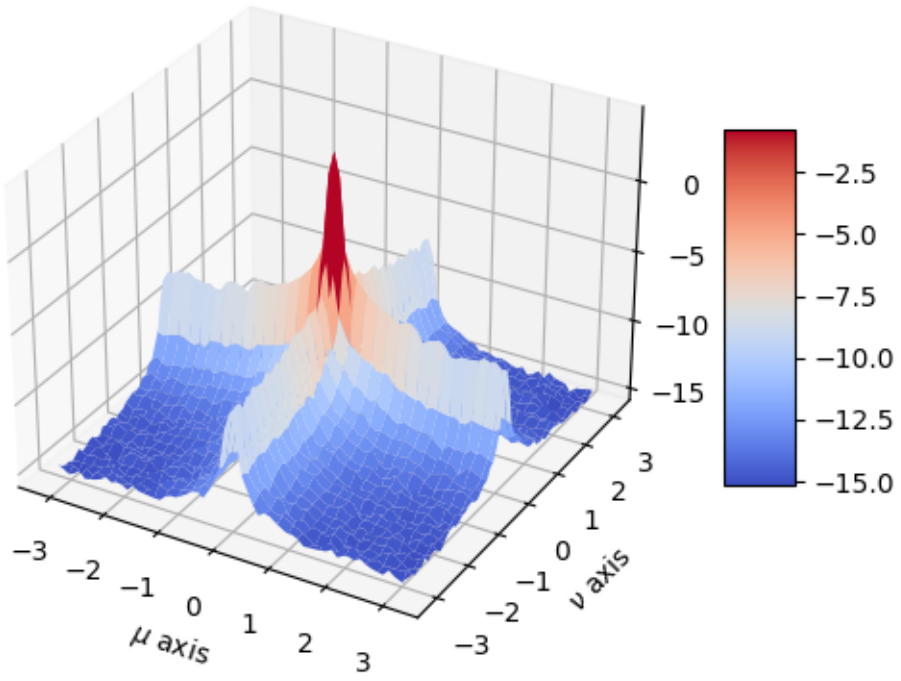

Figure 9: The mesh plot of the estimated power spectral density of y