(Gonzalez 3rd edition)

1. Problem 5.5 (10%)

   The solutions are shown in Fig. P5.5, from left to right.

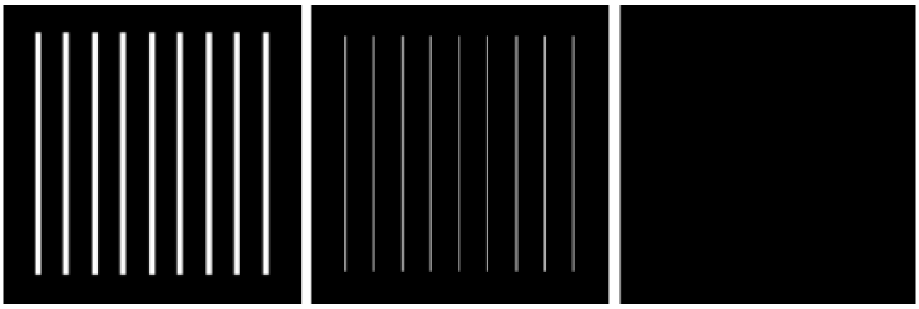


Figure P5.5

2. Problem 5.6 (10%)

   The solutions are shown in Fig. P5.6, from left to right.

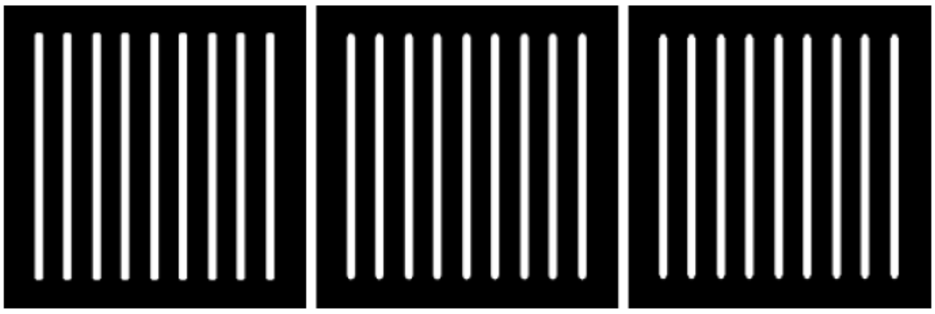


Figure P5.6

3. Problem 5.15 (10%)

   From Eq. (5.4-19)

$$\sigma^2 = \frac{1}{(2a+1)(2b+1)}\sum\sum\{[g(\gamma)-w\eta(\gamma)]-[\overline{g}-w\overline{\eta}]\}^2$$

where "γ" indicates terms affected by the summations. Letting $K = 1/(2a+1)(2b+1)$, taking the partial derivative of σ^2 with respect to w and setting the result equal to zero gives

$$\begin{aligned}
\frac{\partial\sigma^2}{\partial w} &= K\sum\sum 2\left[g(\gamma)-w\eta(\gamma)-\overline{g}+w\overline{\eta}\right]\left[-\eta(\gamma)+\overline{\eta}\right]=0\\
&= K\sum\sum -g(\gamma)\eta(\gamma)+g(\gamma)\overline{\eta}+w\eta^2(\gamma)-w\eta(\gamma)\overline{\eta}+\\
&\qquad \overline{g}\eta(\gamma)-\overline{g}\,\overline{\eta}-w\overline{\eta}\eta(\gamma)+w\overline{\eta}^2\\
&= -\overline{g\eta}+\overline{g}\,\overline{\eta}+w\overline{\eta^2}-w\overline{\eta}^2+\overline{g}\,\overline{\eta}-\overline{g}\,\overline{\eta}-w\overline{\eta}^2+w\overline{\eta}^2\\
&= -\overline{g\eta}+\overline{g}\,\overline{\eta}+w\left(\overline{\eta^2}-\overline{\eta}^2\right)\\
&= 0
\end{aligned}$$

where, for example, we used the fact that

$$\frac{1}{(2a+1)(2b+1)}\sum\sum g(\gamma)\eta(\gamma)=\overline{g\eta}.$$

Solving for w gives us

$$w=\frac{\overline{g\eta}-\overline{g}\,\overline{\eta}}{\overline{\eta^2}-\overline{\eta}^2}.$$

Finally, inserting the variables x and y,

$$w(x,y)=\frac{\overline{g(x,y)\eta(x,y)}-\overline{g}(x,y)\overline{\eta}(x,y)}{\overline{\eta^2}(x,y)-\overline{\eta}^2(x,y)}$$

which agrees with Eq. (5.4-21).

4. Problem 5.16 (10%)
From Eq. (5.5-13),

$$g(x,y)=\iint_{-\infty}^{\infty} f(\alpha,\beta)h(x-\alpha,y-\beta)\,d\alpha\,d\beta.$$

It is given that $f(x,y)=\delta(x-a)$, so $f(\alpha,\beta)=\delta(\alpha-a)$. Then, using the impulse response given in the problem statement,

$$\begin{aligned}
g(x,y) &= \iint_{-\infty}^{\infty} \delta(\alpha-a)e^{-\left[(x-\alpha)^2+(y-\beta)^2\right]}\,d\alpha\,d\beta \\
&= \iint_{-\infty}^{\infty} \delta(\alpha-a)e^{-\left[(x-\alpha)^2\right]}\,e^{-\left[(y-\beta)^2\right]}\,d\alpha\,d\beta \\
&= \int_{-\infty}^{\infty} \delta(\alpha-a)e^{-\left[(x-\alpha)^2\right]}\,d\alpha \int_{-\infty}^{\infty} e^{-\left[(y-\beta)^2\right]}\,d\beta \\
&= e^{-\left[(x-a)^2\right]}\int_{-\infty}^{\infty} e^{-\left[(y-\beta)^2\right]}\,d\beta
\end{aligned}$$

where we used the fact that the integral of the impulse is nonzero only when $\alpha=a$. Next, we note that

$$\int_{-\infty}^{\infty} e^{-\left[(y-\beta)^2\right]}\,d\beta=\int_{-\infty}^{\infty} e^{-\left[(\beta-y)^2\right]}\,d\beta$$

which is in the form of a constant times a Gaussian density with variance $\sigma^2=1/2$ or standard deviation $\sigma=1/\sqrt{2}$. In other words,

$$e^{-\left[(\beta-y)^2\right]}=\sqrt{2\pi(1/2)}\left[\frac{1}{\sqrt{2\pi(1/2)}}e^{-(1/2)\left[\frac{(\beta-y)^2}{(1/2)}\right]}\right].$$

The integral from minus to plus infinity of the quantity inside the brackets is 1, so

$$g(x,y)=\sqrt{\pi}e^{-[(x-a)^2]}$$

which is a blurred version of the original image.

5. Problem 5.17 (10%)

Following the image coordinate convention in the book, vertical motion is in the x-direction and horizontal motion is in the y-direction. Then, the components of motion are as follows:

$$x_0(t)=\begin{cases} \frac{at}{T_1} & 0\le t\le T_1 \\ a & T_1<t\le T_1+T_2 \end{cases}$$

and

$$y_0(t)=\begin{cases} 0 & 0\le t\le T_1 \\ \frac{b(t-T_1)}{T_1} & T_1<t\le T_1+T_2. \end{cases}$$

$$H(u,v) = \int_0^T e^{-j2\pi[ux_0(t)+vy_0(t)]}\,dt \qquad (5.6\text{-}8)$$

Then, substituting these components of motion into Eq. (5.6-8) yields

$$\begin{aligned} H(u,v) &= \int_0^{T_1} e^{-j2\pi[uat/T_1]}dt+\int_{T_1}^{T_1+T_2} e^{-j2\pi[ua+vb(t-T_1)/T_2)}dt \\ &= \frac{T_1}{\pi ua}\sin(\pi ua)e^{-j\pi ua}+e^{-j2\pi ua}\int_{T_1}^{T_1+T_2} e^{-j2\pi vb(t-T_1)/T_2}dt \\ &= \frac{T_1}{\pi ua}\sin(\pi ua)e^{-j\pi ua}+e^{-j2\pi ua}\int_0^{T_2} e^{-j2\pi vb\tau/T_2}d\tau \\ &= \frac{T_1}{\pi ua}\sin(\pi ua)e^{-j\pi ua}+e^{-j2\pi ua}\frac{T_2}{\pi vb}\sin(\pi vb)e^{-j\pi vb} \end{aligned}$$

where in the third line we made the change of variables $\tau=t-T_1$. The blurred image is then

$$g(x,y)=\Im^{-1}[H(u,v)F(u,v)]$$

where $F(u,v)$ is the Fourier transform of the input image.

6. Problem 5.21 (10%)

The key to solving this problem is to recognize that the given function,

$$h(x,y)=\frac{x^2+y^2-2\sigma^2}{\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

is the the second derivative (Laplacian) of the function (see Section 3.6.2 regarding the Laplacian)

$$s(x,y)=e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

That is,

$$\begin{aligned}\nabla^2[s(x,y)] &= \left[\frac{\partial^2 s(x,y)}{\partial x^2}+\frac{\partial^2 s(x,y)}{\partial y^2}\right] \\ &= \frac{x^2+y^2-2\sigma^2}{\sigma^4}e^{-\frac{x^2+y^2}{2\sigma^2}}.\end{aligned}$$

(This result is derived in Section 10.2.6). So, it follows that

$$\begin{aligned}H(u,v) &= \Im\left[h(x,y)\right] \\ &= \Im\left[\nabla^2 s(x,y)\right].\end{aligned}$$

But, we know from the statement of Problem 4.26(a) that

$$\Im\left[\nabla^2 s(x,y)\right] = -4\pi^2(u^2+v^2)F(u,v)$$

where

$$\begin{aligned}F(u,v) &= \Im\left[s(x,y)\right] \\ &= \Im\left[e^{-\frac{x^2+y^2}{2\sigma^2}}\right].\end{aligned}$$

Therefore, we have reduced the problem to computing the Fourier transform of a Gaussian function. From the basic form of the Gaussian Fourier transform pair given in entry 13 of Table 4.3 (note that (x,y) and (u,v) in the present problem are the reverse of the entry in the table), we have

$$\Im\left[e^{-\frac{x^2+y^2}{2\sigma^2}}\right] = 2\pi\sigma^2 e^{-2\pi^2\sigma^2(u^2+v^2)}$$

so we have the final result

$$\begin{aligned}H(u,v) &= -4\pi^2(u^2+v^2)F(u,v) \\ &= \left[-4\pi^2(u^2+v^2)\right]\left[2\pi\sigma^2 e^{-2\pi^2\sigma^2(u^2+v^2)}\right] \\ &= -8\pi^3\sigma^2(u^2+v^2)e^{-2\pi^2\sigma^2(u^2+v^2)}\end{aligned}$$

as desired. Keep in mind that the preceding derivations are based on assuming continuous variables. A discrete filter is obtained by sampling the continuous function.

7. Problem 5.26 (10%)
   One possible solution: (1) Perform image averaging to reduce noise. (2) Obtain a blurred image of a bright, single star to simulate an impulse (the star should be as small as possible in the field of view of the telescope to simulate an impulse as closely as possible. (3) The Fourier transform of this image will give $H(u,v)$. (4) Use a Wiener filter and vary K until the sharpest image possible is obtained.

8. **Noise Reduction Using a Median Filter** (10%)
(a) Develop a program that can perform 3 x 3 median filtering.
(b) Download Fig. 5.7(a) from the course web site and add salt-and-pepper noise to it, with $P_a = P_b = 0.2$.
(c) Apply median filtering to the image in (b). Explain any major differences between your result and Fig. 5.10(b).
Ans:
(a)

(Matlab code)

```
clc;clear all;
src = imread('Fig0507(a)(ckt-board-orig).tif');
% median filter
med_src = medfilt2(src, [3, 3]);
figure(1);
imshow(med_src);
```

(Python code)

```
import cv2
import numpy as np

src = cv2.imread('Fig0507(a)(ckt-board-orig).tif', 0)
med_src = cv2.medianBlur(src, 3)
cv2.imshow('Median Filter', med_src)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

(b)

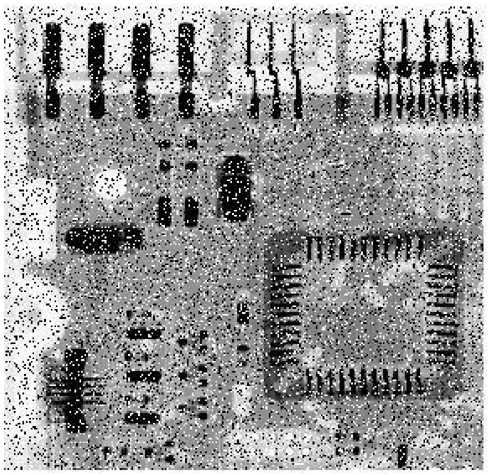
(Matlab code)

```
clc;clear all;
src = imread('Fig0507(a)(ckt-board-orig).tif');
% S&P noise
sp_src = imnoise(src, 'salt & pepper', 0.2);
figure(2);
imshow(sp_src);
```

(Python code)

```
import cv2
import numpy as np
def salt(img, n):
    for k in range(n):
        i = int(np.random.random() * img.shape[1])
        j = int(np.random.random() * img.shape[0])
        if img.ndim == 2:
            img[j,i] = 255
        elif img.ndim == 3:
            img[j,i,0]= 255
            img[j,i,1]= 255
            img[j,i,2]= 255
    return img
def pepper(img, n):
    for k in range(n):
        i = int(np.random.random() * img.shape[1])
        j = int(np.random.random() * img.shape[0])
        if img.ndim == 2:
            img[j, i] == 0
        elif img.ndim == 3:
            img[j,i,0]= 0
            img[j,i,1]= 0
            img[j,i,2]= 0
    return img
img = cv2.imread('Fig0507(a)(ckt-board-orig).tif', 0)
saltRe = salt(img, 30000)
result = pepper(saltRe, 30000)
```

```
cv2.imshow("SaltPepper", result)
cv2.waitKey(0)
```



(c)

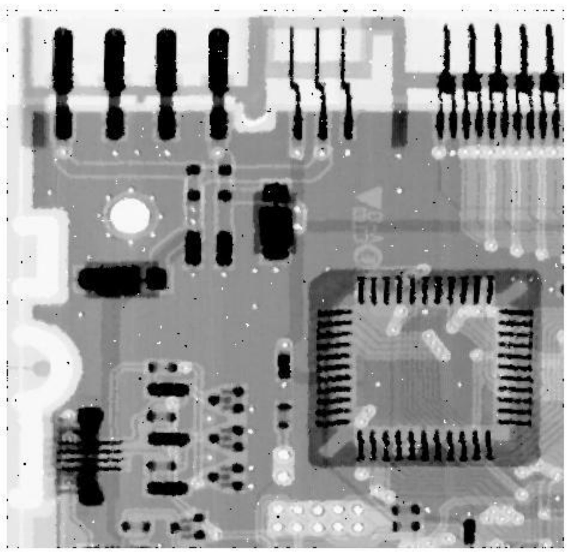
(Matlab code)

```
clc;clear all;
src = imread('Fig0507(a)(ckt-board-orig).tif');
% S&P noise
sp_src = imnoise(src, 'salt & pepper', 0.2);
figure(2);
imshow(sp_src);
% apply median filter to s&p
result = medfilt2(sp_src, [3, 3]);
figure(3);
imshow(result);
```

(Python code)

```
import cv2
import numpy as np

def salt(img, n):
    for k in range(n):
        i = int(np.random.random() * img.shape[1])
        j = int(np.random.random() * img.shape[0])
        if img.ndim == 2:
            img[j,i] = 255
        elif img.ndim == 3:
            img[j,i,0]= 255
            img[j,i,1]= 255
            img[j,i,2]= 255
    return img
def pepper(img, n):
    for k in range(n):
        i = int(np.random.random() * img.shape[1])
        j = int(np.random.random() * img.shape[0])
        if img.ndim == 2:
            img[j, i] == 0
        elif img.ndim == 3:
```

```
        img[j,i,0]= 0
        img[j,i,1]= 0
        img[j,i,2]= 0
    return img
img = cv2.imread('Fig0507(a)(ckt-board-orig).tif', 0)
saltRe = salt(img, 50000)
result = pepper(saltRe, 50000)
median = cv2.medianBlur(result, 5)
cv2.imshow("SaltPepper", result)
cv2.imshow("Median", median)
cv2.waitKey(0)
```



此張圖的 $P_a = P_b = 0.2$，明顯比 Fig. 5.10(b)還有更多雜訊，因此還原後比 Fig 5.10(b)較為模糊。

9. **Periodic Noise Reduction Using a Notch Filter** (10%)

(a) Write a program that implements sinusoidal noise of the form given in Problem 5.14. The inputs to the program must be the amplitude A, and the two frequency components u0 and v0 shown in the problem equation.

(b) Download image 5.26(a) from the course web site and add sinusoidal noise to it, with u0 = M/2 (the image is square) and v0 = 0. The value of A must be high enough for the noise to be clearly visible in the image.

(c) Compute and display the spectrum of the image. If the FFT program can only handle images of size equal to an integer power of 2, reduce the size of the image to 512 x 512 or 256 x 256. Resize the image before adding noise to it.

(d) Notch-filter the image using a notch filter of the form shown in Fig. 5.19(c).

Ans:

(Matlab code)

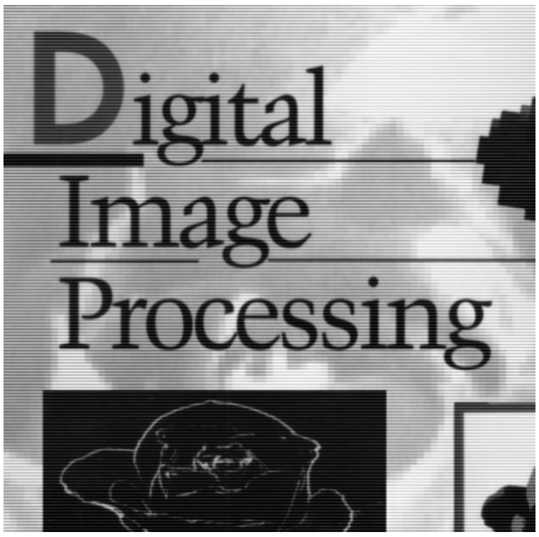
```
clc;clear all;close all;
img = double(imread('Fig0526(a)(original_DIP).tif'));
[M,N] = size(img);
A = input('sinusoidal noise amplitude = ');
u0 = input('sinusoidal noise u0 = ');
v0 = input('sinusoidal noise v0 = ');
if(isempty(A))
```

```
    A = 50;
end
if(isempty(u0))
    u0 = M/2;
end
if(isempty(v0))
    v0 = 0;
end
imgOF = fftshift(fft2(img));
imgOF_log = log(abs(imgOF) + 1);
for i=1:M
    for j=1:N
        img(i,j) = img(i,j) + A*sin(u0*i+v0*j);
    end
end
figure,imshow(uint8(img));
imgF = fftshift(fft2(img));
imgF_log = log(abs(imgF) + 1);
figure,imshow(mat2gray(imgF_log));
amplitudeThreshold = 0.2;
temp = abs(imgF_log-imgOF_log);
brightSpikes = temp > amplitudeThreshold;
figure,imshow(brightSpikes);
imgF(brightSpikes) = 0;
iFimg = ifft2(ifftshift(imgF));
iFimg = iFimg(1:M,1:N);
figure,imshow(uint8(abs(iFimg)));
```

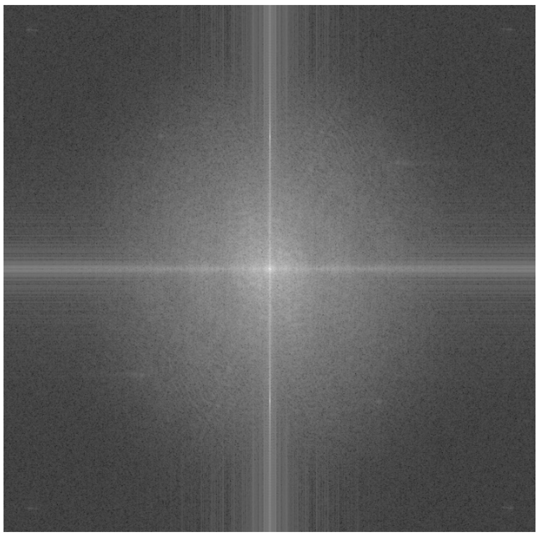
(Python code)

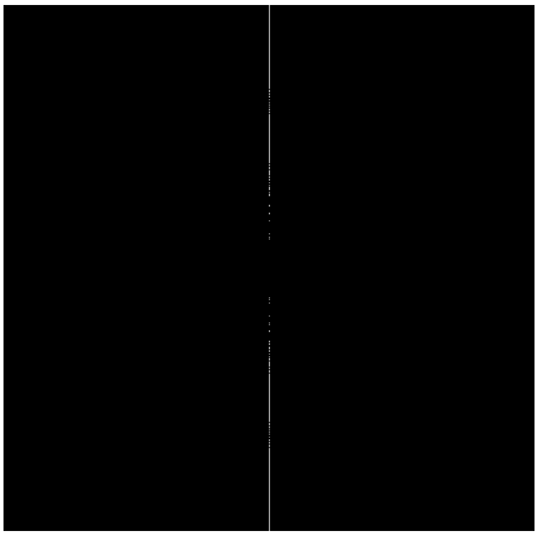
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread

img = np.double(imread('Fig0526(a)(original_DIP).tif'))
M, N = img.shape

A = input('sinusoidal noise amplitude = ')
u0 = input('sinusoidal noise u0 = ')
v0 = input('sinusoidal noise v0 = ')
if not A:
    A = 50
if not u0:
    u0 = M/2
if not v0:
    v0 = 0

plt.imshow(np.uint8(img), cmap='gray')
plt.show()
imgF = np.fft.fftshift(np.fft.fft2(img))
imgF_log = np.log(np.abs(imgF) + 1)
plt.imshow(imgF_log, cmap='gray')
plt.show()
```

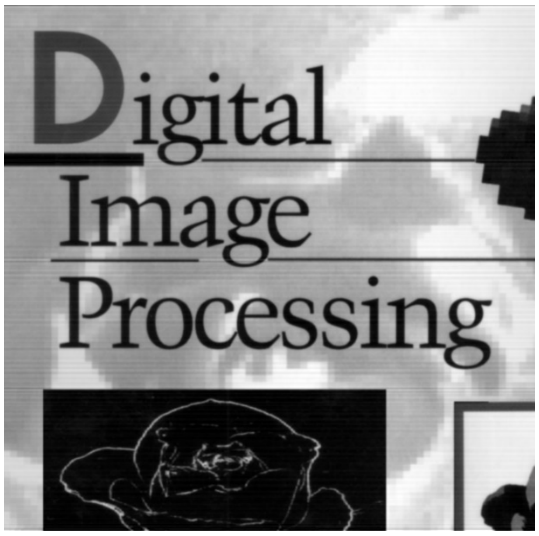
```
amplitudeThreshold = 0.2
temp = np.abs(imgF_log - imgF_log)
brightSpikes = temp > amplitudeThreshold
imgF[brightSpikes] = 0
iFimg = np.fft.ifft2(np.fft.ifftshift(imgF))
iFimg = iFimg[0:M, 0:N]
plt.imshow(np.abs(iFimg), cmap='gray')
plt.show()
```



Noisy image


Noisy image spectrum


Notch pass filter



Notch filtering image

10. **Parametric Wiener Filter** (10%)
(a) Implement a blurring filter as in Eq. (5.6-11).
(b) Download Fig. 5.26(a) from the course web site and blur it in the +45-degree direction using T = 1, as in Fig. 5.26(b).
(c) Add Gaussian noise of 0 mean and variance of 10 pixels to the blurred image.
(d) Restore the image using the parametric Wiener filter given in Eq. (5.8-6).
Ans:
(a)-(c)
(Matlab code)

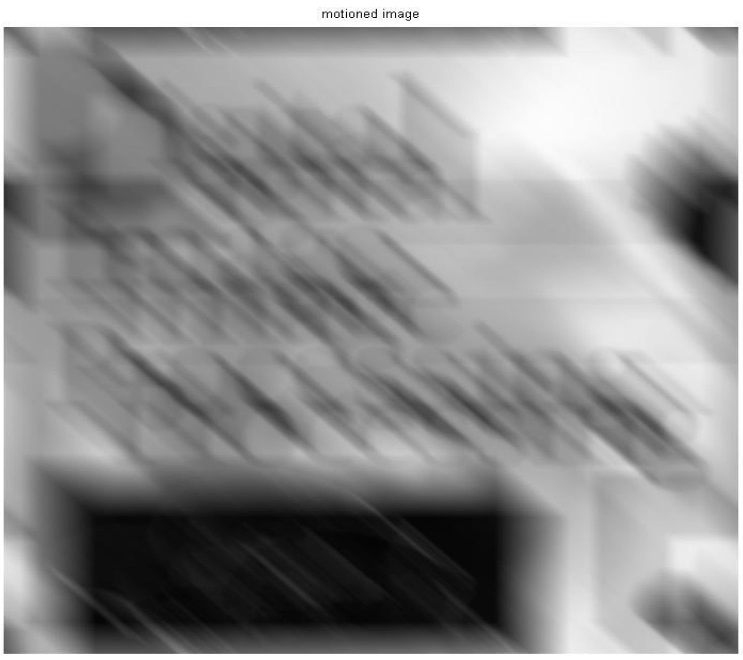
```
clear all;
```

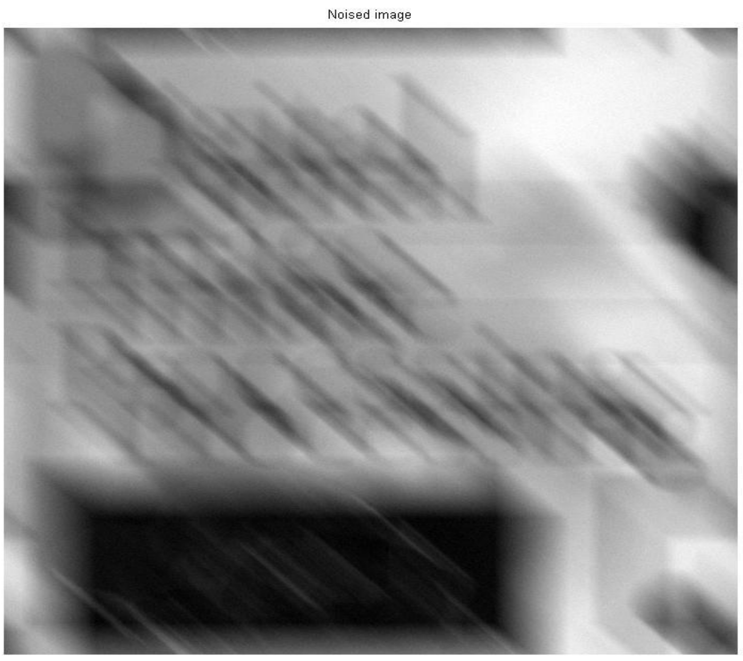
```
clc;

img = imread('Fig0526(a)(original_DIP).tif');
imgf = im2double(img);

dist = 100;ang = -45;
Hib = fspecial('motion',dist,ang);
imgM = imfilter(imgf,Hib,'conv','circular');

M = 0;
V = 10/255^2;%normalized
imgN = imnoise(imgM ,'gaussian',M,V);

nsR = V/var(imgf(:));
imgRe = deconvwnr(imgN,Hib,nsR);
figure;imshow(imgM);title('blurred image');
figure;imshow(imgN);title('Noised image');
figure;imshow(imgRe);title('Restored image');
```

(Python code)

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import cv2

def motion(height, weight, k=0.001):
    h_uv = np.zeros((height, weight), dtype='complex128')
    for u in range(height):
        for v in range(weight):
            q = np.power((u ** 2 + v ** 2), (5.0 / 6.0))
            h_uv[u][v] = np.exp(-(k * q))
    return h_uv
def Noise(src, h_uv):
    noise = np.random.normal(0, 1, size=src.shape)
    f_uv = np.fft.fft2(src)
    arr = np.multiply(f_uv, h_uv) + noise
    return arr, noise
def winner(src, h_uv, k=0.001):
    p_uv = np.conj(h_uv) / (np.abs(h_uv) ** 2 + 0.00005*k)
    rst = np.multiply(src, p_uv)
    return np.abs(np.fft.ifft2(rst))
def main(img_path):
    src = np.array(Image.open(img_path).convert("L"))
    height, weight = src.shape
    h_uv = motion(height, weight, k=0.01)
    noise_src, noise = Noise(src, h_uv)
    mot=np.abs(np.fft.ifft2(noise_src))
    mov_noi_src = np.abs(np.fft.ifft2(noise_src))
    k = np.power(np.abs(np.fft.fft2(noise)), 2) / np.power(np.abs(np.fft.fft2(src)), 2)
    src_processed_winner = winner(noise_src, h_uv, k)
    img_list = [src, mot, mov_noi_src, src_processed_winner]
    img_name = ["original", "motioned", "noised", "restored"]
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    _, axs = plt.subplots(2, 2, figsize=(12, 12))
    for i in range(2):
        for j in range(2):
            axs[i][j].imshow(img_list[i * 2 + j], cmap='gray')
            axs[i][j].set_title(img_name[i * 2 + j])
            axs[i][j].axes.get_xaxis().set_visible(False)
            axs[i][j].axes.get_yaxis().set_visible(False)
    plt.show()
if __name__ == '__main__':
    main('D:\pythonProject\Fig0526(a)(original_DIP).tif')
```

(d)





Restored image