سوال اول) الف: حذف نویز تصویر با کمک FFT:

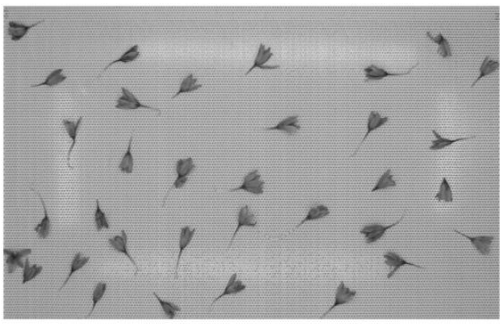
ابتدا تصویر را با کد رو به رو میخوانیم؛

```
%matplotlib inline
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
```

Part A

```
im = cv2.imread('images/img_01.jpg',0)
plt.imshow(im, cmap='gray')
plt.axis('off')
```

```
(-0.5, 1289.5, 814.5, -0.5)
```

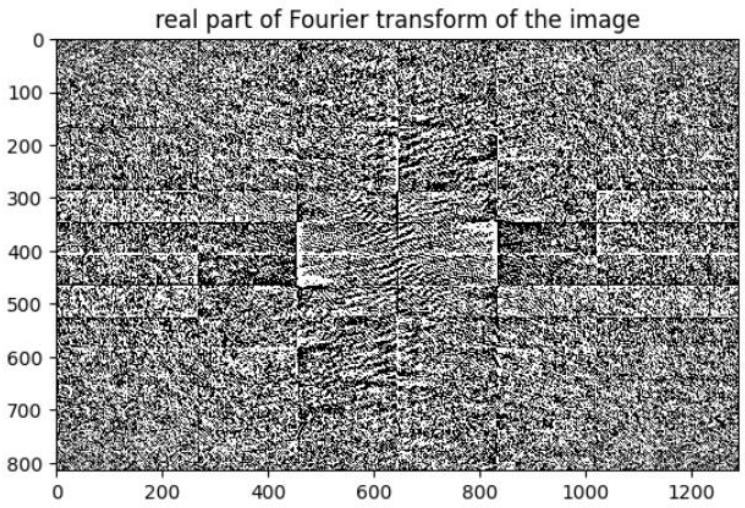


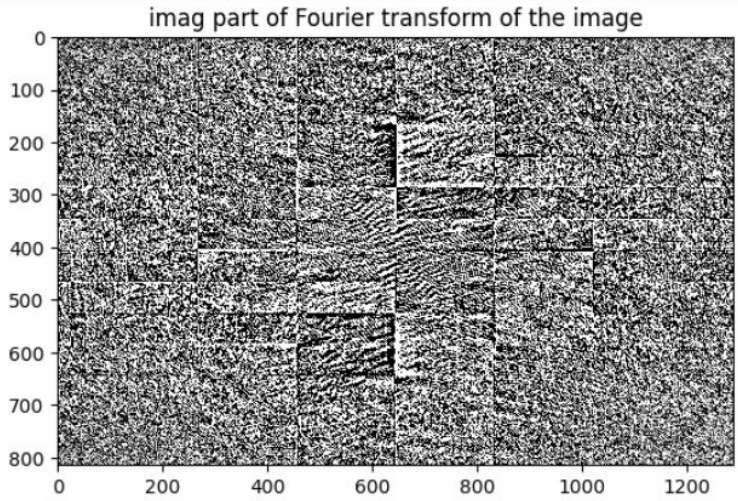
بعد از آن باید نویز های تصویر را تشخیص دهیم.

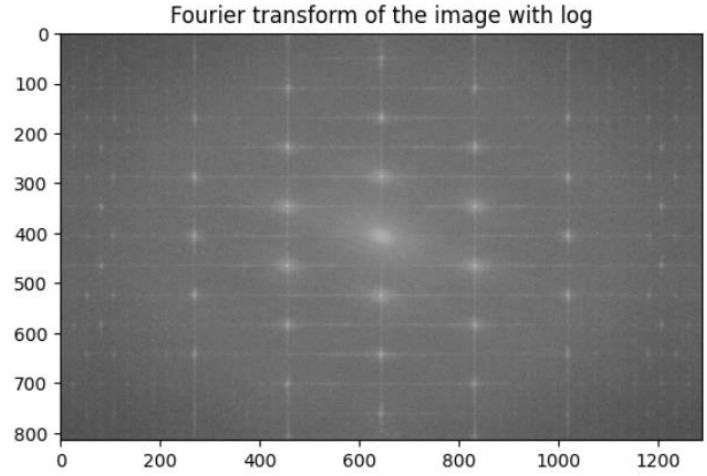
برای اینکار تبدیل فوریه ی تصویر را به دست می آوریم سپس برای نمایش مناسب آن را شیفت میدهیم تا نقطه ی صفر تصویر از گوشه ی سمت چپ و بالا به وسط تصویر انتقال پیدا کند سپس باید برای کنتراست بالاتر و تشخیص بهتر لگاریتم بگیریم که برای لگاریتم نیاز است تمام مقادیر مثبت باشند از آنجایی که مقدار فوریه عدد مختلط هستند باید اول اندازه ی آن ها را حساب کنیم سپس لگاریتم بگیریم بعد از اعمال این تغییرات تبدیل فوریه را رسم میکنیم.

```
denoised = im.copy()

denoisedfft = np.fft.fft2(denoised)
denoisedshift = np.fft.fftshift(denoisedfft)
denoisedabs = np.abs(denoisedshift)
plt.imshow(np.real(denoisedshift), cmap = 'gray', vmin=-255, vmax=255)
plt.title('real part of Fourier transform of the image')
plt.show()

plt.imshow(np.imag(denoisedshift), cmap = 'gray', vmin=-255, vmax=255)
plt.title('imag part of Fourier transform of the image')
plt.show()

denoisedlog = np.log(denoisedabs)
denoisedshow = 15 * denoisedlog
plt.imshow(denoisedshow, cmap = 'gray')
plt.title('Fourier transform of the image with log')
plt.show()
```

تصویر اول و دوم خروجی برای این بود که قسمت حقیقی و موهومی تصویر رو جداگانه ببینیم. تصویر سوم هم تبدیل فوریه ی تصویر نویزی با توجه به اندازه ی عدد مختلط حاصل است.

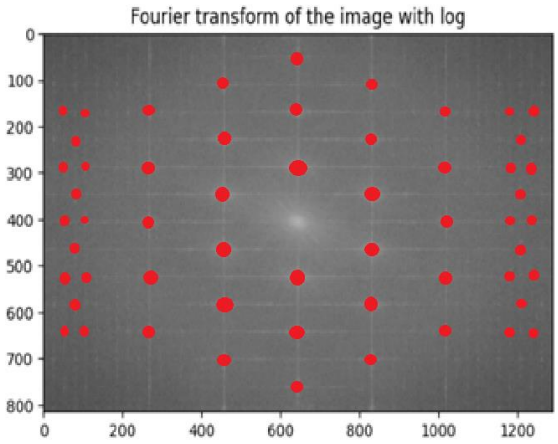

real part of Fourier transform of the image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


imag part of Fourier transform of the image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


Fourier transform of the image with log
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

نویز ها مقادیر با فرکانس بالا هستند که در تصویر بالا با رنگ سفید دیده میشوند البته به جز نقطه ی وسط که نقطه ی صفر تصویر است.

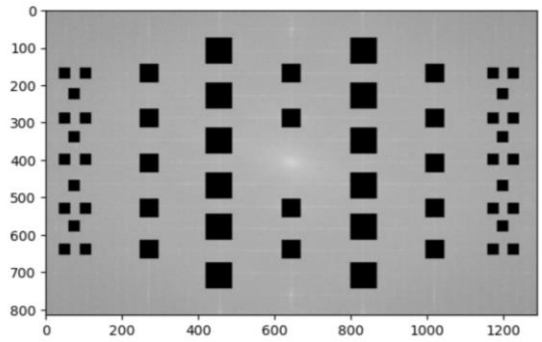
خطوطی که بنظرم نویز هستند در تصویر زیر با رنگ قرمز مشخص کردم و باید آن ها را در تصویر صفر کنیم.



با دقت در تصویر بالا مختصات نقاط را پیدا کردم و روشنایی این نقاط را با شعاع های مختلف نزدیک صفر گذاشتم یعنی عدد 0.00001 که برای لگاریتم گیری هنگام تصویر پس از تغییرات مشکلی پیش نیاید. برای نقاط خیلی دور که پرتو نوری کمی از آن ها ساتع شده کمترین شعاع و برای نقاط نزدیکتر به مرکز جهت حذف نشدن جزئیات تصویر فرکانس متوسط و برای نقاط با پرتو نوری بیشتر و فاصله ی متوسط از مرکز بیشترین شعاع را درنظر گرفتم.

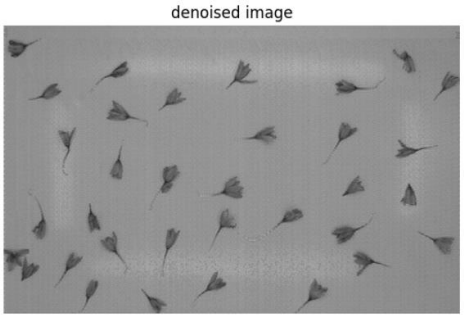
```
In [79]: #for smallest points:
         r1 = 15

         x1 = [50 , 105, 1175, 1228]
         y1 = [640, 531, 400, 290, 170]

         x2 = [75, 1200]
         y2 = [225, 340, 470, 578]

         for i in x1:
             for j in y1:
                 denoisedshift[j - r1 : j + r1, i - r1 : i + r1] = 0.00001

         for i in x2:
             for j in y2:
                 denoisedshift[j - r1 : j + r1, i - r1 : i + r1] = 0.00001

         #for smaller points:
         r2 = 25

         x3 = [272, 645, 1022]
         y3 = [640, 530, 290, 170]

         x4 = [272, 1022]
         y4 = [410]

         r3 = 35

         x5 = [455, 835]
         y5 = [710, 580, 470, 350, 230, 110]

         for i in x3:
             for j in y3:
                 denoisedshift[j - r2 : j + r2, i - r2 : i + r2] = 0.00001

         for i in x4:
             for j in y4:
                 denoisedshift[j - r2 : j + r2, i - r2 : i + r2] = 0.00001

         for i in x5:
             for j in y5:
                 denoisedshift[j - r3 : j + r3, i - r3 : i + r3] = 0.00001

         denoisedshow2 = 20 * np.log(np.abs(denoisedshift))
         plt.imshow(denoisedshow2, cmap = 'gray')
         plt.show()

         denoisedimage = np.fft.ifftshift(denoisedshift)
         denoisedimage = np.real(np.fft.ifft2(denoisedimage))
```

خروجی کد بالا بعد حذف نویز به شکل زیر میشود.



پس از آن باید معکوس فوریه را روی ماتریس اعمال کنیم و آن را رسم کنیم تا تصویر بدن نویز حاصل شود.

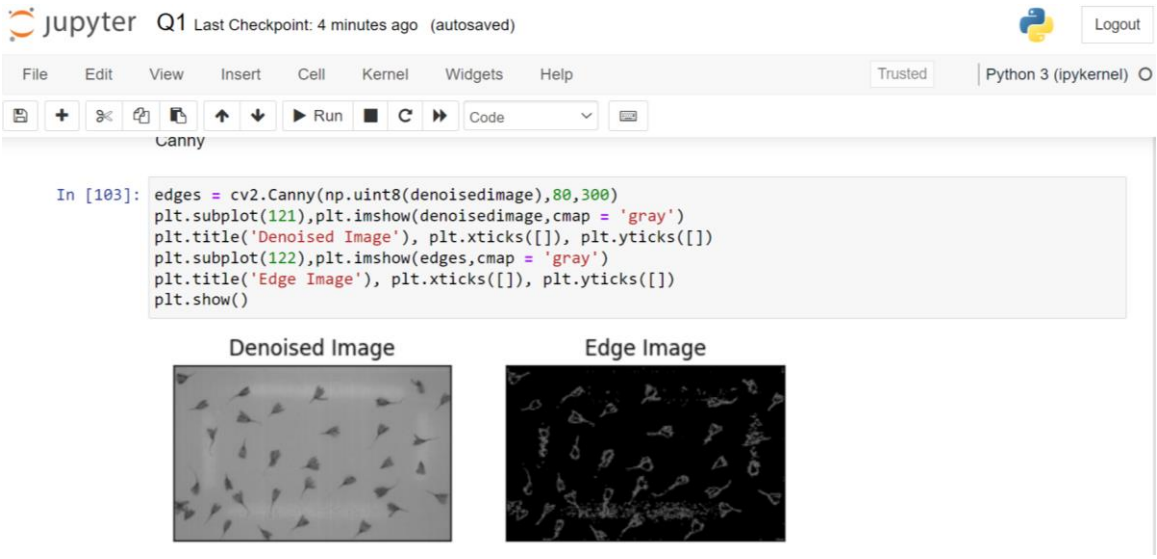
```
plt.imshow(denoisedimage, cmap='gray')
plt.title("denoised image")
plt.axis('off')
plt.show()
```



سوال اول) ب: لبه یاب Canny:

لینک استفاده شده: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

اولین آرگومان تصویر ورودی ما است. آرگومان های دوم و سوم به ترتیب minVal و maxVal ما هستند. آرگومان چهارم aperture_size است. اندازه هسته Sobel است که برای یافتن گرادیان های تصویر استفاده می شود. به طور پیش فرض 3 است. آخرین آرگومان L2gradient است که معادله را برای یافتن قدر گرادیان مشخص می کند. اگر درست باشد، از معادله ذکر شده در بالا استفاده می کند که دقیق تر است، در غیر این صورت از این تابع استفاده می کند: .Edge_Gradient(G)=|Gx|+|Gy|به طور پیش فرض، False است.

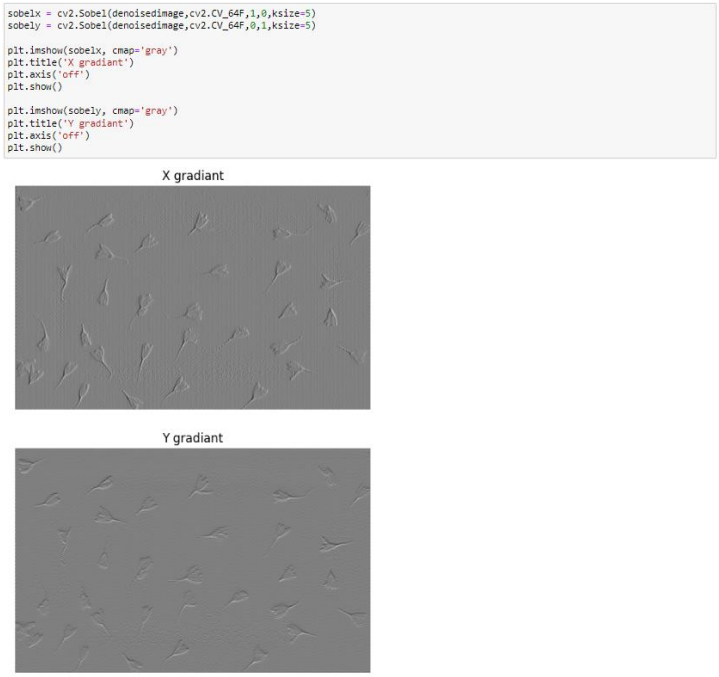


تصویر سمت راست خروجی canny میباشد که در آن گلهای زعفران مشخص هستند.

سوال اول) ج: محاسبه ی جهت گرادیان تصویر:

لینک های استفاده شده: https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html

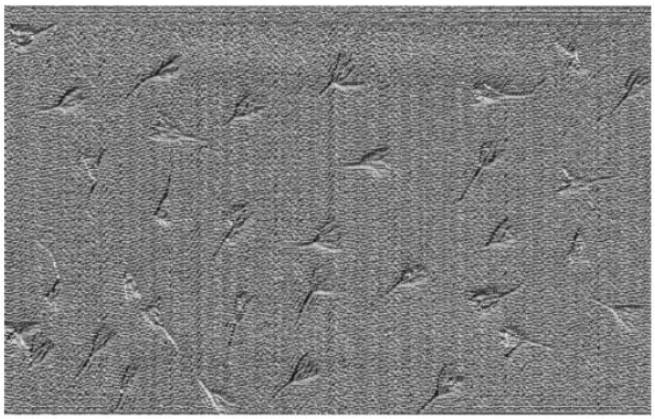
برای اینکار گرادیان را هم در جهت xو هم در جهت y حساب میکنیم چون برای محاسبه ی جهت گرادیان به آن نیاز داریم با استفاده از لاپلاسین هم میشود گرادیان را تصویر را حساب کرد اما پیداکردن جهت سخت تر میشود.



در مرحله ی بعد باید با عملگر arctan2 جهت گرادیان را پیدا کنیم.

```
gradient_oreintation = np.arctan2(sobely , sobelx)
gradient_oreintation = gradient_oreintation * 180 / np.pi

plt.imshow(gradient_oreintation, cmap='gray')
plt.axis('off')
```

```
(-0.5, 1289.5, 814.5, -0.5)
```



همانطور که در بالا مشخص است ابتدا زوایای تصویر را در فضای رادیان بدست آوردیم سپس برای فهم مقیاس درست تر آن را به درجه تبدیل کردیم. در نهایت نتایج را رسم کردیم. در جاهایی که تغییر رنگ وجود نداشته باشد گرادیان صفر شده و رنگ آن نواحی سیاه است.

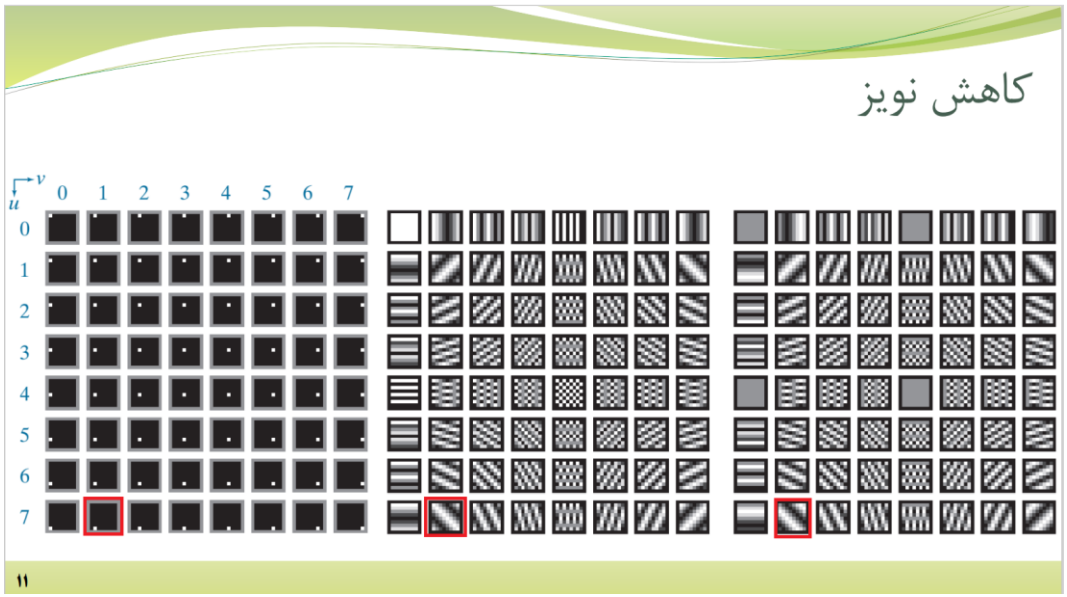
سوال اول) د: راه حل برای برش گلبرگ از ساقه:

اگر به تصویر دقت کنیم پیکسل های روی ستقه ی گلبرگ دارای روشن ترین مقادیر هستند و جهت گرادین هم از سمت روشن به تیره هست یعنی اطراف ساقه از خود ساقه تیره تر هست برای همین در مرز های ساقه با محیط اطراف جهت گرادیان از بیرون به داخل ساقه و عمود بر آن است و هرچه به محل برخورد یا لبه ی گلبرگ ها و ساقه نزدیک تر میشویم رنگ ها تیره تر میشوند و بعد سیاه میشوند و در نهایت گلبرگ ها چون بنفش هستن باز هم از محل لبه تا انتهای گلبرگ ها رنگ ها روشن تر هستند.

این مسئله رو میتوان مشابه با پیدا کردن یک خط دید چون تقریبا ساقه ها با تقریب خوبی صاف هستند و شبیه پاره خط هستند که گرادیان های یک طرف این ساقه با هم موازی و بر خود آن عمود هستند و در نقطه ی برش چون این نقطه رنگ سیاه دارد جهت گرادیان از داخل ساقه به سمت این لبه است. الگوریتم LSD میتواند با استفاده از رای گیری بین گرادین ها که در کدام قسمت این شرایط برقرار است و جهت گرادیان های موازی تغییر میکند، به ما کمک کند. در واقع یک سر این پاره خط که جهت گرادیان با شدت بیشتر تغییر کرده میشود محل جدایی گلبرگ و ساقه که پیدا کردن آن با تکنیک رای گیری بین نقاط انجام میشود.

سوال دوم) پیدا کردن طیف فرکانسی تصویر:

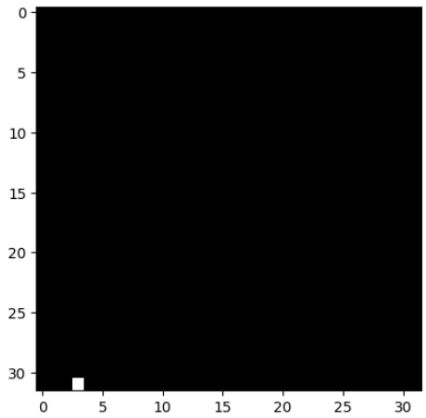
با توجه به اسلاید زیر در پاورپینت هفتم میتوان حدس زد طیف فرکانسی تصویر داده شده به چه شکل است. فرکانس هایی که دور آن ها خط قرمز کشیدم مشابه تصویر داده شده هستند اما چون تعداد پیکسل ها ی تصویر داده شده ۸ نیست با توجه به فرمول ها و تبدیل ها آن را خودمان بدست آورده و نمایش میدهیم.



با توجه به ابعاد داده شده در صورت سوال ابتدا سعی میکنیم تصویر را در نوتبوک رسم کنیم. میدانیم رنگ مشکی برای پیکسل ها با مقدار ۰ و رنگ سفید دارای بیشترین مقدار یعنی ۲۵۵ است و ابعاد تصویر ۳۲ در ۳۲ است.

کد زیر این تصویر را تولید میکند:

```
m = 32
n = 32
im = np.zeros((m, n))
#مختصات نقطه ی سفید که با سعی و خطا بدست آمد
im[31 , 3] = 255
plt.imshow(im, cmap = "gray")
plt.show()
```



حالا باید طبق خواسته ی سوال طیف فرکانسی این تصویر را پیدا کنیم.

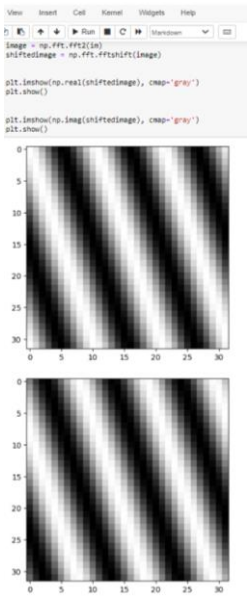
برای اینکار از فرمول تبدیل فوریه استفاده میکنیم تا فضای حالت را تغییر دهیم و از مکان به فرکانس برسیم.

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

مقادیر $f(x,y)$ به جز در نقطه ی (۳۱و۳) صفر هستند. با جایگذاری در رابطه خواهیم داشت:

$$F(u,v) = 255.e^{-j2\pi(\frac{u.3}{32}+\frac{v.31}{32})}$$

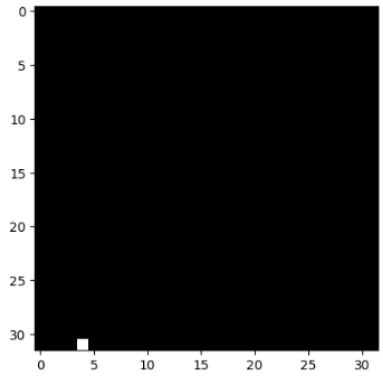
با توجه به اینکه حاصل عددی مختلط است برای راحت تر دیدن ضرایب در دو نمودار مختلف قسمت حقیقی و موهومی را جدا میکنیم.



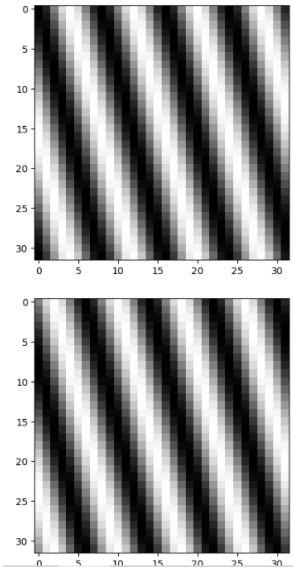
تصویر بالا برای بخش حقیقی و تصویر پایین برای بخش موهومی آن است که هر دو شبیه به هم هستند.

برای قسمت دوم سوال همین مراحل را برای تصویری که پیکسل (۳۱و۴) در آن روشن است تکرار میکنیم.

```
m = 32
n = 32
im2 = np.zeros((m, n))
im2[31 , 4] = 255
plt.imshow(im2, cmap = "gray")
plt.show()
```



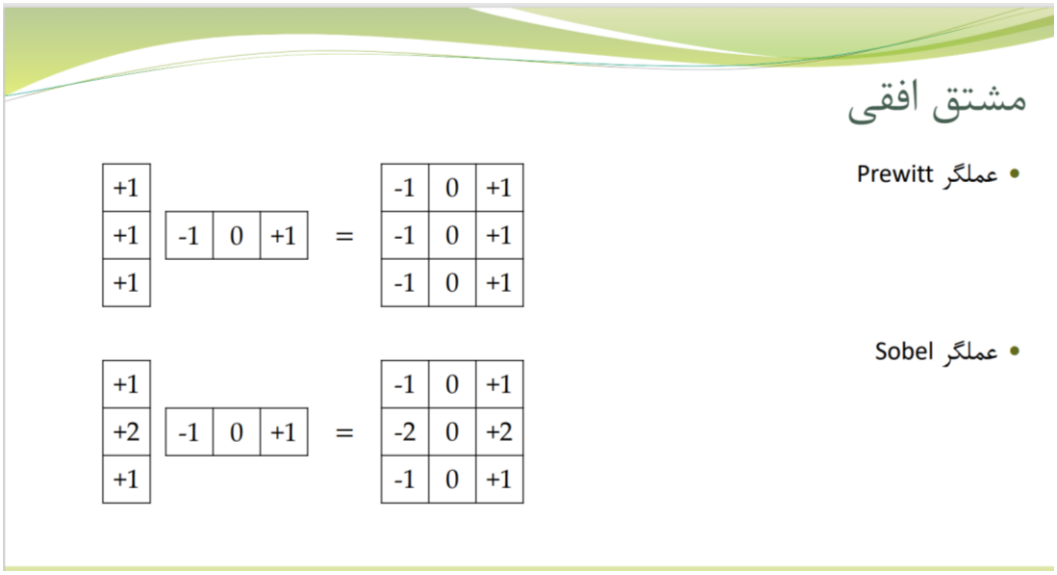
```
image2 = np.fft.fft2(im2)
shiftedimage2 = np.fft.fftshift(image2)


plt.imshow(np.real(shiftedimage2), cmap='gray')
plt.show()


plt.imshow(np.imag(shiftedimage2), cmap='gray')
plt.show()
```



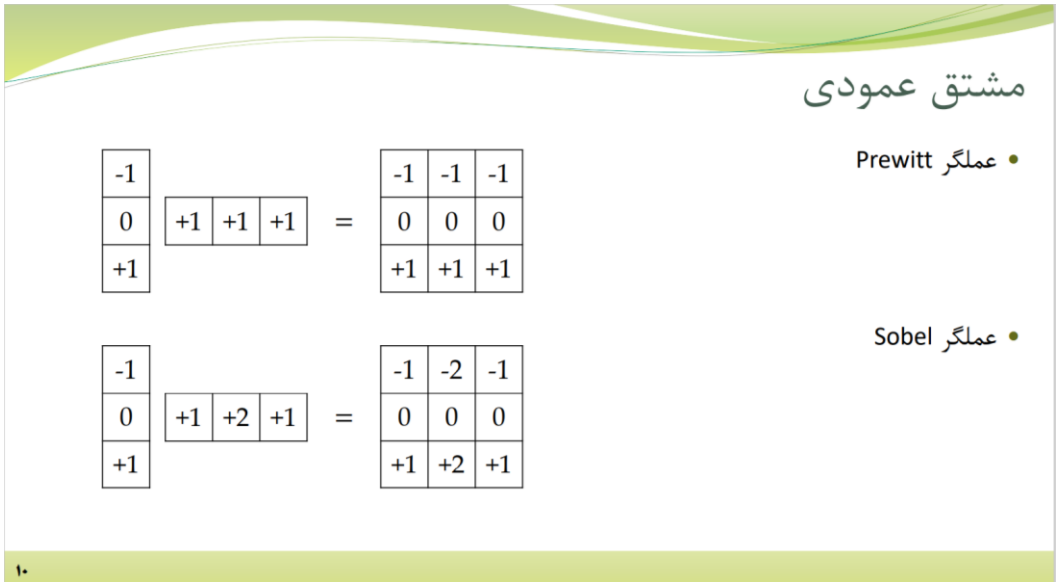
بخش حقیق تصویر بالا و بخش حقیقی تصویر پایین است که همانطور که مشخص است با هم اندکی تفاوت دارند. تصویر پایین همان تصویر بالاست دو واحد به چپ انتقال پیدا کرده است.

سوال سوم)اعمال عملگر Sobel روی یک ماتریس کوچک:

لینک استفاده شده: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html



عملگر Sobel از کرنل نشان داده شده در اسلاید بالا برای مشتق گیری افقی از تصویر استفاده میکند. در واقع نوعی میانگین گیر وزن دار برای حفظ لبه هاست چون ضریب بیشتری به پیکسل های میانی میدهد.



از کرنل بالا هم برای مشتق عمودی تصویر استفاده میکند.

یک آرایه ی numpy میسازیم که تمامی مقادیر آن به جز مقادیر ستون وسط یعنی خانه هایی که x=2 دارند همگی صفر باشند و مقادیر این ستون وسط ۲۵۵ باشد. و آن را نمایش میدهیم.

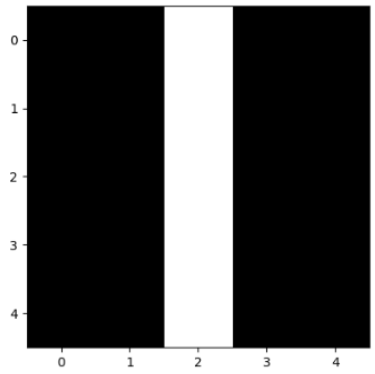
Setup

```
%matplotlib inline
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
```

Q3

```
image = np.zeros((5, 5))
image[0 : 5 , 2] = 255
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x20f70a9cc40>
```

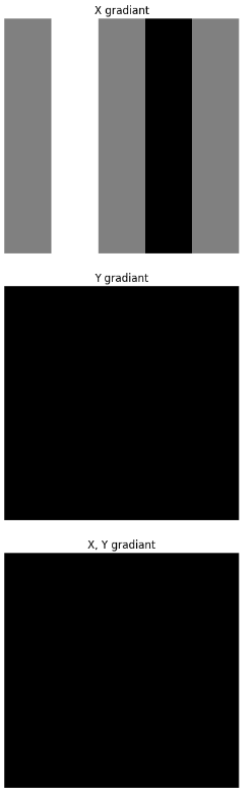


این ستون وسط همان لبه ی خواسته شده است چون با دو طرف خودش شدت روشنایی خیلی متفاوتی دارد.

با استفاده از کد زیر Sobel در راستای x,y و در هر دو بعد را به دست آورده و رسم میکینم.

```python
sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)

plt.imshow(sobelx, cmap='gray')
plt.title('X gradiant')
plt.axis('off')
plt.show()

sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

plt.imshow(sobely, cmap='gray')
plt.title('Y gradiant')
plt.axis('off')
plt.show()

sobel2D = cv2.Sobel(image, cv2.CV_64F, 1, 1, ksize=3)

plt.imshow(sobel2D, cmap='gray')
plt.title('X, Y gradiant')
plt.axis('off')
plt.show()
```

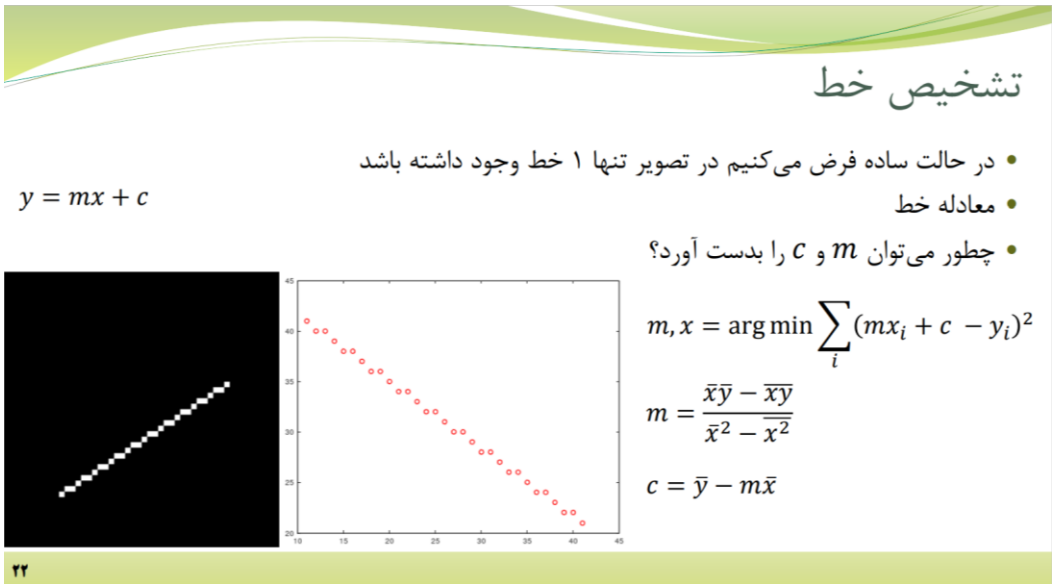
خروجی کد به شکل زیر خواهد بود:



قابل مشاهده است که در جهت عمودی تغییراتی نداریم برای همین مشتق همه ی نقاط صفر شد.

سوال چهارم) بدست آوردن معادله خط:

منابع استفاده شده: اسلاید های انتهایی پاورپینت هشتم



در تصویر داده شده هم فقط یک خط وجو دارد برای همین به راحتی میتوان از فرمول های بالا استفاده کرد.

Q4

```
#نمایش تصویر
im = cv2.imread('images/img_02.jpg', 0 )
plt.imshow(im, cmap='gray')
plt.show()

s = im.shape
m = s[0]
n = s[1]

#پیدا کردن نقاط غیر سفید
points = [(y, x) for x in range(n) for y in range(m) if im[y, x] != 255]
line = []
for i in range(n):
    for j in range(m):
        if im[j, i] != 255:
            line.append((j, i))

Y = []
X = []
for o in line:
    X.append(o[0])
    Y.append(o[1])

Y_m = np.mean(Y)
X_m = np.mean(X)

s1 = np.multiply(Y, X)
XY_m = np.mean(s1)

X_m_2 = X_m ** 2

s2 = np.power(X, 2)
X_2_m = np.mean(s2)

m = np.float64((Y_m * X_m - XY_m) / (X_m_2 - X_2_m))

c = Y_m - m * X_m

print('y = (' , m , ')x + (' , c, ')' )

x = np.linspace(0, m)

y = m * x + c

plt.plot(x, y, '-r', label=f'y=({m})x+({c})', linewidth=1)

plt.title('line')

plt.xlabel('x')

plt.ylabel('y')

plt.legend(loc='upper left')

plt.imshow(im)

plt.show()
```

خروجی کد به شکل زیر خواهد بود که اول ضرایب را پیدا کرده و سپس آن را کشیدیم و روی تصویر انداختیم.