Cornelio, Andrew HW3

March 24, 2020

1 Mathematical Image Analysis (553.493)

Cornelio, Andrew

March 24, 2020

[1]:
```
import cv2
import numpy as np
import math
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from scipy import ndimage, misc, signal
from scipy.ndimage.interpolation import shift
import argparse
import itertools
from PIL import Image
```

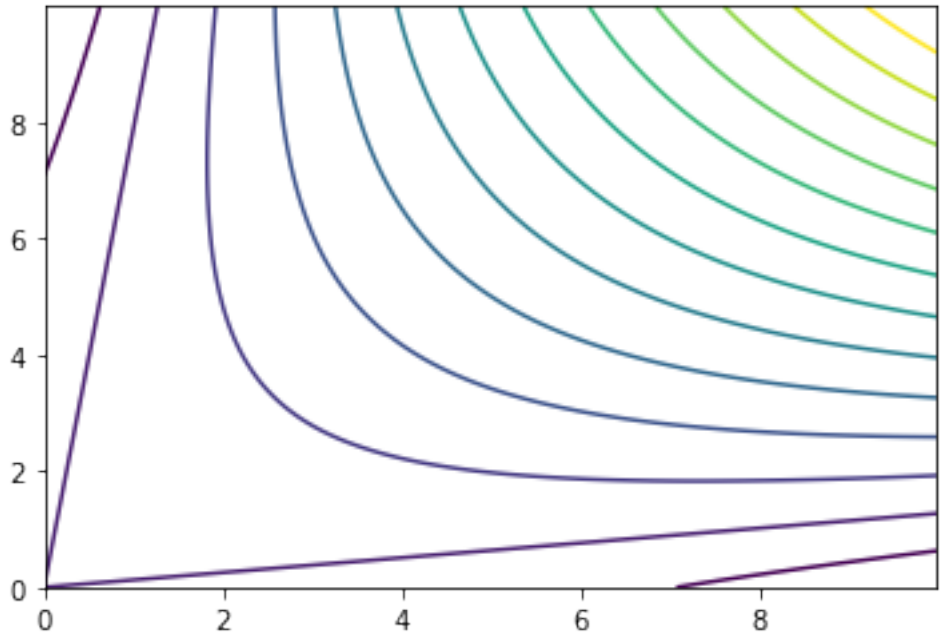
[2]:
```
poly1 = Image.open('image-polygons.gif')
poly2 = np.array(poly1.getdata()).reshape(poly1.size[0], poly1.size[1])
image_polygons = poly2.astype(np.uint64);

house1 = Image.open('image-house.gif')
house2 = np.array(house1.getdata()).reshape(house1.size[0], house1.size[1])
image_house = house2.astype(np.uint64);

image_zebra = cv2.imread('image-zebra.png', 0).astype(np.int64);
image_box = cv2.imread('box.jpg', 0).astype(np.int64);
image_tiles = cv2.imread('image-tiles.jpg', 0).astype(np.int64);
```

1.0.1 Problem 3.1.a

We can use the matplotlib's contour function to plot the 3D function.

[3]:
```
x = np.arange(0, 10, 0.01)
y = np.arange(0, 10, 0.01)
X, Y = np.meshgrid(x, y)
```

```
Z = X*Y - 0.1*(X+Y)**2
levels = np.arange(-5, 60, 5)

# Basic contour plot
fig, ax = plt.subplots()
CS = ax.contour(X, Y, Z, levels)
```



1.0.2 Problem 3.1.b

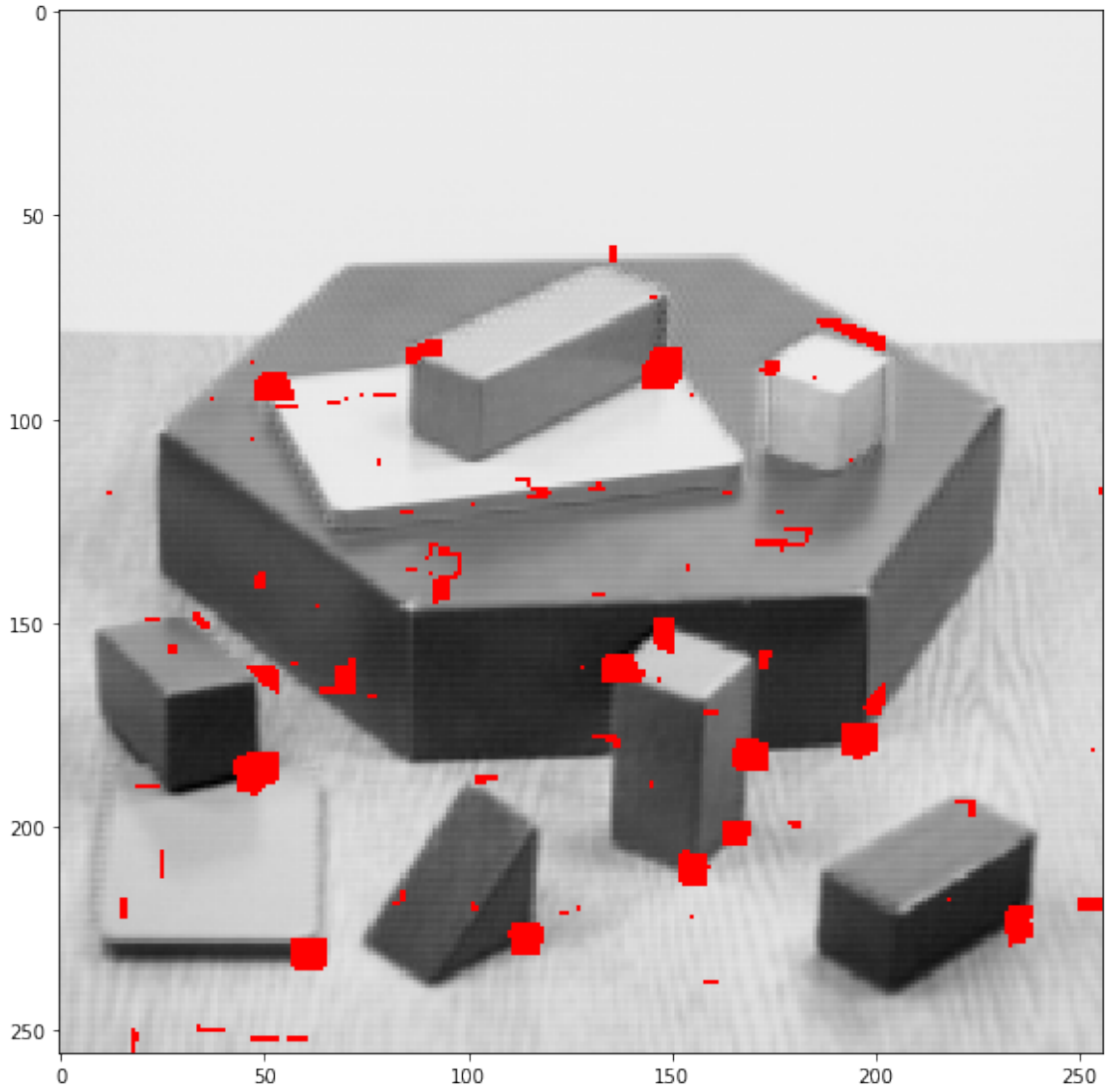
We first define the harris function, which calculates the gradient, and then the values in the local structure matrix by applying a uniform filter to the product of combination sof the gradient values. Then it calculate the determinant and trace of the local structure matrix. It then calculates and thresholds the corner respose function for each pixel in the image. Finally it converts the image from greyscale to color and applies the thresholded corner response function.

We apply the function to four images. We can see that the function works better if the image is simple with 90 degree corners of distinct colors from the background. For example, it works thebest with the third image, which is the most simple of the images, and it utterly fails on the fourth image, which has the highest quality and hence the most noise/complexity/texture. Notice that it also doesn't recognize the non 90degree corners in the polygon image.

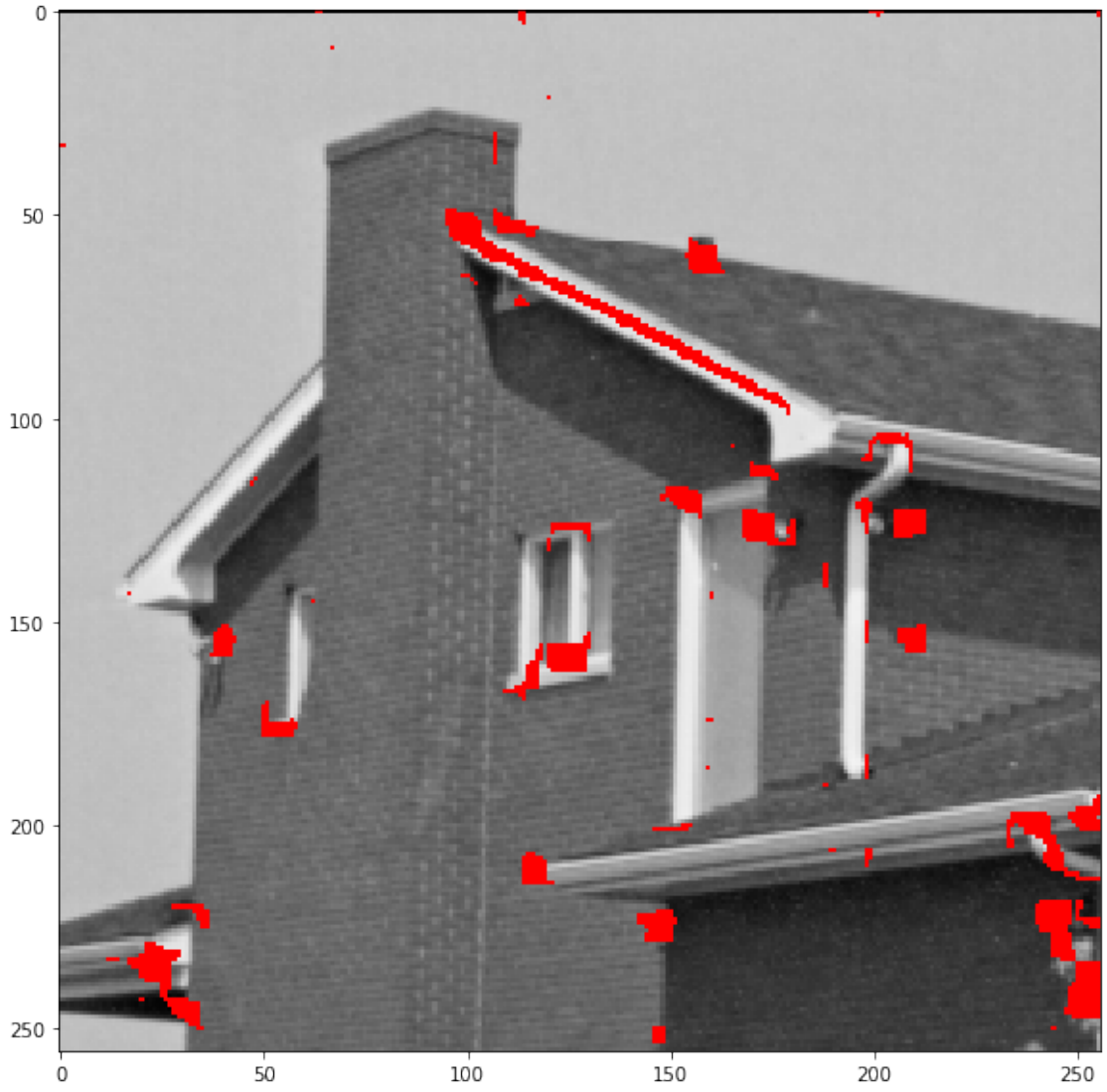
[4]:
```
def harris(img, win_sze, alph, t):
    # calculate the gradient
    Ix = ndimage.sobel(img, axis=0)
    Iy = ndimage.sobel(img, axis=1)
```

```
    # calculate the local structure matrix values
    Ixx = ndimage.uniform_filter(Ix * Ix, size=win_sze)
    Ixy = ndimage.uniform_filter(Ix * Iy, size=win_sze)
    Iyy = ndimage.uniform_filter(Iy * Iy, size=win_sze)

    # determinant & trace of local structure matrix
    detM = Ixx * Iyy - Ixy ** 2
    traceM = Ixx + Iyy

    # corner response function above threshold
    cnr_resp = np.where(detM - alph * traceM ** 2 >= t)

    #convert grey scale to color and apply mask
    img_col = cv2.cvtColor(img.astype(np.uint8),cv2.COLOR_GRAY2RGB)
    img_col[cnr_resp[0], cnr_resp[1], :] = [255, 0, 0]

    # return masked image
    return img_col
```

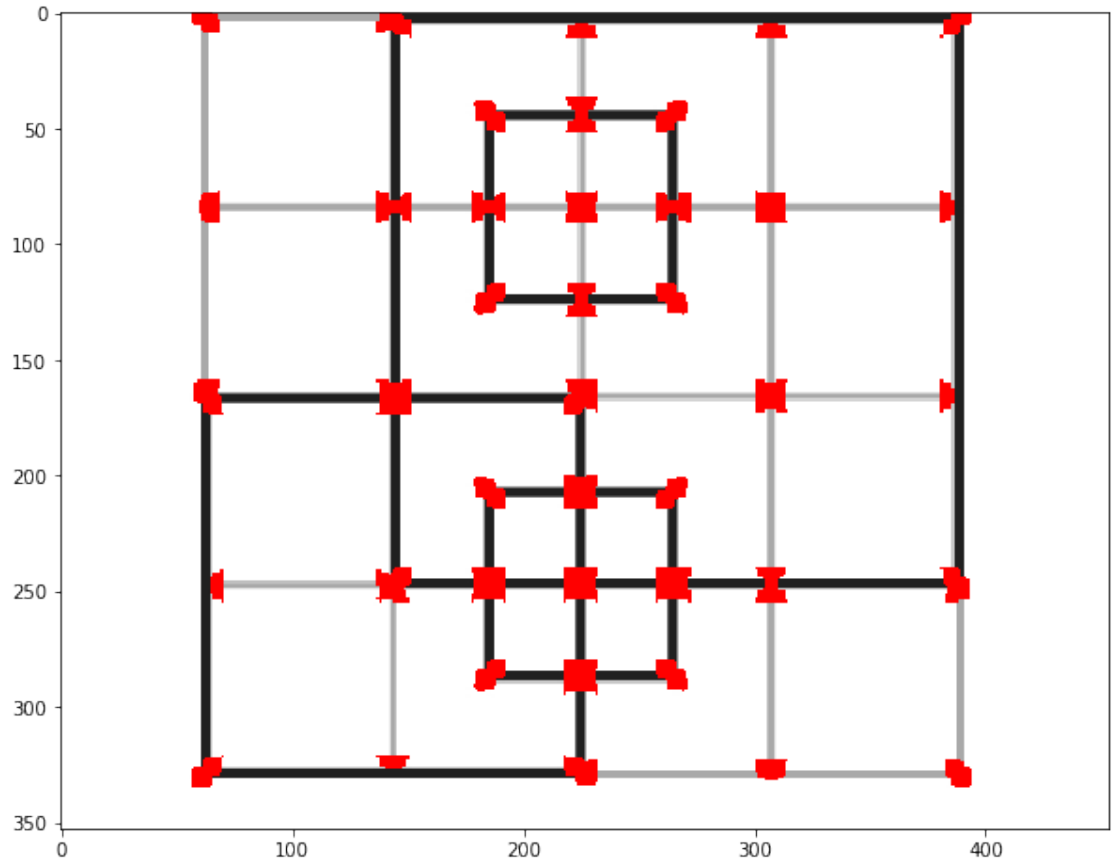
[5]:
```
resp = harris(image_polygons, 9, .05, 10 ** 7)
fig,ax = plt.subplots(1)
ax.imshow(resp)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```



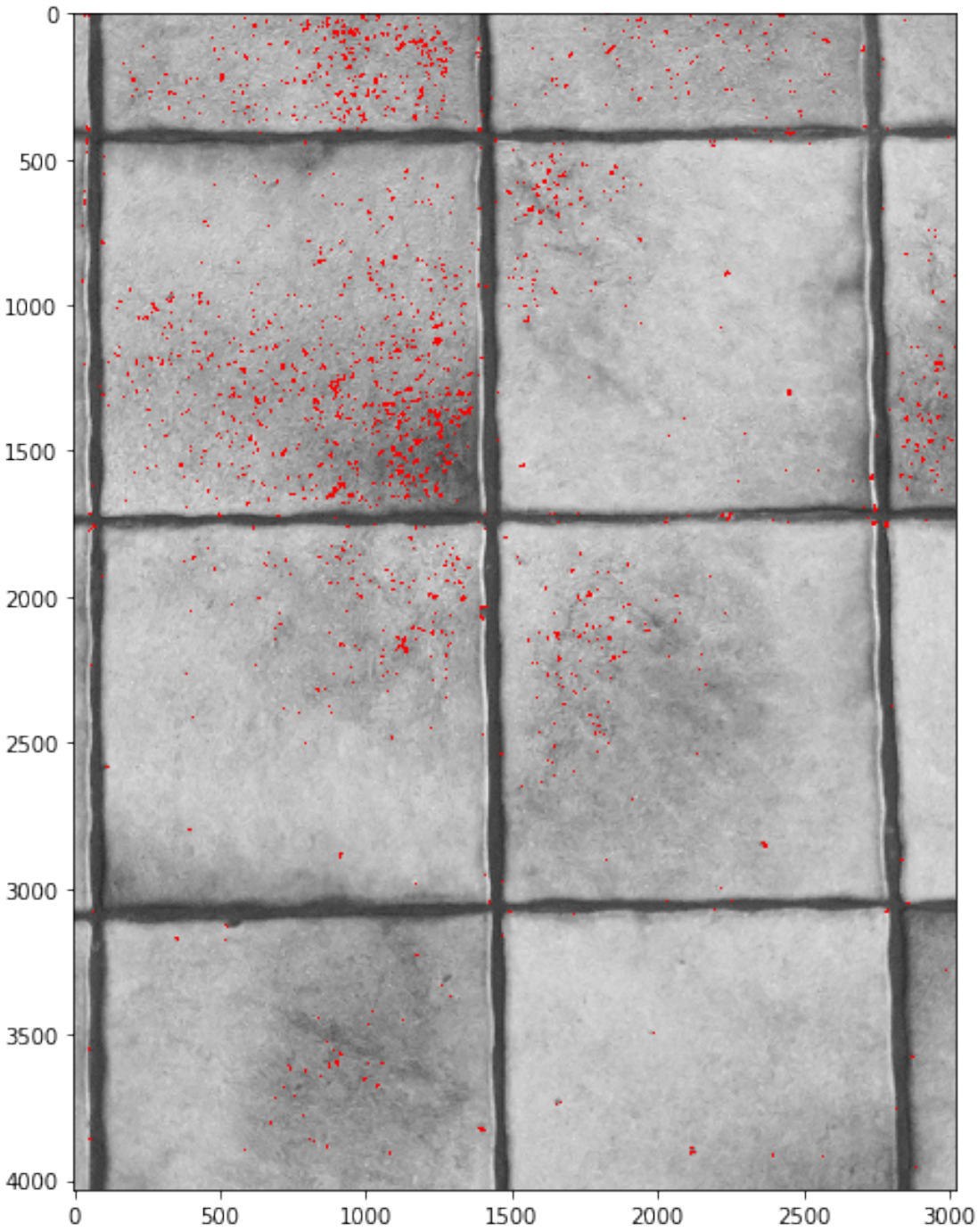
[6]:
```
resp = harris(image_house, 9, .15, 10 ** 6)
fig,ax = plt.subplots(1)
ax.imshow(resp)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```



[7]:
```
resp = harris(image_box, 9, .05, 10 ** 6)
fig,ax = plt.subplots(1)
ax.imshow(resp)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```



[8]:
```
resp = harris(image_tiles, 10, .05, 10 ** 6)
fig,ax = plt.subplots(1)
ax.imshow(resp)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```



1.0.3 Problem 3.2.a

We use scikit's convolve function to calculate the 2d convolution of $h_L * g$

[9]:
```
hl = np.array([[0, 1, 0],
               [1, -4, 1],
               [0, 1, 0]])

g =  1/16 * np.array([[1, 2, 1],
                      [2, 4, 2],
                      [1, 2, 1]])

hLoG = signal.convolve2d(hl, g)
print(hLoG)
```

```
[[ 0.      0.0625  0.125   0.0625  0.    ]
 [ 0.0625  0.     -0.125   0.      0.0625]
 [ 0.125  -0.125  -0.5    -0.125   0.125 ]
 [ 0.0625  0.     -0.125   0.      0.0625]
 [ 0.      0.0625  0.125   0.0625  0.    ]]
```

1.0.4 Problem 3.2.b

First we define a function that counts the number of valid crossings to make sure it is at least 2. Valid crossings must satisfy the properties: 1)

$$A \times B < 0$$

and

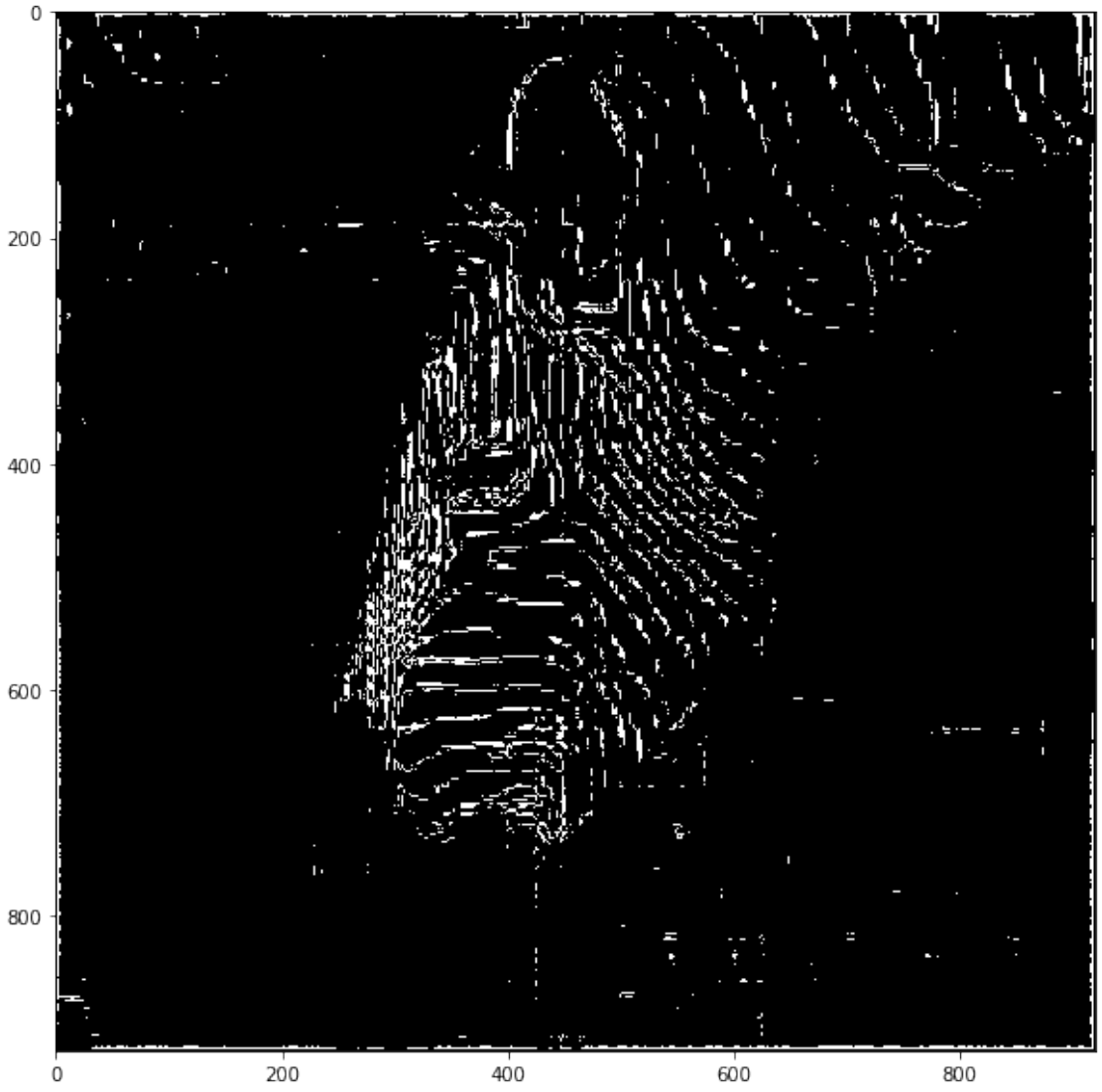
$$|A - B| > t$$

Where $t \in \mathbb{Z}$ is a threshold. This is implemented in the crossing function and the crosssing helper function. Note that the crossings could either be calculated using convolving the image with special filters, or by simply shifting the images by a row or column. I chose the latter. Finally, we then apply the laplacian of gaussian filter to the image and threshold with the crossing function.

[10]:
```
def crossing_help(img, c, t):
    rows = np.size(img,0)
    cols = np.size(img,1)

    img2 = img
    img1 = img

    if c == 0:    # bottom - top
        img2 = np.pad(img,((0,1),(0,0)), mode='constant')[1:rows+1, 0:cols]
        img1 = np.pad(img,((1,0),(0,0)), mode='constant')[0:rows, 0:cols]
    if c == 1:    # right - left
        img2 = np.pad(img,((0,0),(0,1)), mode='constant')[0:rows, 1:cols+1]
        img1 = np.pad(img,((0,0),(1,0)), mode='constant')[0:rows, 0:cols]
    if c == 2:    # lower right - upper left
        img2 = np.pad(img,((0,1),(0,1)), mode='constant')[1:rows+1, 1:cols+1]
        img1 = np.pad(img,((1,0),(1,0)), mode='constant')[0:rows, 0:cols]
    if c == 3:    # lower left - upper right
```

```
        img2 = np.pad(img,((0,1),(1,0)), mode='constant')[1:rows+1, 0:cols]
        img1 = np.pad(img,((1,0),(0,1)), mode='constant')[0:rows, 1:cols+1]

    img_cross = np.where(img2 * img1 < 0, 1, 0)       # crossed 0
    img_diff = np.where(abs(img2 - img1) > t, 1, 0)   # abs of diff is greater
 ↪than threshold

    return img_cross * img_diff    # contains binary data about valid crossings


def crossing(img, t):
    img_c0 = crossing_help(img, 0, t)
    img_c1 = crossing_help(img, 1, t)
    img_c2 = crossing_help(img, 2, t)
    img_c3 = crossing_help(img, 3, t)

    num_cross = np.where(img_c0 + img_c1 + img_c2 + img_c3 >= 2, 255, 0)
    return num_cross
```

[11]:
```
zeb = signal.convolve2d(image_zebra, hLoG)
fig,ax = plt.subplots(1)
ax.imshow(crossing(zeb, 20), cmap='gray', vmin=0, vmax=255)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10, 10)
```

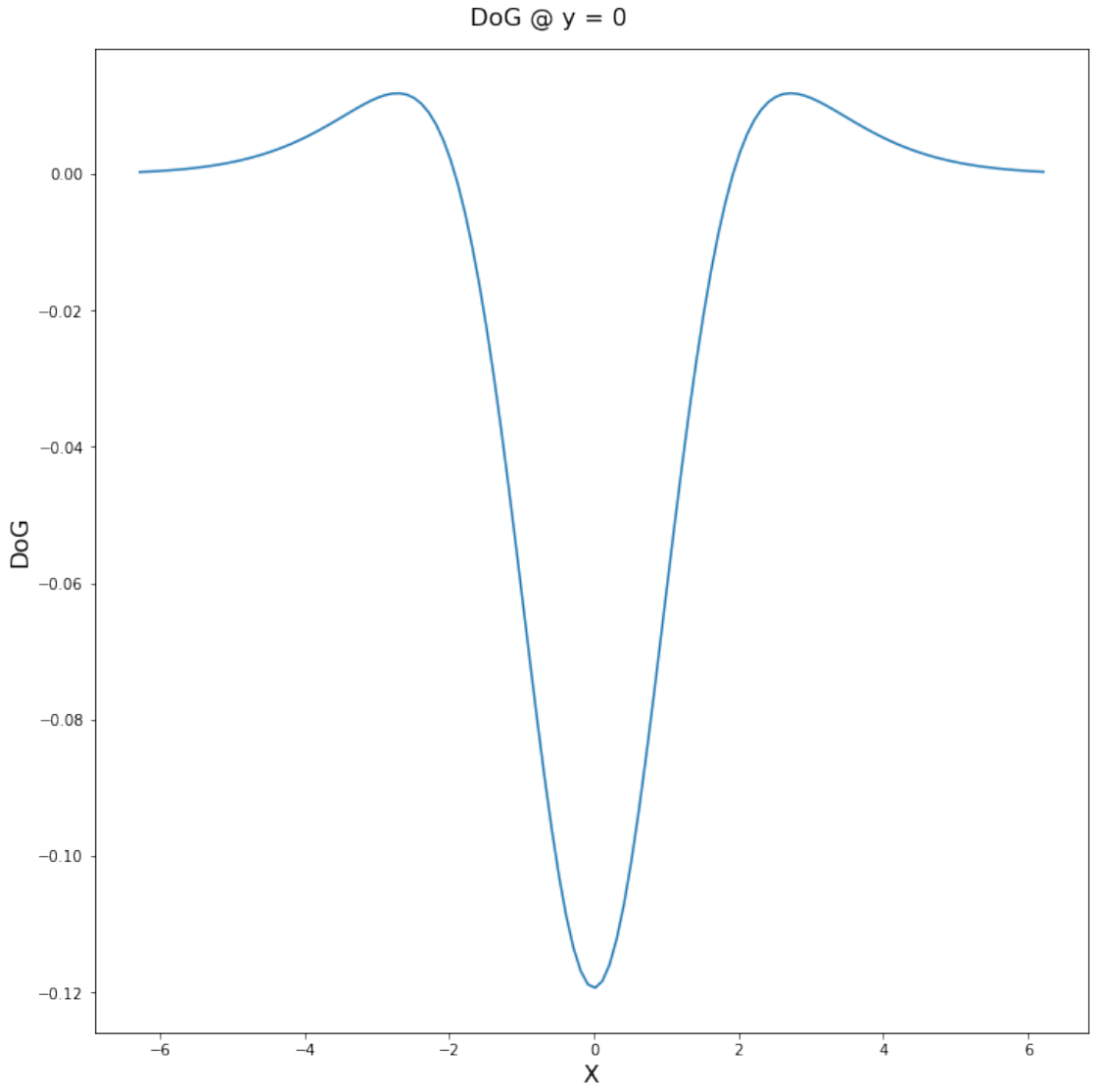


1.0.5 Problem 3.2.c

First we plot the $d(x, 0)$ with $\sigma_1 = 2$ and $\sigma_2 = 1$. Note that the graph axes are not equal. This is to exagerate the shape to make it easier to see.

[12]:
```
X = np.arange(-2*math.pi,2*math.pi,0.1);
d = 1/(2*math.pi*4)*np.exp(-0.5*X**2/4) - 1/(2*math.pi)*np.exp(-0.5*X**2)

fig, axs = plt.subplots(figsize=(10, 10), constrained_layout=True)
fig.suptitle('DoG @ y = 0', fontsize=16)

axs.plot(X, d)
```

```
axs.set_xlabel('X',fontsize = 16) #xlabel
axs.set_ylabel('DoG', fontsize = 16)#ylabel

plt.show()
```



To verify that the zero crossings are the same, first we must solve for x at σ_1 and σ_2 at the zero crossings of the $d(x, 0)$. We will consider the cross section at $y = 0$.

$$\frac{1}{2\pi\sigma_1^2}e^{-\frac{1}{2}\frac{x^2}{\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2}e^{-\frac{1}{2}\frac{x^2}{\sigma_2^2}} = 0$$
$$e^{\frac{1}{2}\frac{x^2}{\sigma_2^2}-\frac{1}{2}\frac{x^2}{\sigma_1^2}} = \frac{\sigma_1^2}{\sigma_2^2}$$
$$\frac{1}{2}\frac{x^2}{\sigma_2^2} - \frac{1}{2}\frac{x^2}{\sigma_1^2} = ln\left(\frac{\sigma_1^2}{\sigma_2^2}\right)$$
$$x^2\left(\frac{1}{\sigma_2^2} - \frac{1}{\sigma_1^2}\right) = 2ln\left(\frac{\sigma_1^2}{\sigma_2^2}\right)$$
$$x^2 = \frac{2\sigma_1^2\sigma_2^2}{\sigma_1^2 - \sigma_2^2}ln\left(\frac{\sigma_1^2}{\sigma_2^2}\right)$$

Now we do the same for the LoG. We notice that the exponential is never equal to 0, hence we can divide it out:

$$\frac{x^2 - 2\sigma^2}{\sigma^4}\frac{1}{2\pi\sigma^2}e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} = 0$$
$$x^2 - 2\sigma^2 = 0$$
$$x^2 = 2\sigma^2$$

Hence we see that if we wish for the LoG and $d(x, 0)$ to have the same zero crossings, we must set:

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 - \sigma_2^2}ln\left(\frac{\sigma_1^2}{\sigma_2^2}\right)$$

Now that we have verified the substitution, we must find M. First we find the value of $LoG(0, 0)$. By simple substitution we obtain:

$$\begin{aligned} LoG(0,0) &= -\frac{2\sigma^2}{\sigma^4}\frac{1}{2\pi\sigma^2} \\ &= -\frac{1}{\pi\sigma^4} \end{aligned}$$

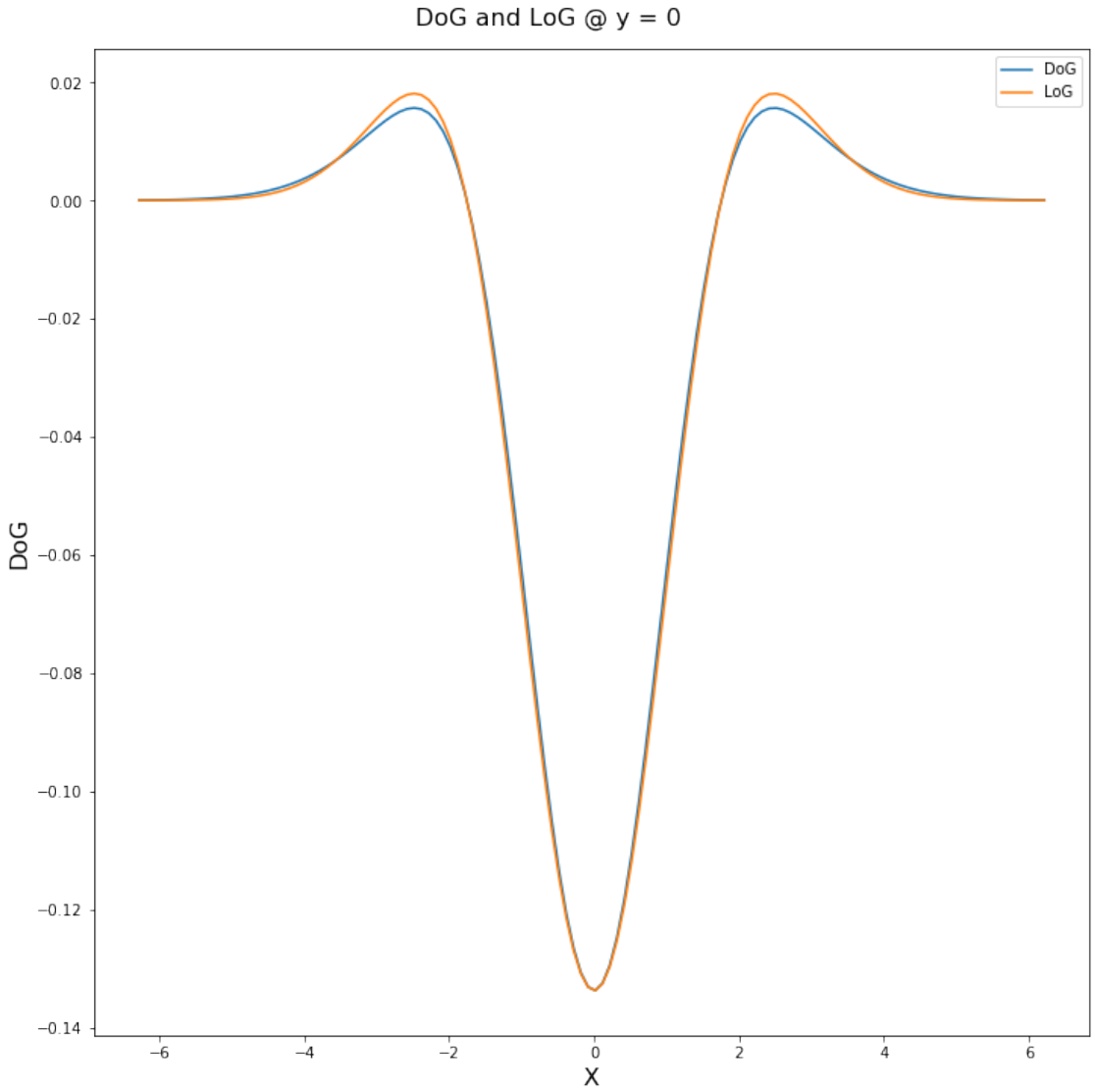
Now we find value of $d(0, 0)$.

$$\begin{aligned} d(0,0) &= \frac{1}{2\pi\sigma_1^2} - \frac{1}{2\pi\sigma_2^2} \\ &= \frac{1}{2\pi}\frac{\sigma_2^2 - \sigma_1^2}{\sigma_1^2\sigma_2^2} \end{aligned}$$

So, we see that M should be equal to:

$$\begin{aligned} M &= -\frac{2\pi}{\pi\sigma^4}\frac{\sigma_1^2\sigma_2^2}{\sigma_2^2 - \sigma_1^2} \\ &= -\frac{2}{\sigma^4}\frac{\sigma_1^2\sigma_2^2}{\sigma_2^2 - \sigma_1^2} \end{aligned}$$

[13]:
```
var1 = 1.6 ** 2
var2 = 1 ** 2
var = var1 * var2 / (var1 - var2) * np.log(var1 / var2)
M = - 2/var **2 * var1 * var2 / (var2 - var1)

X = np.arange(-2*math.pi,2*math.pi,0.1);
d = 1/(2*math.pi*var1)*np.exp(-0.5*X**2/var1) - 1/(2*math.pi*var2)*np.exp(-0.
↪5*X**2/var2)
DoG = M * d
LoG = (X ** 2 - 2 * var) / var ** 2 * 1/(2 * math.pi * var) * np.exp(-0.5*X**2/
↪var)

fig, axs = plt.subplots(figsize=(10, 10), constrained_layout=True)
fig.suptitle('DoG and LoG @ y = 0', fontsize=16)

axs.plot(X, DoG, label='DoG')
axs.plot(X, LoG, label='LoG')
axs.set_xlabel('X',fontsize = 16) #xlabel
axs.set_ylabel('DoG', fontsize = 16)#ylabel
axs.legend()

plt.show()
```



1.0.6 Problem 3.3

a) Let $s(t)$ and $g(t)$ be two even signals, and let $a, b, t \in \mathbb{R}$. Now let $h(t) = as(t) + bg(t)$. Since $s(t)$ is even, then for all $t \in \mathbb{R}$, $s(t) = s(-t)$. If this is true, then it should also be true that $as(t) = as(-t)$. Using the same reasoning, we can see that $bg(t) = bg(-t)$. Finally, adding these two equalities, we see that for all $t \in \mathbb{R}$, $as(t) + bg(t) = as(-t) + bg(-t)$. Hence, $h(t) = as(t) + bg(t)$ is even.

Now, let $s(t)$ and $g(t)$ be odd even signals, and let $a, b \in \mathbb{R}$. Also, let $h(t) = as(t) + bg(t)$. Since $s(t)$ is odd, then for all $t \in \mathbb{R}$, $-s(t) = s(-t)$. If this is true, then it should also be true that $-as(t) = as(-t)$. Using the same reasoning, we can see that $-bg(t) = bg(-t)$. Finally, adding these two equalities, we see that for all $t \in \mathbb{R}$, $-(as(t) + bg(t)) = as(-t) + bg(-t)$. Hence, $h(t) = as(t) + bg(t)$ is odd.

b) If a signal $s(t)$, $t \in \mathbb{R}$, that is simultaneously even and odd, then $s(t) = 0$ for all $t \neq 0$. We will prove this by contradiction. Suppose there exists some $x \in \mathbb{R}$ such that $x \neq 0$ and $s(x) \neq 0$. Then, since the signal is even, $s(x) = s(-x)$. However, because the signal is odd, $s(x) = -s(-x)$. However, this is a contradiction because it implies $s(-x) = -s(-x)$, whch is not true for all non zero real numbers. Hencewe have reached a contradiction, and we see $s(t) = 0$ for all $t \neq 0$. (Note that if the signal is continuous, then we can further say that $s(t) = 0$ for all $t \in \mathbb{R}$).

c) If a signal $s(t)$, $t \in \mathbb{R}$, then define the even component of the signal as $s_e(t) = \frac{1}{2}[s(t) + s(-t)]$ and the odd component of the signal as $s_o(t) = \frac{1}{2}[s(t) - s(-t)]$.

First we verify that $s_e(t)$ is even. We can chose an arbitrary $t \in \mathbb{R}$ and negate it. We see that $s_e(-t) = \frac{1}{2}[s(-t) + s(t)] = \frac{1}{2}[s(t) + s(-t)] = s_e(t)$. Hence $s_e(t)$ is indeed even.

Next we verify that $s_o(t)$ is even. We can chose an arbitrary $t \in \mathbb{R}$ and negate it. We see that $s_o(-t) = \frac{1}{2}[s(-t) - s(t)] = -\frac{1}{2}[s(t) - s(-t)] = -s_0(t)$. Hence $s_o(t)$ is indeed odd.

Finally, we can see from basic algebra that:

$$s_e(t) + s_o(t) = \frac{1}{2}[s(t) + s(-t)] + \frac{1}{2}[s(t) - s(-t)]$$

$$= \frac{1}{2}s(t) + \frac{1}{2}s(t) = s(t)$$

Hence, the even and odd parts of the add up to the original signal.

d) Assume a signal $s(t)$, $t \in \mathbb{R}$ satisfies the identity $s(t) = g_e(t) + g_o(t)$, where $g_e(t)$ is even and $g_o(t)$ is odd. This identity can hold if and only if $s(-t) = g_e(t) - g_o(t)$ by the definition of even and odd functions. Adding these two identities together, we see that the odd components cancel out and we are left with $s(t) + s(-t) = 2g_e(t)$. Rewriting this, we obtain $g_e(t) = \frac{1}{2}[s(t) + s(-t)]$. Similarly, subtracting the two identites and dividing by 2 yields $g_o(t) = \frac{1}{2}[s(t) + s(-t)]$. Since we only assumed that $g_e(t)$ and $g_o(t)$ were even and odd respectively, and that they summed to $s(t)$, and we were able to show that $g_e(t) = s_e(t)$ and $g_o(t) = s_o(t)$, it must be the case that $s_e(t)$ and $s_o(t)$ are unique a unique decomposition.

e) Assume a signal $s(t)$ is differentiable. If $s(t)$ is even, then we can represent $s(t)$ as $s(t) = \frac{1}{2}[s(t) + s(-t)]$. If we differentiate this identity, then we find by the chain rule that $s'(t) = \frac{1}{2}[s'(t) - s'(-t)]$. Since $s'(t)$ satisfies the identity for odd functions, we conclude that $s'(t)$ is odd.

Assume a signal $s(t)$ is differentiable. If $s(t)$ is odd, then we can represent $s(t)$ as $s(t) = \frac{1}{2}[s(t) - s(-t)]$. If we differentiate this identity, then we find by the chain rule that $s'(t) = \frac{1}{2}[s'(t) + s'(-t)]$. Since $s'(t)$ satisfies the identity for even functions, we conclude that $s'(t)$ is even.