CSE 5524 – Homework #1

9/5/2022

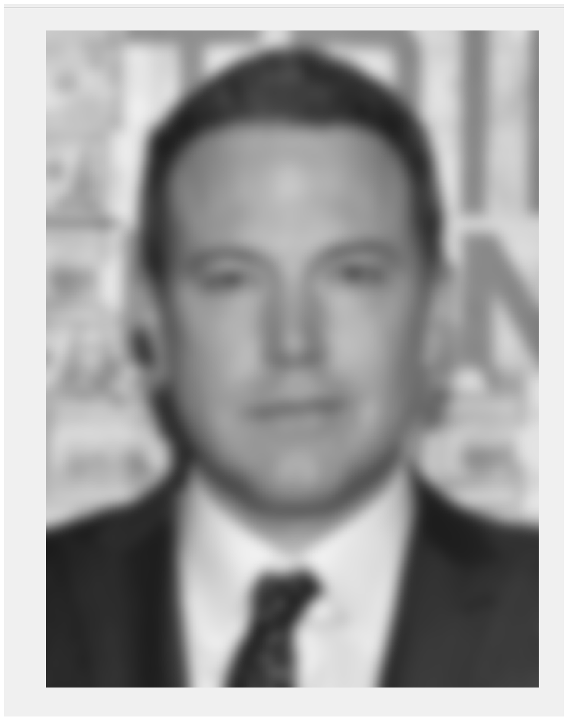
Utkarsh Pratap Singh Jadon

1) **Perform Gaussian smoothing on the grayscale image affleck_gray.png (or affleck_gray_flip.png). Try with multiple sigma values, starting with larger values (e.g., 20 to .5). When does the face become recognizable to your friends?**

Code:

```
sigma=11; % use different values
G = fspecial('gaussian', 2*ceil(3*sigma)+1, sigma);
faceIm = double(imread('affleck_gray.png'));
gIm = imfilter(faceIm, G, 'replicate');
imshow(gIm/255); % double images need range of 0-1
imwrite(uint8(gIm), 'gIm.bmp');
```

Output:



Inference:

Image becomes clearer as we vary sigma from 20 to 0.5. Most of my friends were able to recognize Ben Affleck from the above picture which was at sigma = 11.

2) **Write a function to compute and display the 2D Gaussian derivative masks Gx and Gy for a given sigma. Sample the Gaussian derivative/gradient (2D) equation directly (see class notes) at the appropriate x,y (col, row!) locations. Note: each mask is a square 2D matrix. Please ensure that the positive derivative lobe is on the side of the increasing direction of each axis for each mask (see notes regarding the negative sign in the equations). Plot each mask (either in 2D or 3D).**

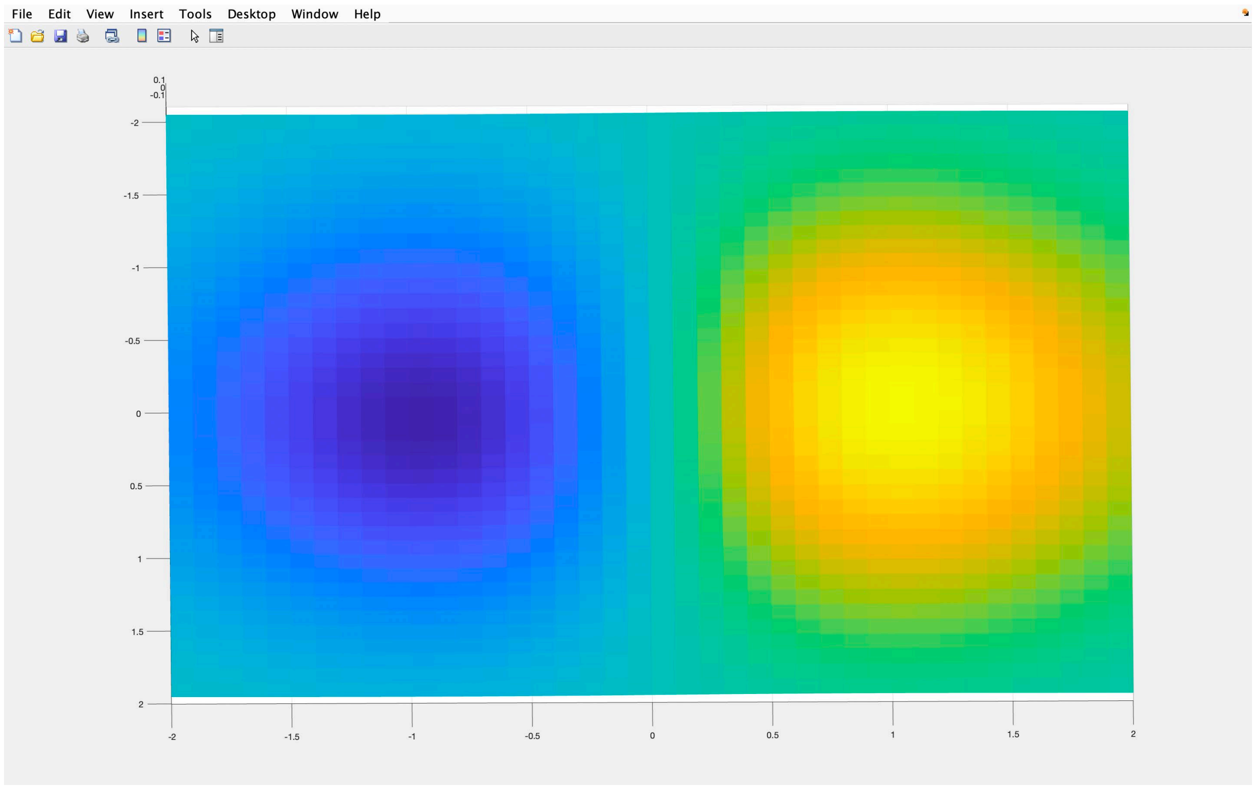
Code:

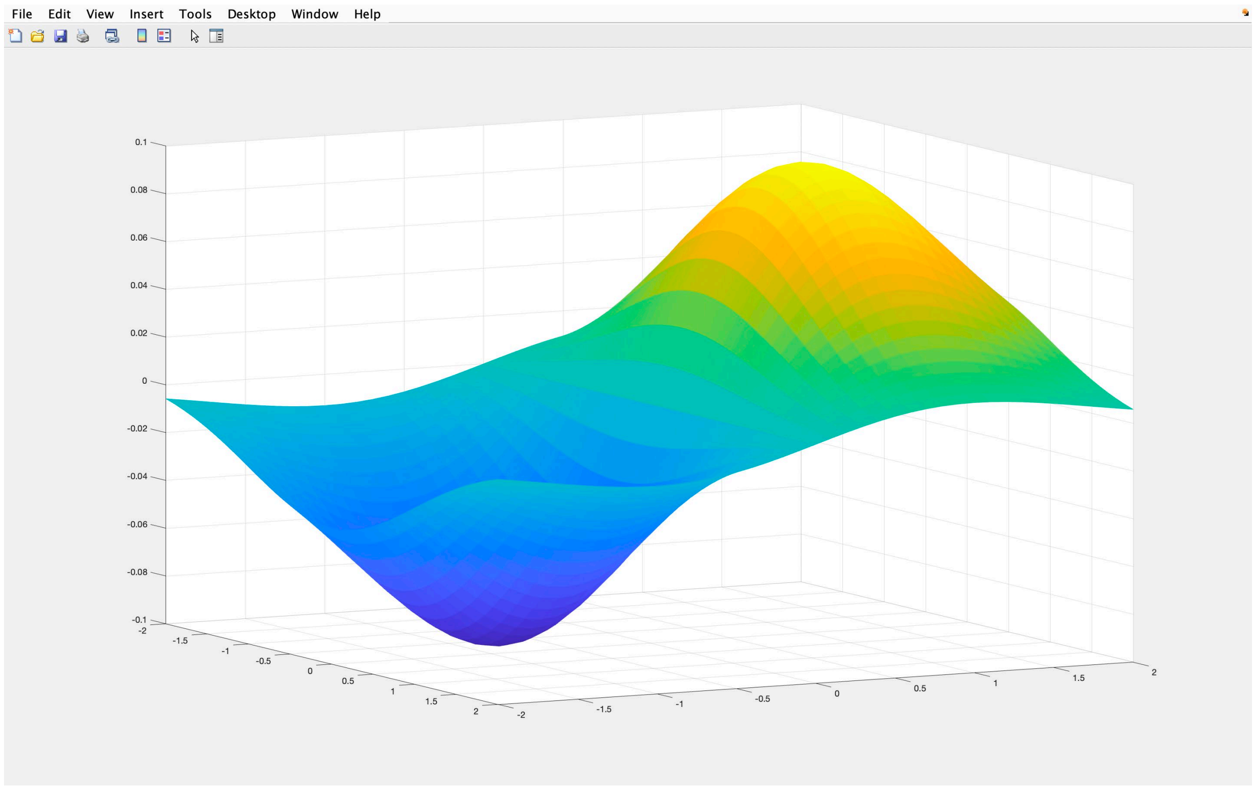
```
[Gx Gy] = gaussDeriv2D(1);
function [Gx, Gy] = gaussDeriv2D(sigma);
x=-ceil(2*sigma):1:ceil(2*sigma);
y=-ceil(2*sigma):1:ceil(2*sigma);


% I used step-size of 0.1 to get the figures. Otherwise I used step-size of
% 1 to get Gx and Gy matrices

for i=1:1:length(x)
for j=1:1:length(y)
   Gx(i,j)=((x(i))/(2*pi*sigma^4))*(exp(-(x(i)^2+y(j)^2)/(2*sigma^2)));
   Gy(i,j)=((y(j))/(2*pi*sigma^4))*(exp(-(x(i)^2+y(j)^2)/(2*sigma^2)));

end
end
Gx     %To get Gx matrix
Gy     %To get Gy matrix

figure(1);
s=mesh(x,y,Gx)          %To print Gx mesh figure (Step size = 0.1)
s.FaceColor = 'flat';

figure(2);
s=mesh(x,y,Gy)          %To print Gy mesh figure (Step size = 0.1)
s.FaceColor = 'flat';
end
```

Output:

Gx:

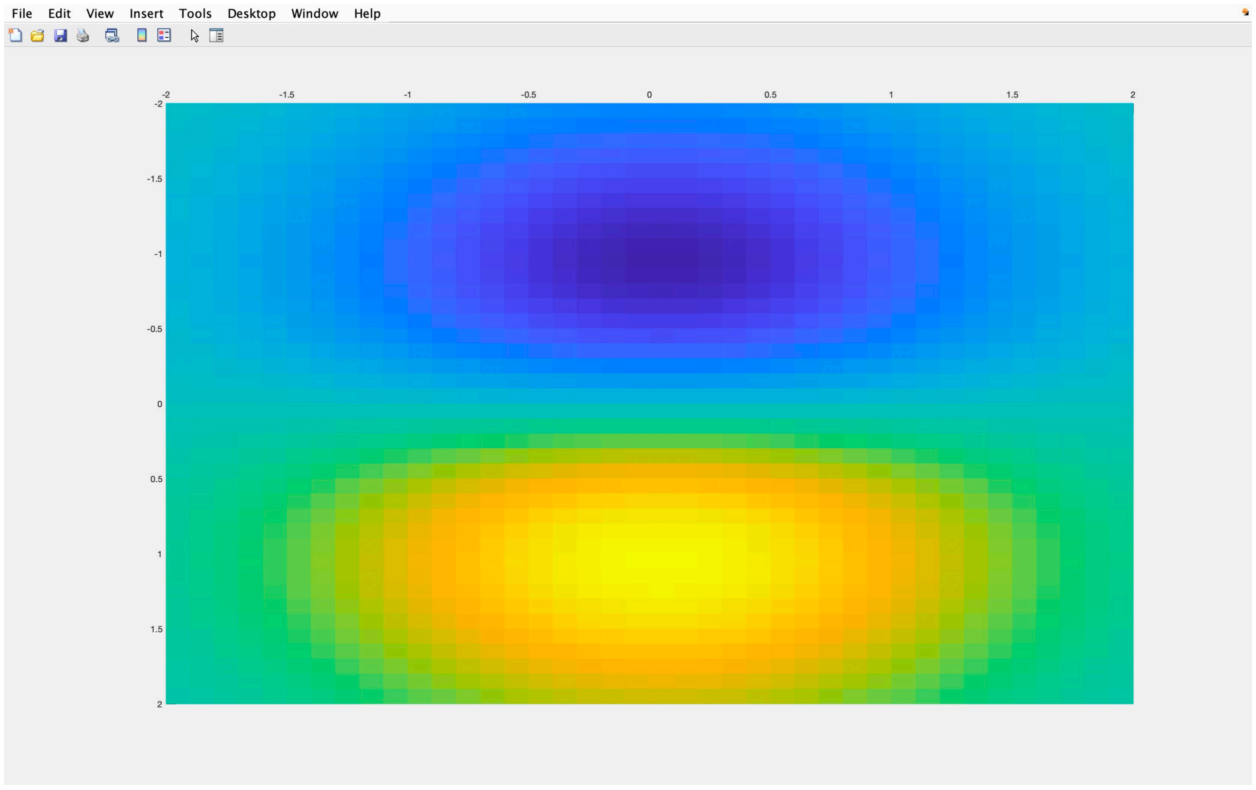
```
  -0.0058   -0.0261   -0.0431   -0.0261   -0.0058
  -0.0131   -0.0585   -0.0965   -0.0585   -0.0131
      0         0           0         0             0
   0.0131    0.0585    0.0965    0.0585    0.0131
   0.0058    0.0261    0.0431    0.0261    0.0058
```

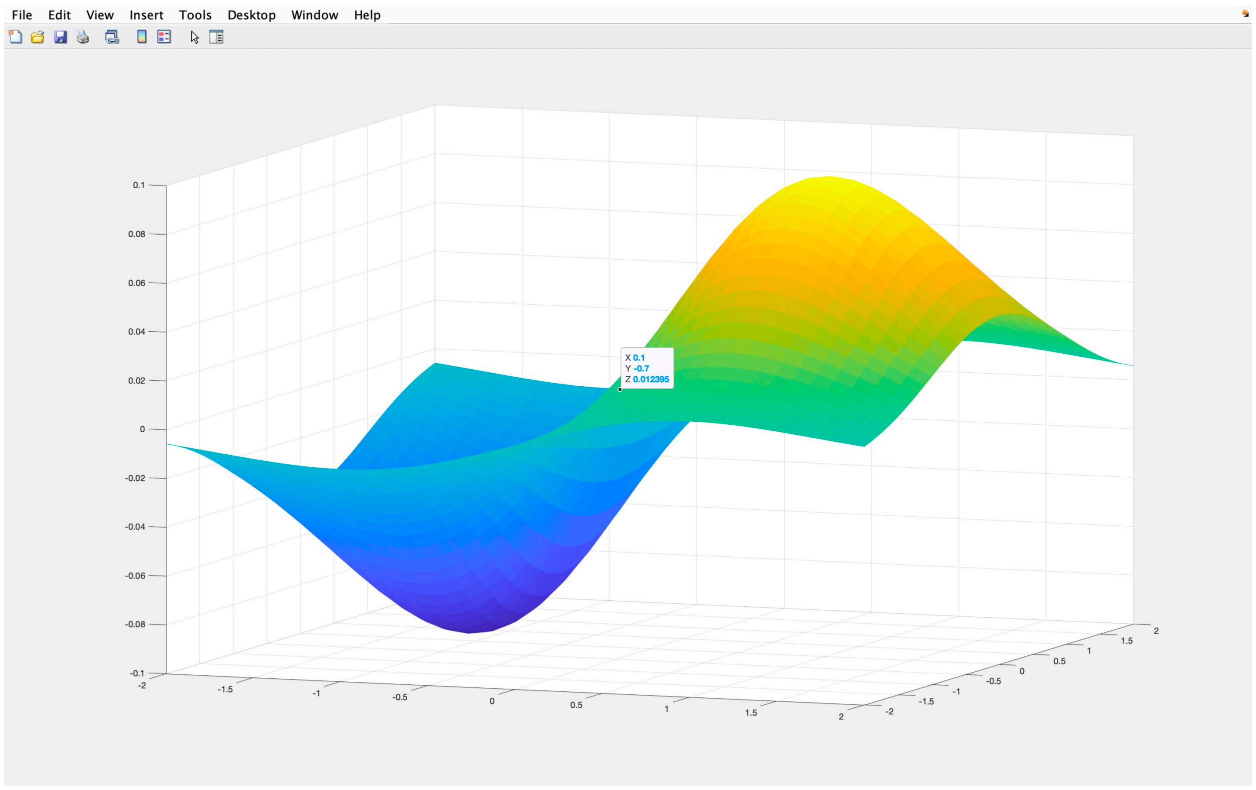

File
Edit
View
Insert
Tools
Desktop
Window
Help
0.1
0
-0.1
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


File
Edit
View
Insert
Tools
Desktop
Window
Help
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Gy:

```
   -0.0058   -0.0131         0    0.0131    0.0058
   -0.0261   -0.0585         0    0.0585    0.0261
   -0.0431   -0.0965         0    0.0965    0.0431
   -0.0261   -0.0585         0    0.0585    0.0261
   -0.0058   -0.0131         0    0.0131    0.0058
```

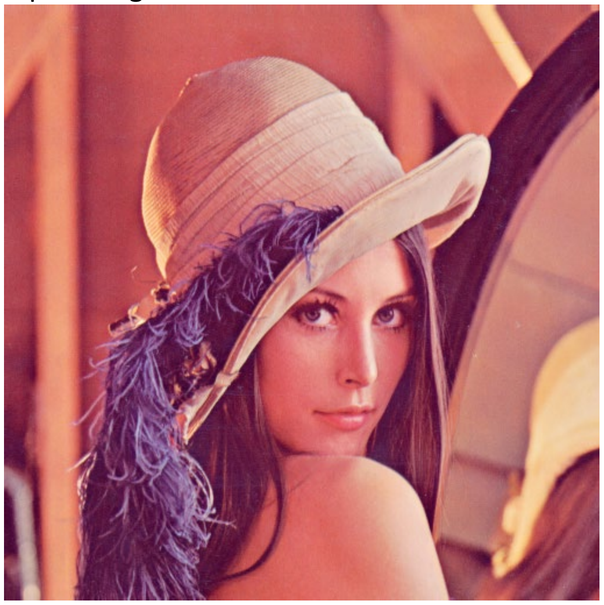




Inference: Even order derivatives are symmetric around zero as they are even functions

3) Compute and display the gradient magnitude of an image - search the web for an interesting image that has strong vertical and horizontal boundaries/edges; convert to grayscale if necessary (you know how to do this!); make sure to upload the image with your code in the Carmen submission.

Input image:

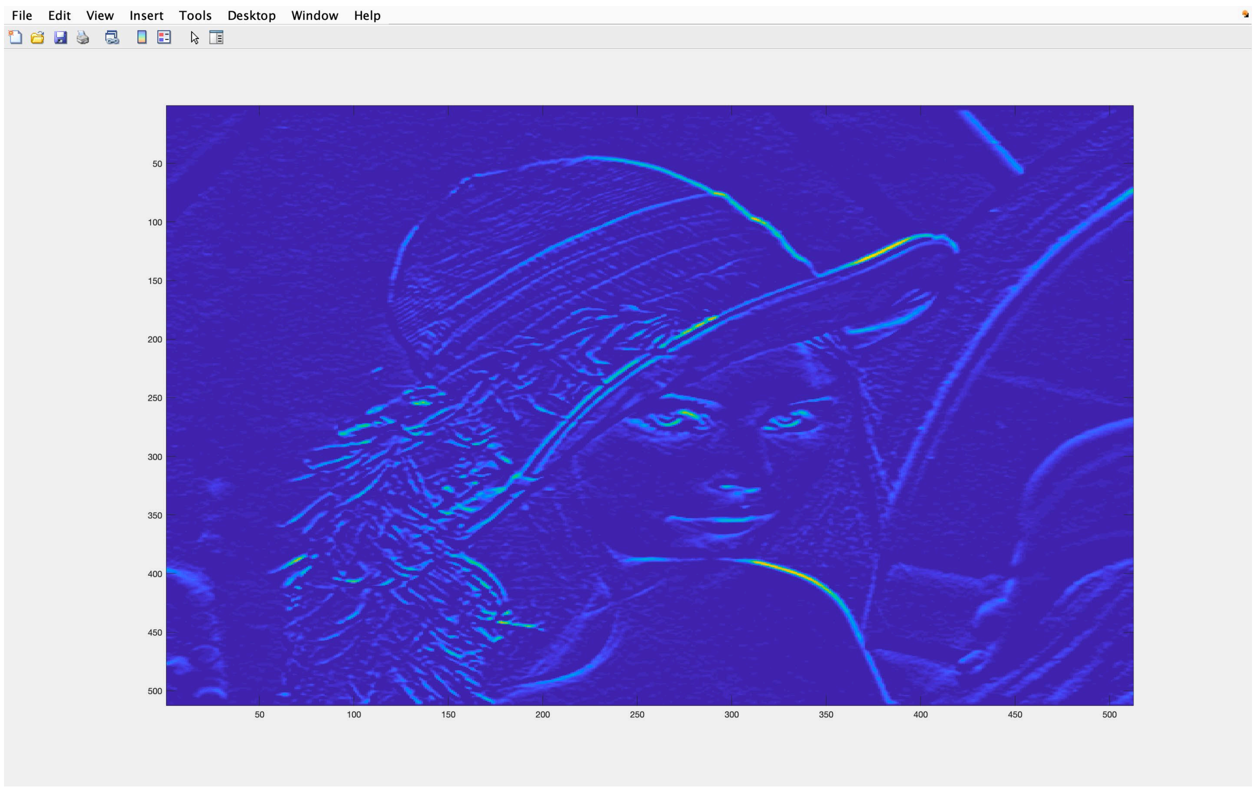


Code:

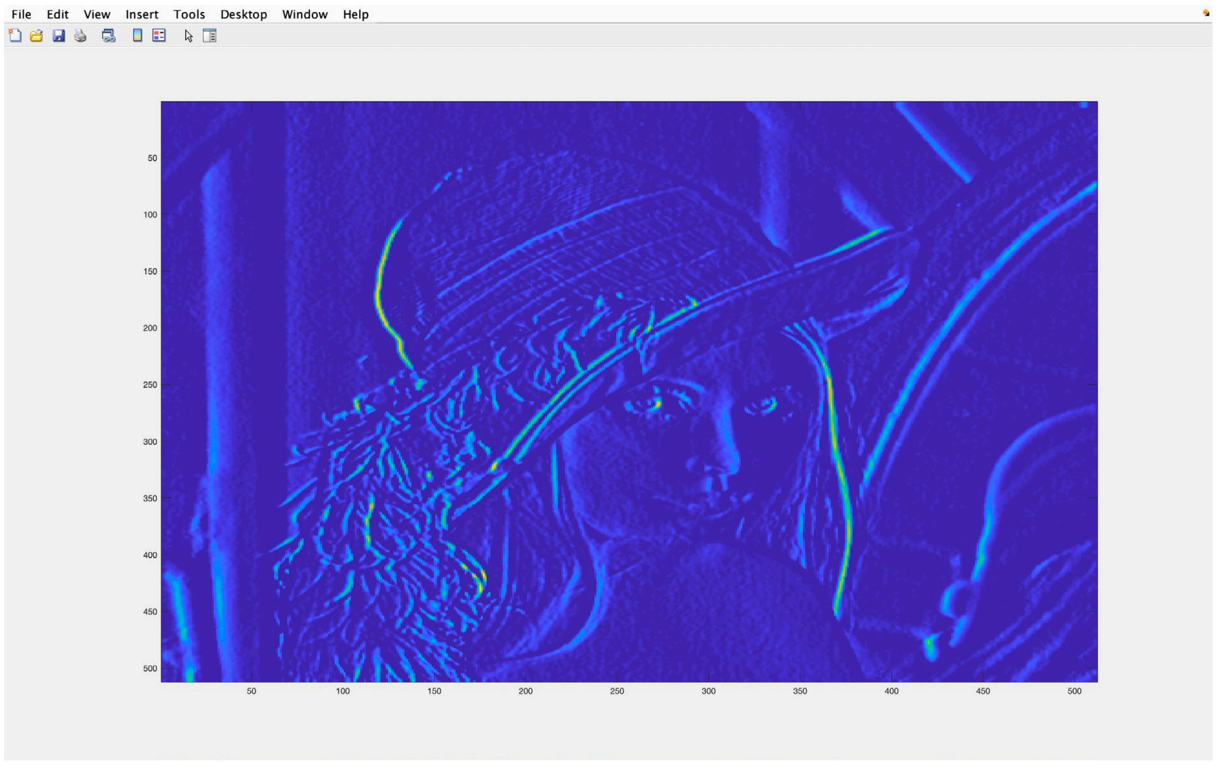
```
rgbIm = imread('Lenna.png');
myIm = rgb2gray(rgbIm);
gxIm = double(imfilter(myIm, Gx, 'replicate'));
gyIm = double(imfilter(myIm, Gy, 'replicate'));
magIm = sqrt(gxIm.^2 + gyIm.^2);
imagesc(gxIm);
imagesc(gyIm);
imagesc(magIm);
```

Output:

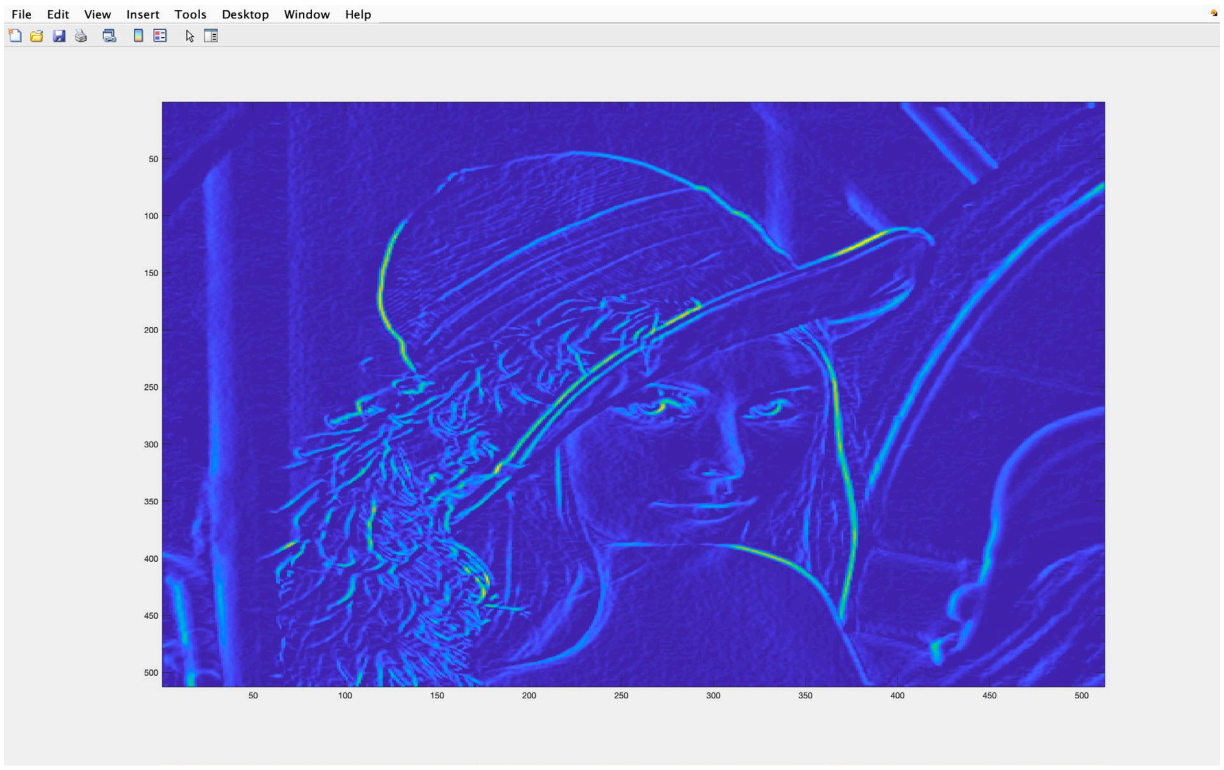
gxIm:



gyIm:


File
Edit
View
Insert
Tools
Desktop
Window
Help
50
100
150
200
250
300
350
400
450
500

magIm:


File
Edit
View
Insert
Tools
Desktop
Window
Help
50
100
150
200
250
300
350
400
450
500

Inference: gxIm can capture horizontal gradients efficiently, gyIm can capture vertical gradients efficiently, and hence magIm captures both gradients efficiently

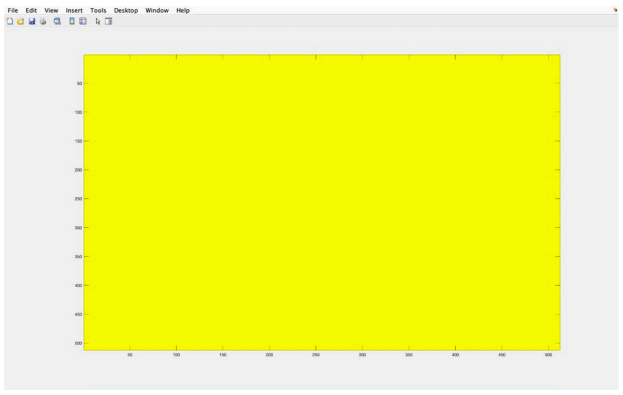
4) Threshold and display the "gradient magnitude" image with different threshold T levels.

Code:

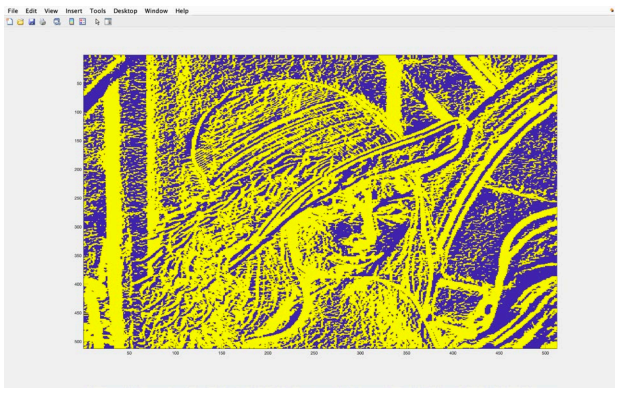
```
tIm = magIm > 5;
imagesc(tIm);
```

Output:

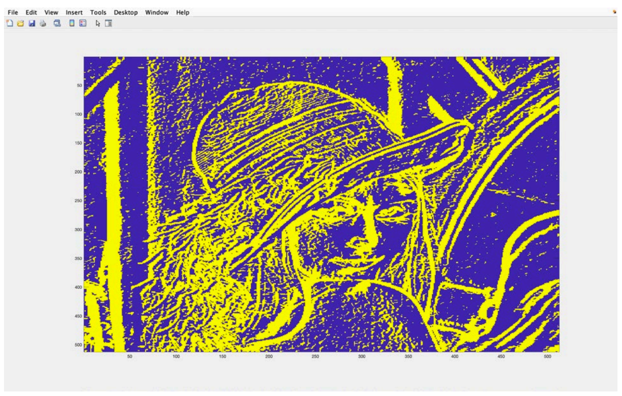
T=-1



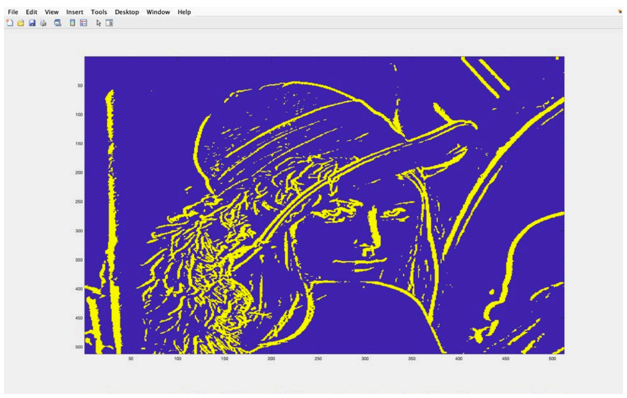
T=0



T=1

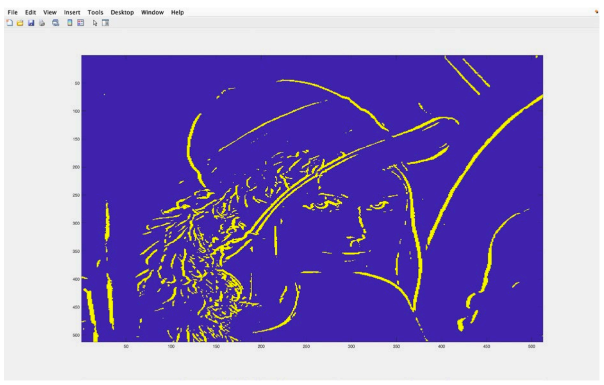


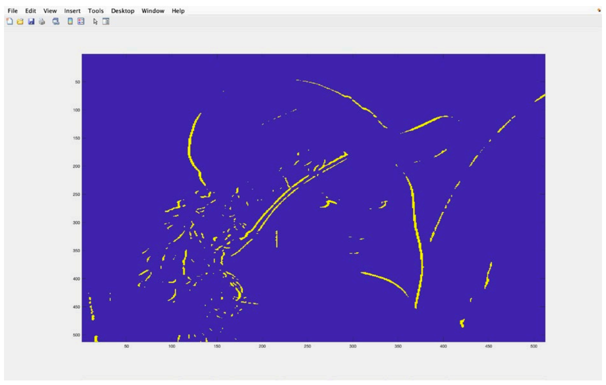
T=5



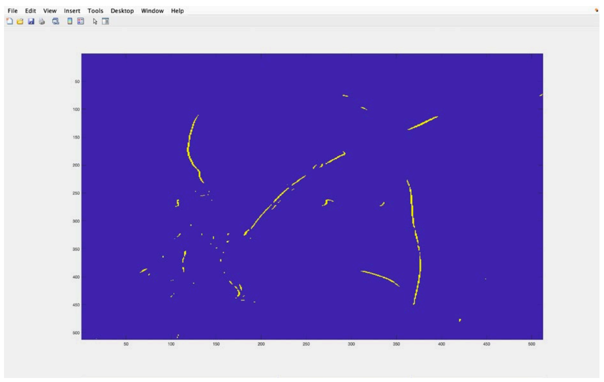
T=10

T=20





T=30



Inference: Gradient becomes more recognizable as we move T from -1 to 5. T = 1 to 5 gives best result. As we move to T=30, we see that gradients are lost

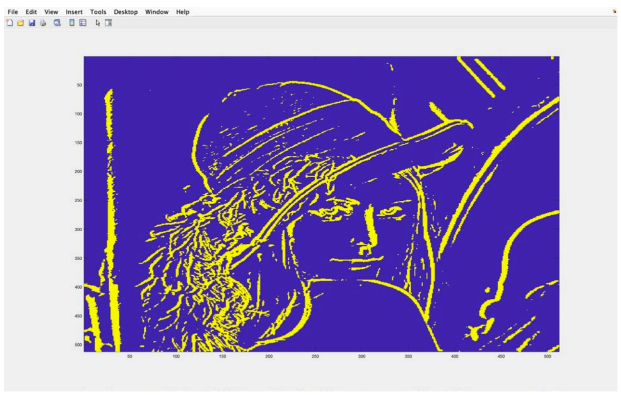
5) Compare the above results with the Sobel masks

Code:

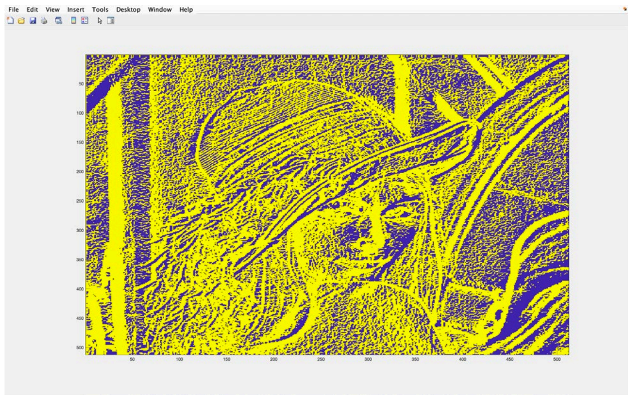
```
Fx = -fspecial('sobel')';
fxIm = double(imfilter(myIm,Fx));
Fy = -fspecial('sobel');
fyIm = double(imfilter(myIm,Fy));
magIm = sqrt(fxIm.^2 + fyIm.^2);
tIm = magIm > 20;
imagesc(tIm);
```

Output:

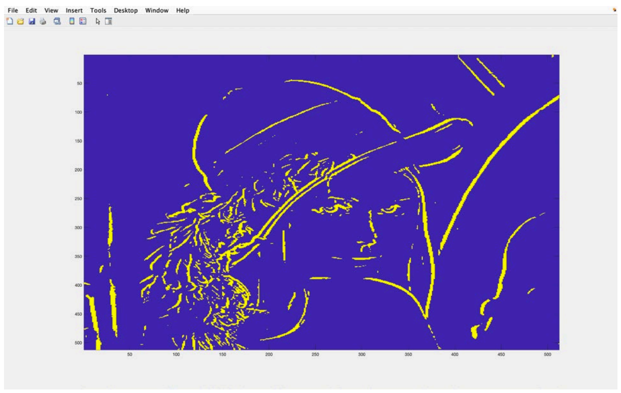
Above result (T=5)



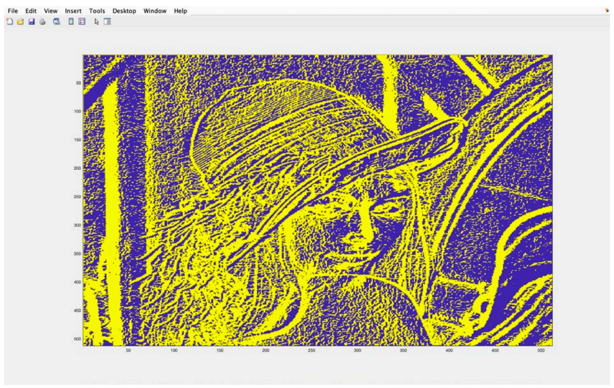
Sobel (T=5)



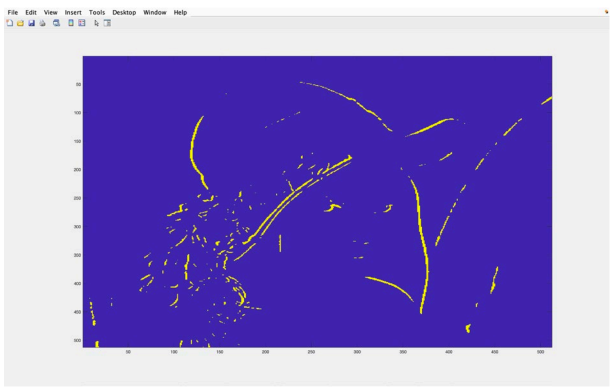
Above result (T=10)



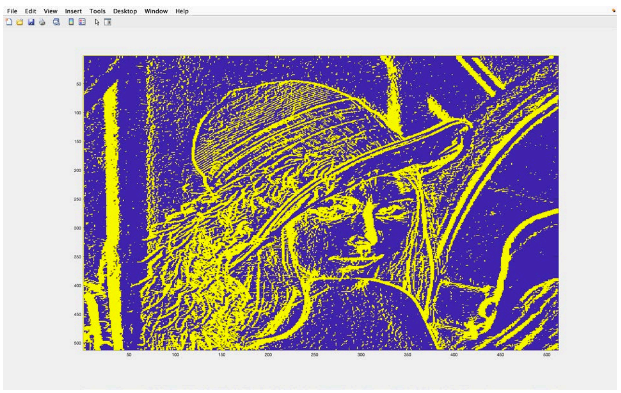
Sobel (T=10)



Above result (T=20)



Sobel (T=20)



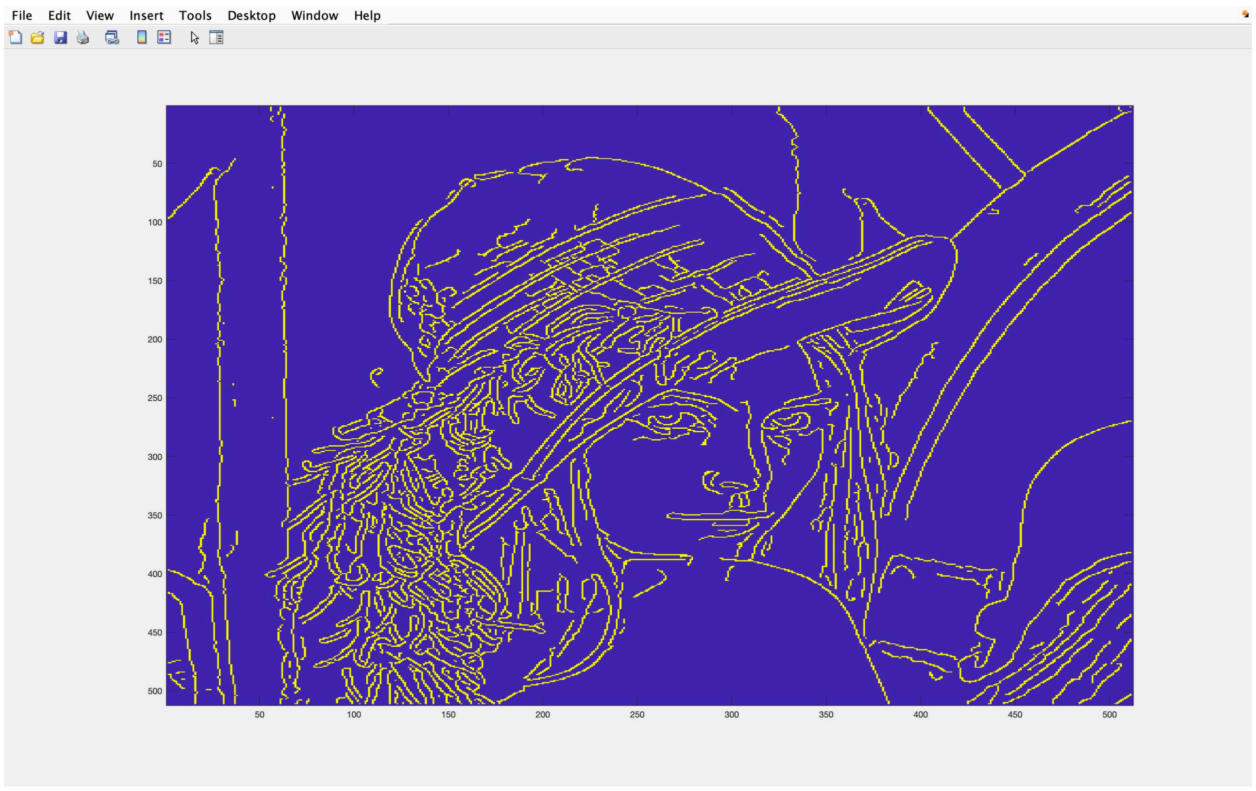
Inference: We see that given image becomes more recognizable as we increase T from 5 to 20 for Sobel, but less recognizable for previous method

6) **Run the MATLAB canny edge detector, edge(Im,'canny'), on your image and display the default results. How does it compare? (Python: you can use the scikit-image package)**

Code:

```
canIm = edge(myIm,'canny');
imagesc(canIm);
```

Output:



Inference: We see that canny edge detector detects most of the edges in foreground and less in background, whereas previous methods detected edges equally in foreground as well as background