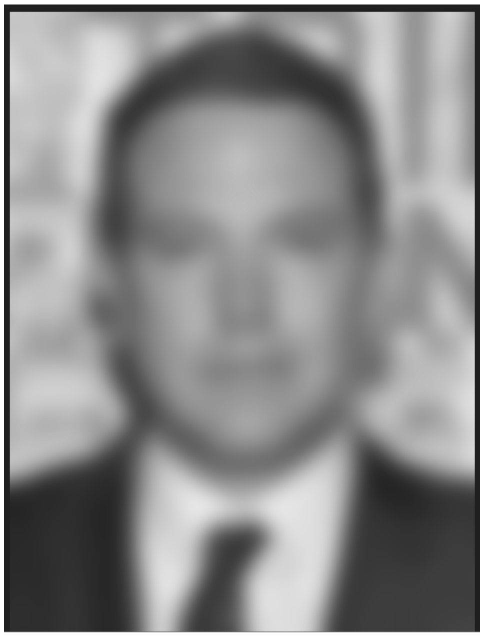
CSE 5524
8/26/22
John Wu

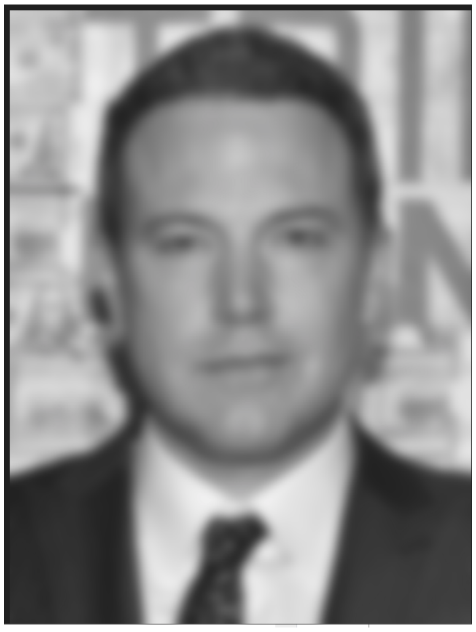
1. **Perform Gaussian smoothing on the grayscale image affleck_gray.png (or affleck_gray_flip.png). Try with multiple sigma values, starting with larger values (e.g., 20 to .5). When does the face become recognizable to your friends? [2 pts]**

   Shown below, the tested sigma values are 20, 10, 5, 1, and 0.5.

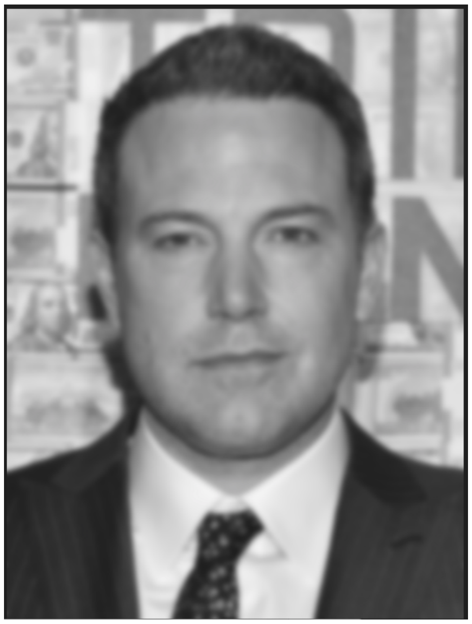
   Sigma Value 20:



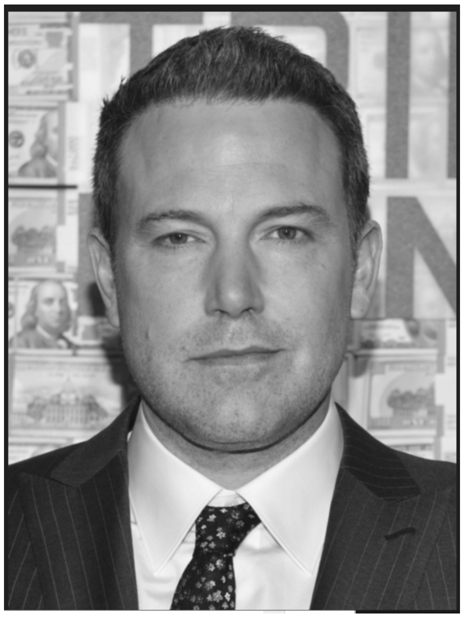
   Sigma Value 10:



   Sigma Value 5:



   Sigma Value 1:

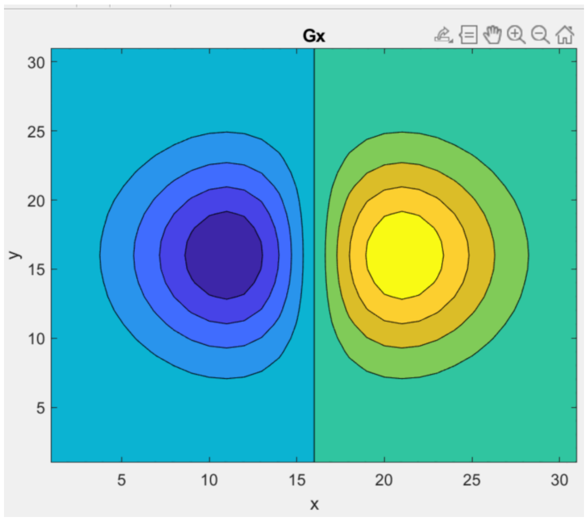


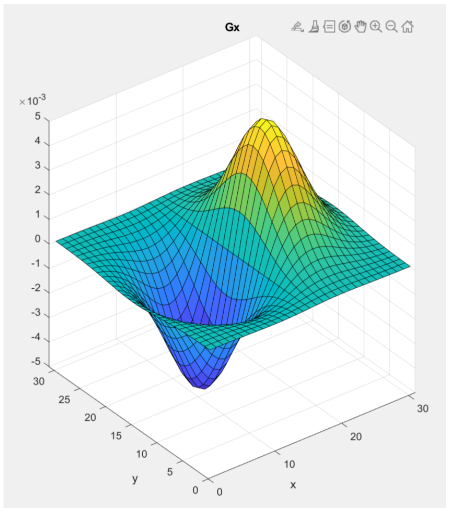
   Sigma Value 0.5:

My friend was only able to truly determine his least favorite batman at sigma value 5. For sigma values at 10 and above, the picture is too “smoothed” or “blurred” to make out the facial features of Ben Affleck.

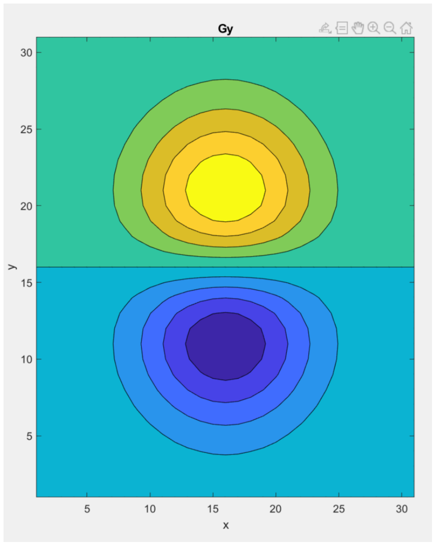
2. **Write a function to compute and display the 2D Gaussian derivative masks Gx and Gy for a given sigma. Sample the Gaussian derivative/gradient (2D) equation directly (see class notes) at the appropriate x,y (col, row!) locations. Note: each mask is a square 2D matrix. Please ensure that the positive 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). [3 pts]**

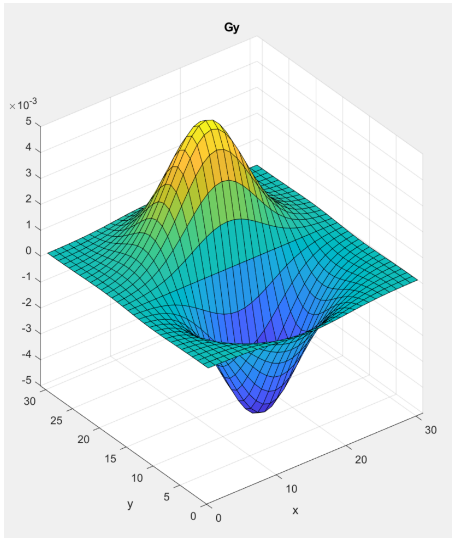
As a sanity check, both 2D and 3D plots were done.
Given sigma = 5, we have the following plots. (Next Page).
Gx





Gy

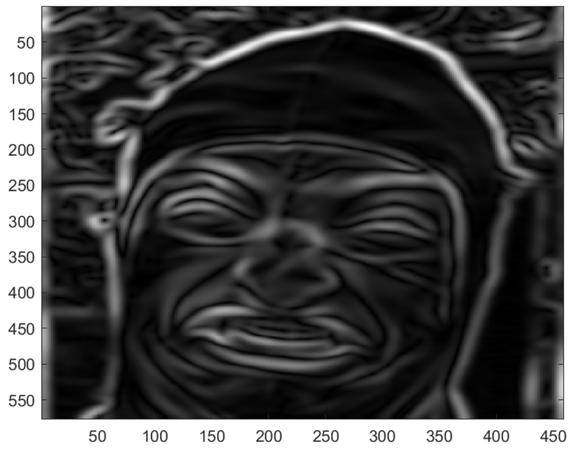




Both positive derivative lobes are on the “increasing” side of their respective axes. Gx’s positive (lighter color) is increasing on the x-axis and similarly for Gy. The code will plot and save both. Mission success!

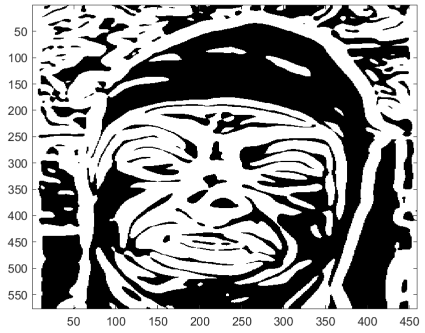
3. **Compute and display the gradient magnitude of an image - search the web for an interesting image that has strong vertical and horizontal boundaries/edges; convert to grayscale if necessary (you know how to do this!); make sure to upload the image with your code in the Carmen submission. [2 pts]**

   Note that we had to first convert the picture to grayscale first as the original was in RGB, which would not work well with the filter scheme that was made with the Gaussian derivative. It’s interesting to note that the person’s face has a lot of edges all over the place in both the X and Y direction (i.e diagonal edges, etc. etc.).



4. **Threshold and display the “gradient magnitude” image with different threshold T levels. [2 pts]**

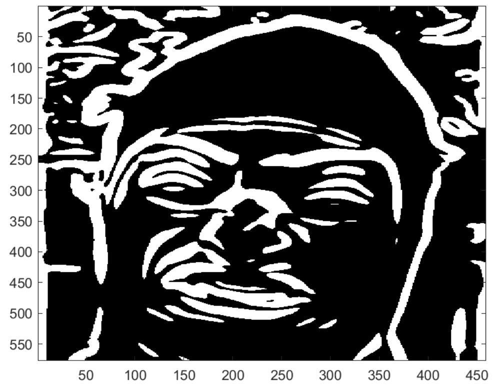
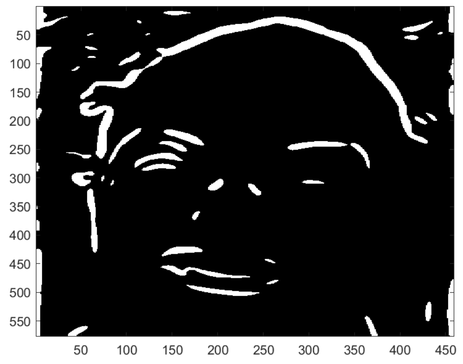
In this case, we chose different thresholds based on the maximum magnitude gradient value, maxThresh, in the “magIm” image. We chose 5 different thresholds,maxThresh/10, maxThresh / 5, ⅖ * maxThresh, ⅗ * maxThresh, and ⅘ * maxThresh. These thresholds correspond to approximately, 9.4, 18.7, 37.4, 56.2, and 74.9 respectively.



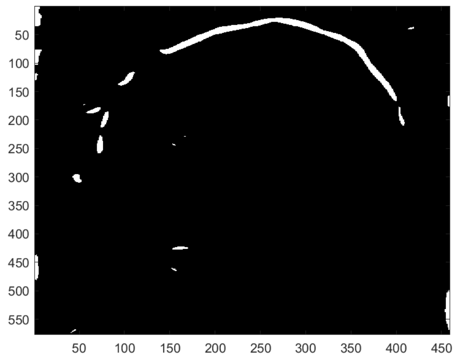
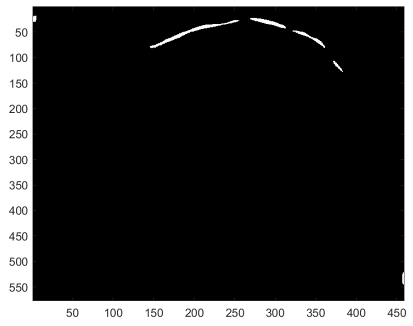
T = 9.4

T = 18.7

T = 37.4

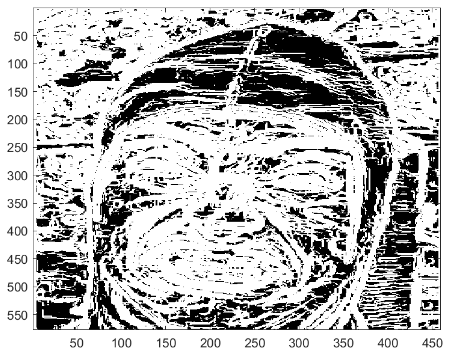
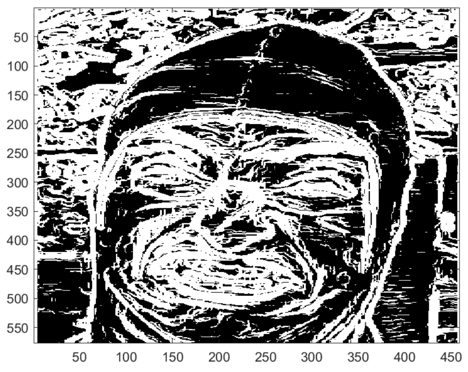
T = 56.2 T = 74.9

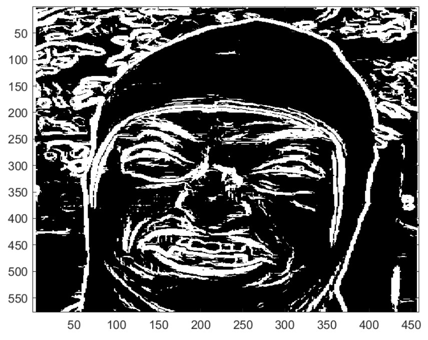
A key takeaway from these thresholds is that (1) not all edges have the same magnitude of “gradients” and (2) that there is an optimal point or interval of thresholds as you can include either too much noise (unimportant features may be a better term) or take out too much and lose important information. T = 18.7 is a pretty good medium for retaining enough information.

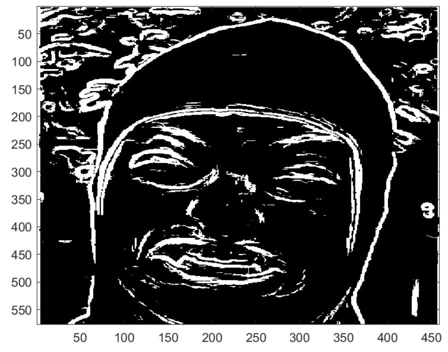
5. **Compare the above results with the Sobel masks. [2 pts]**

T = 9.4 T = 18.7

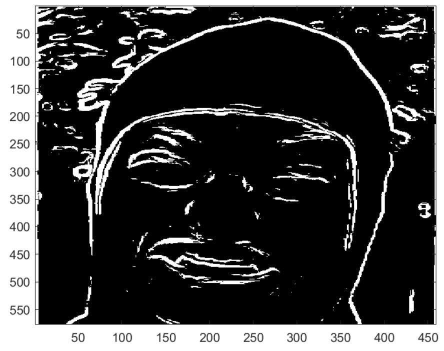



T = 37.4 T = 56.2



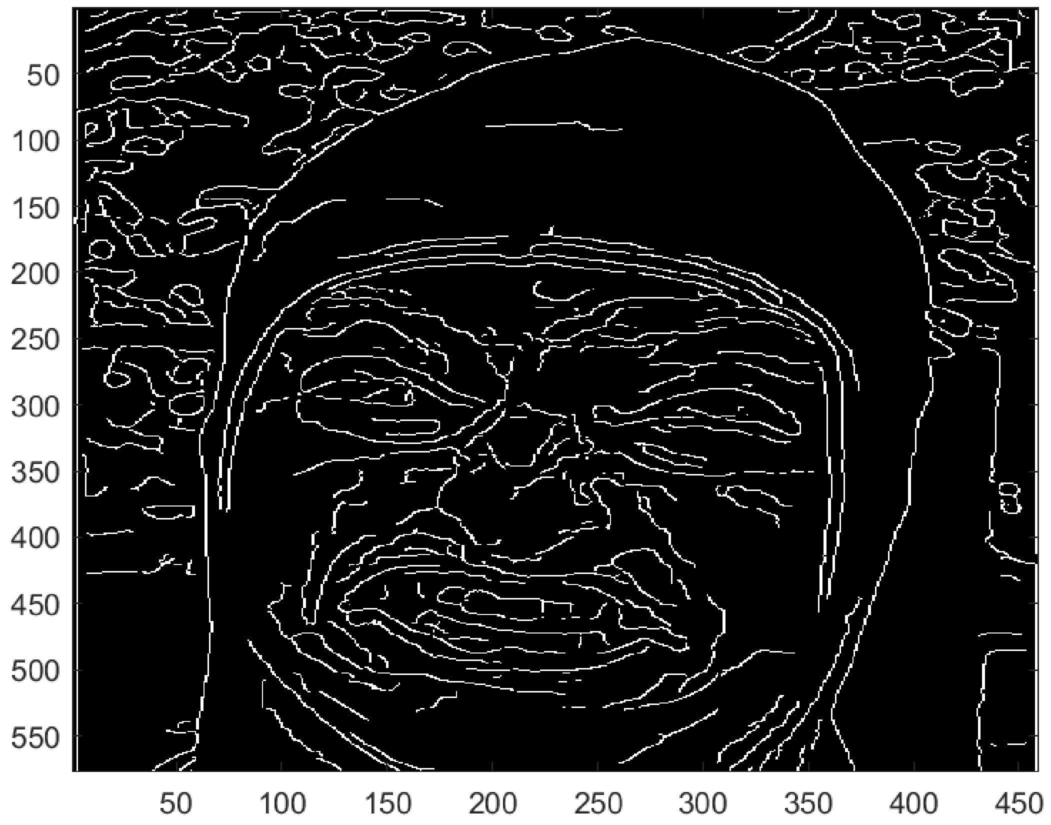


T = 74.9



This is a fascinating result where it seems like the optimal interval of thresholding is much higher than the gaussian filter (i.e. T = 37.4) Observe that the first two images have a lot more noise than the first two seen in the gaussian filter. Furthermore, all edges look much more "noisy" in general.

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) [2 pts]**



One, it's great that you don't have to define thresholds anymore. Two, it seems that canny is capable of capturing the Teeth in much more detail. However, it seems to have caught some edges that may not be as easy to see from the human eye, especially on his cheeks, which might be wrinkles. It is able to capture quite an amazing amount of information. However, when I ask my friends who this is, only by showing the original picture can they discern that it is the pink guy.

```
% John Wu
% CSE 5524
% 9/1/2022
% 1. perform gaussian smoothing on the grayscale image affleck_gray.png (or
% affleck_gray_flip.png). Try it for 5 values, sigma = 20, sigma = 10,
% sigma = 5, sigma = 1, sigma = 0.5
sigmas = [20, 10, 5, 1, 0.5];
%sigma=1.0; % use different values
for sigma = sigmas
  G = fspecial('gaussian', 2*ceil(3*sigma)+1, sigma); % create gaussian filter of 3sigma
  faceIm = double(imread('affleck_gray.png')); % read in image with double precision
  gIm = imfilter(faceIm, G, 'replicate'); % now let's filter it
  imshow(gIm/255); % double images need range of 0-1
  imwrite(uint8(gIm), 'gIm_Sigma'+ string(sigma) + '.bmp'); % convert back to integer
```

```
   pause;
end
%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). [3 pts]
% function definitions below!!
sigma = 5;
[Gx, Gy] = gaussDeriv2D(sigma);
figure
contourf(Gx); % 2D / 3D representations, note all the axis labels to make things clear.
title('Gx');
xlabel('x');
ylabel('y');
saveas(gcf,'gX.png');
pause;
surf(Gx);
saveas(gcf,'gX3D.png');
title('Gx')
xlabel('x')
ylabel('y')
pause;
contourf(Gy)
saveas(gcf,'gY.png');
title('Gy')
xlabel('x')
ylabel('y')
pause;
surf(Gy)
saveas(gcf,'gY3D.png');
title('Gy')
xlabel('x')
ylabel('y')
pause;
% 3. Compute and display the gradient magnitude of an image on the web
myIm = imread('PinkGuy.png');
myIm = double(rgb2gray(myIm)); % convert to gray scale
imagesc(myIm);
colormap("gray");
pause;
gxIm = imfilter(myIm, Gx, 'replicate');
gyIm = imfilter(myIm, Gy, 'replicate');
magIm = sqrt(gxIm.^2 + gyIm.^2); % magnitude
imagesc(gxIm);
pause;
imagesc(gyIm);
pause;
```

```
% display magnitude image and save image
imagesc(magIm);
colormap(gray)
saveas(gcf, 'magnitudeImagePinkGuy.png')
pause;
% 4.) Threshold and display the "gradient magnitude” image with different threshold T
% levels. [2 pts] Please note that the
maxThresh = max(max(magIm));
% pick 4 maxThreshs
Thresholds = [maxThresh / 10, maxThresh / 5, 2*maxThresh/5, 3*maxThresh/5, 4*maxThresh/5];
for T = Thresholds
  tIm = magIm > T;
  imagesc(tIm);
  saveas(gcf, 'pGuyThreshGauss' + string(T) + '.png');
  pause;
end
% 5) Compare the above results with the Sobel masks. [2 pts]
Im = myIm;
for T = Thresholds
  Fx = -fspecial('sobel')';
  fxIm = imfilter(Im,Fx);
  Fy = -fspecial('sobel');
  fyIm = imfilter(Im,Fy);
  magIm = sqrt(fxIm.^2 + fyIm.^2);
  tIm = magIm > T;
  imagesc(tIm);
  saveas(gcf, 'pGuySobelThresh' + string(T) + '.png');
  pause;
end
% 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) [2 pts]
cannyIm = edge(Im, 'canny');
imagesc(cannyIm);
saveas(gcf, 'pGuyCannyEdge.png');
%----------------- FUNCTION DEFINITIONS ----------------- %
%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). [3 pts]
function dGc = dColumnGauss(r, c, centerColumn, centerRow, sigma)
  mainTerm = (c - centerColumn) / (2 * pi * sigma^4);
  exponentialTerm = exp(-(((c - centerColumn)^2 + (r - centerRow)^2))/(2*sigma^2));
  dGc = (mainTerm * exponentialTerm);
end
function dGr = dRowGauss(r, c, centerColumn, centerRow, sigma)
  mainTerm = (r - centerRow) / (2 * pi * sigma^4);
  exponentialTerm = exp(-(((c - centerColumn)^2 + (r - centerRow)^2))/(2*sigma^2));
  dGr = mainTerm * exponentialTerm;
end
function [Gx, Gy] = gaussDeriv2D(sigma)
  % use 3 sigma mask size equation
  maskDim = ceil(3*sigma);
  maskSize = 2*maskDim + 1;
  centerRow =  maskDim + 1;
  centerColumn = maskDim + 1;
  Gx = zeros(-maskDim, maskDim); % x == column-wise
  Gy = zeros(-maskDim, maskDim); % y == row-wise
  for r=1:maskSize
     for c=1:maskSize
        Gx(r,c) = dColumnGauss(r, c, centerColumn, centerRow, sigma);
        Gy(r,c) = dRowGauss(r, c, centerColumn, centerRow, sigma);
     end
  end
  Sx = sum(abs(Gx), 'all'); % Probably same value, because square matrix
  Sy = sum(abs(Gy), 'all');
  Gx = Gx / Sx; % make sure mask adds up to 1.
  Gy = Gy / Sy;
end
```