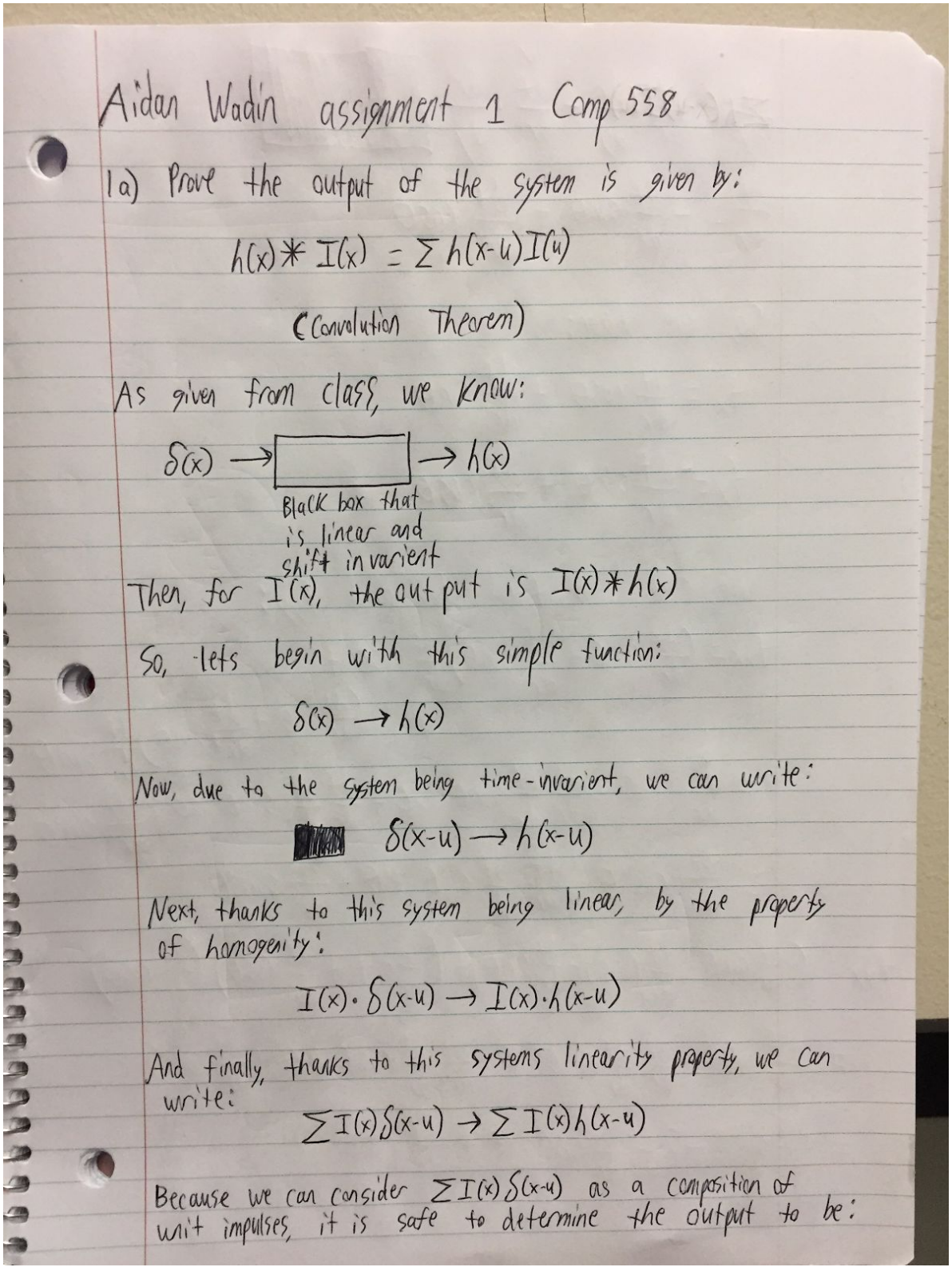
Aidan Wadin - COMP 558 - Assignment 1

1.A.



Aidan Wadin assignment 1 Comp 558

1a) Prove the output of the system is given by:

$$h(x) * I(x) = \sum h(x-u) I(u)$$

(Convolution Theorem)

As given from class, we know:

$\delta(x) \rightarrow$ [ ] $\rightarrow h(x)$

Black box that is linear and shift invarient

Then, for $I(x)$, the output is $I(x) * h(x)$

So, lets begin with this simple function:

$$\delta(x) \rightarrow h(x)$$

Now, due to the system being time-invarient, we can write:

$$\delta(x-u) \rightarrow h(x-u)$$

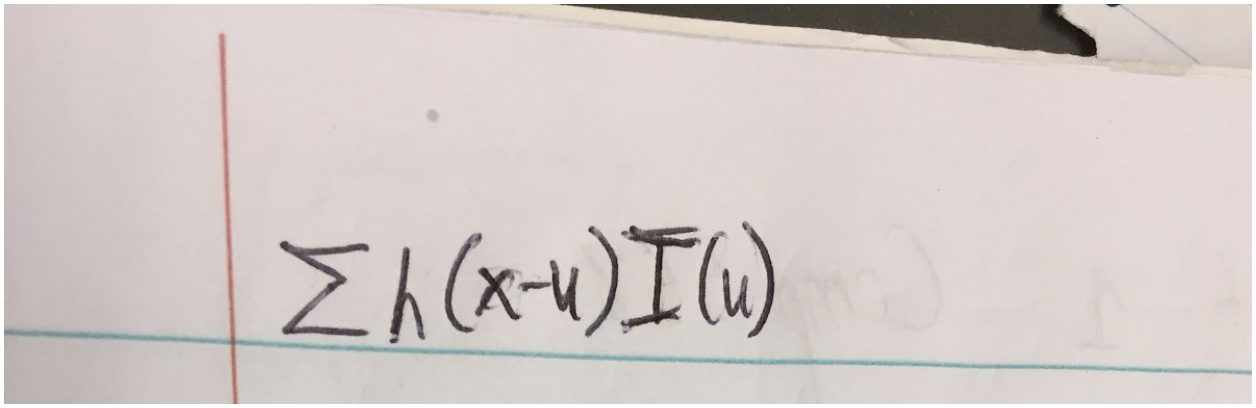
Next, thanks to this system being linear, by the property of homogenity:

$$I(x) \cdot \delta(x-u) \rightarrow I(x) \cdot h(x-u)$$

And finally, thanks to this systems linearity property, we can write:

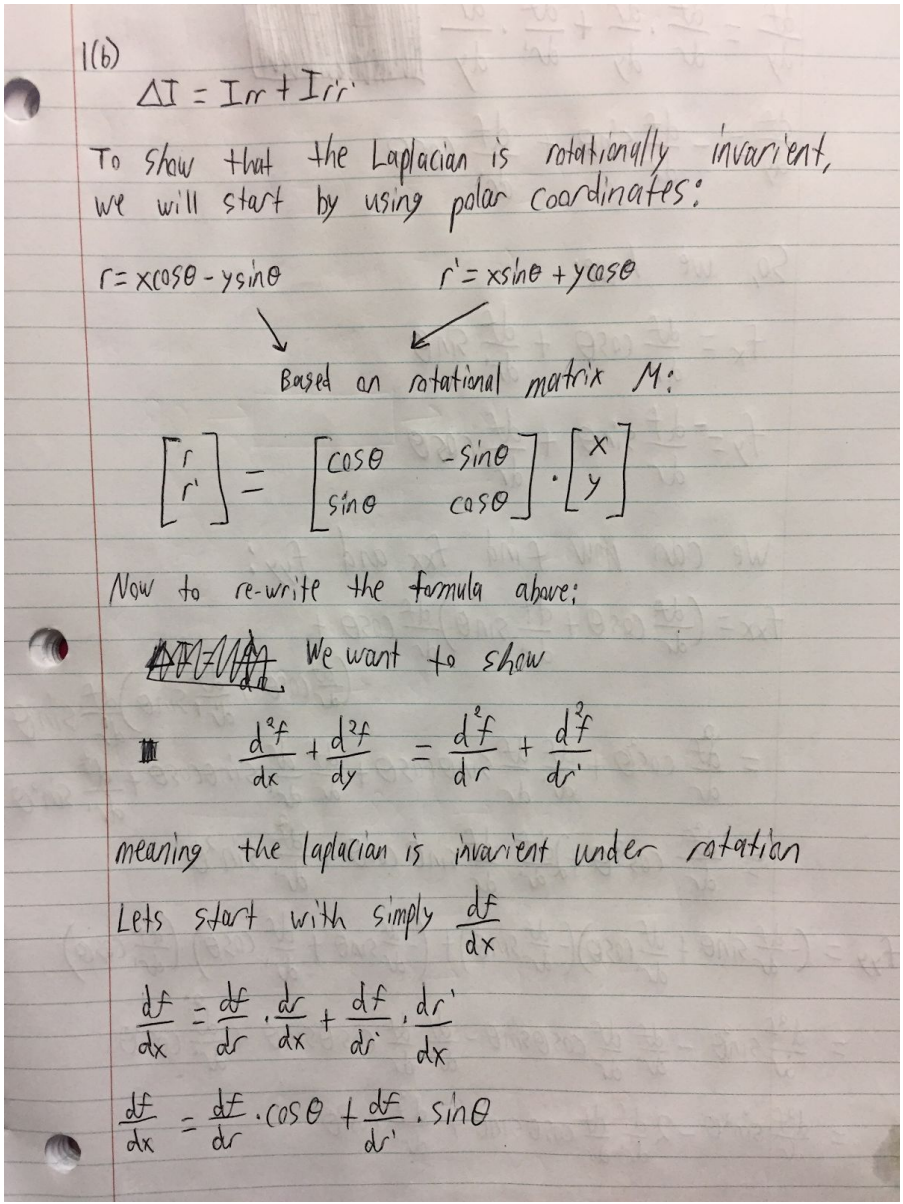
$$\sum I(x) \delta(x-u) \rightarrow \sum I(x) h(x-u)$$

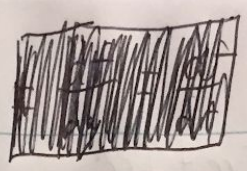
Because we can consider $\sum I(x) \delta(x-u)$ as a composition of unit impulses, it is safe to determine the output to be:



With I being our input pixel at each point. This shows that we can go from the basic impulse input -> impulse response all the way to the full convolution theorem by using the properties of linearity and time-invariance alone.

B.



$$\frac{df}{dy} = \frac{df}{dr}\cdot\frac{dr}{dy} + \frac{df}{dr'}\cdot\frac{dr'}{dy}$$ 

$$\frac{df}{dy} = -\frac{df}{dr}\sin\theta + \frac{df}{dr'}\cos\theta$$

So, we have:

$$f_x = \frac{df}{dr}\cos\theta + \frac{df}{dr'}\sin\theta$$

$$f_y = -\frac{df}{dr}\sin\theta + \frac{df}{dr'}\cos\theta$$

We can now find f_{xx} and f_{yy}:

$$f_{xx} = \left(\frac{df}{dr}\cos\theta + \frac{df}{dr'}\sin\theta\right)\frac{df}{dr}\cos\theta + \left(\frac{df}{dr}\cos\theta + \frac{df}{dr'}\sin\theta\right)\frac{df}{dr'}\sin\theta$$

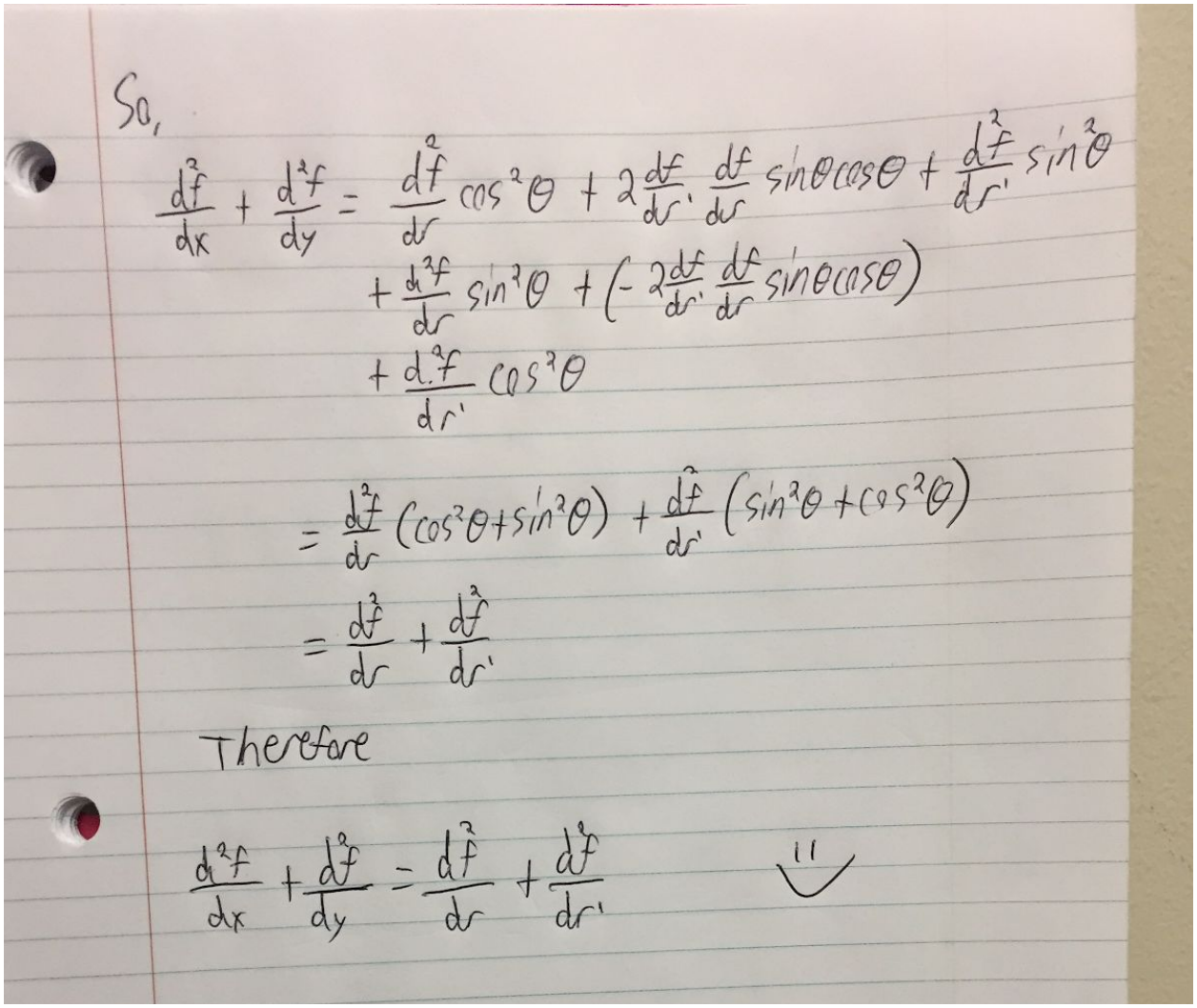
$$= \frac{d^2f}{dr}\cos^2\theta + \frac{df}{dr'}\frac{df}{dr}\sin\theta\cos\theta + \frac{df}{dr}\frac{df}{dr'}\sin\theta\cos\theta + \frac{d^2f}{dr'}\sin^2\theta$$

$$= \frac{d^2f}{dr}\cos^2\theta + 2\frac{df}{dr'}\frac{df}{dr}\sin\theta\cos\theta + \frac{d^2f}{dr'}\sin^2\theta$$

$$f_{yy} = \left(-\frac{df}{dr}\sin\theta + \frac{df}{dr'}\cos\theta\right)\left(-\frac{df}{dr}\sin\theta\right) + \left(-\frac{df}{dr}\sin\theta + \frac{df}{dr'}\cos\theta\right)\left(\frac{df}{dr'}\cos\theta\right)$$

$$= \frac{d^2f}{dr}\sin^2\theta - \frac{df}{dr'}\frac{df}{dr}\cos\theta\sin\theta - \frac{df}{dr'}\frac{df}{dr}\cos\theta\sin\theta + \frac{d^2f}{dr'}\cos^2\theta$$

$$= \frac{d^2f}{dr}\sin^2\theta - 2\frac{df}{dr'}\frac{df}{dr}\cos\theta\sin\theta + \frac{d^2f}{dr'}\cos^2\theta$$



As you can see, after running through the calculation of the double derivative, all terms with theta in them have been removed, meaning that those terms do not matter in the total equality, supporting the claim that the Laplacian operator is rotationally invariant.

C.

The condition that they argue must be satisfied is known as the condition of linear variation, and it states: “the intensity variation near and parallel to the line of zero-crossings should be locally linear”. What that means is that the intensity differences along (parallel to) the edge being detected needs to be linear in order for the Laplacian to pick up the direction perpendicular to the local direction of the edge.

We know that the following is the formula for the Laplacian:

Laplacian (operator)

$$\nabla^2 I(x,y) \equiv \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

Now this can be explained through the following example:

Example: vertical image edge

$I(x,y)$  $\bullet$ $\nabla^2 G(x,y,\sigma)$

Q: What happens when we convolve with $\nabla^2 G(x,y,\sigma)$?

$$\nabla^2 G(x,y,\sigma) * I(x,y) \equiv \frac{\partial^2 G(x,y,\sigma) * I(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y,\sigma) * I(x,y)}{\partial y^2}$$

$= 0$

$I(x,y)$ does not depend on y, and so $G(x,y,\sigma) * I(x,y)$ does not depend on y, and so derivative in y direction is 0.

33

Intensity only depends on x, the intensity on y is constant

As we can see here, the edge is in the vertical direction, therefore, the intensity variation in the y direction is linearly constant, so its Laplacian element will be 0. This then allows us to easily find the perpendicular direction to the edge because of the removal of the y direction element from the Laplacian.

With that in mind, we know the Laplacian is rotationally invariant, meaning as long as the 2 “x” and “y” variables are orthogonal, the operator will work properly. If we encounter and edge like:

$I(x,y)$ 

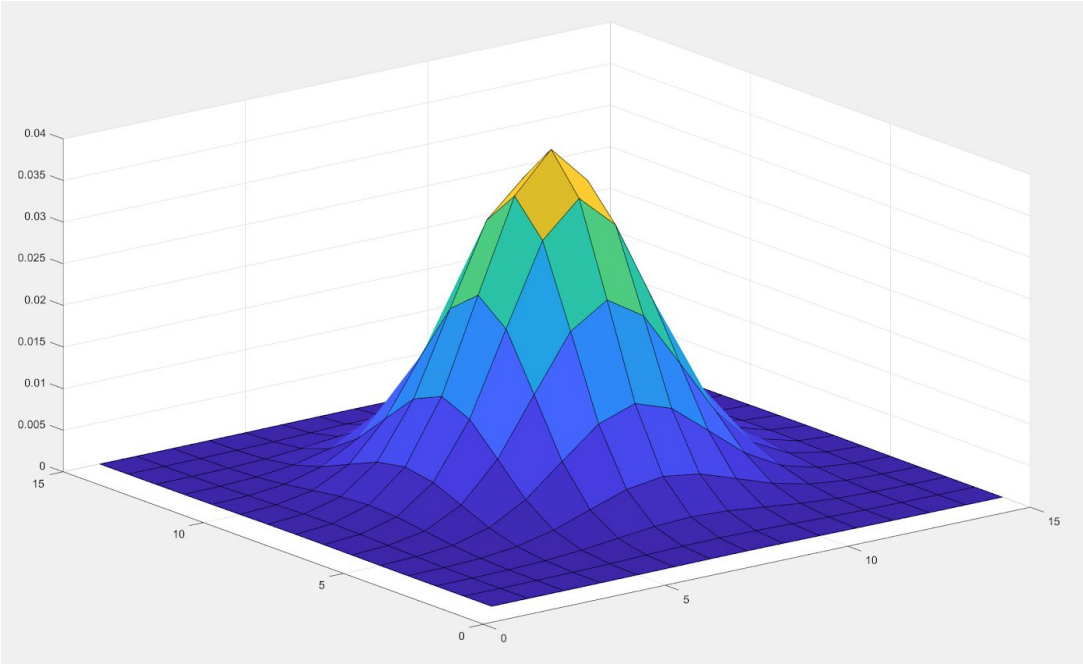
We can see that x and y are not perfectly orthogonal along this line, but we can then rotate the axis’ we are using in order to get to “x” and “y”s such that the two directions are orthogonal (one parallel to the edge and the other perpendicular). Under the assumption that the intensity

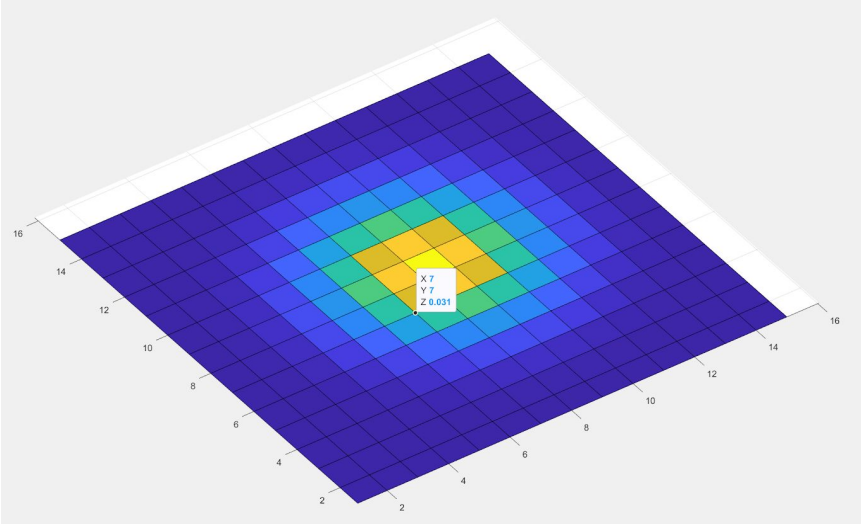
parallel to the edge is linear, the part of the Laplacian that is parallel to the edge will always be able to be taken out of the equation (end up = 0 when evaluated). This will leave only the term in the Laplacian that coincided to being perpendicular to the direction of the edge! If the edge and area around and parallel to it was in someway not linear, then the second term in the Laplacian equation would not be immediately be removed, and then the exact direction perpendicular to the edge would not be able to be immediately determined.

++

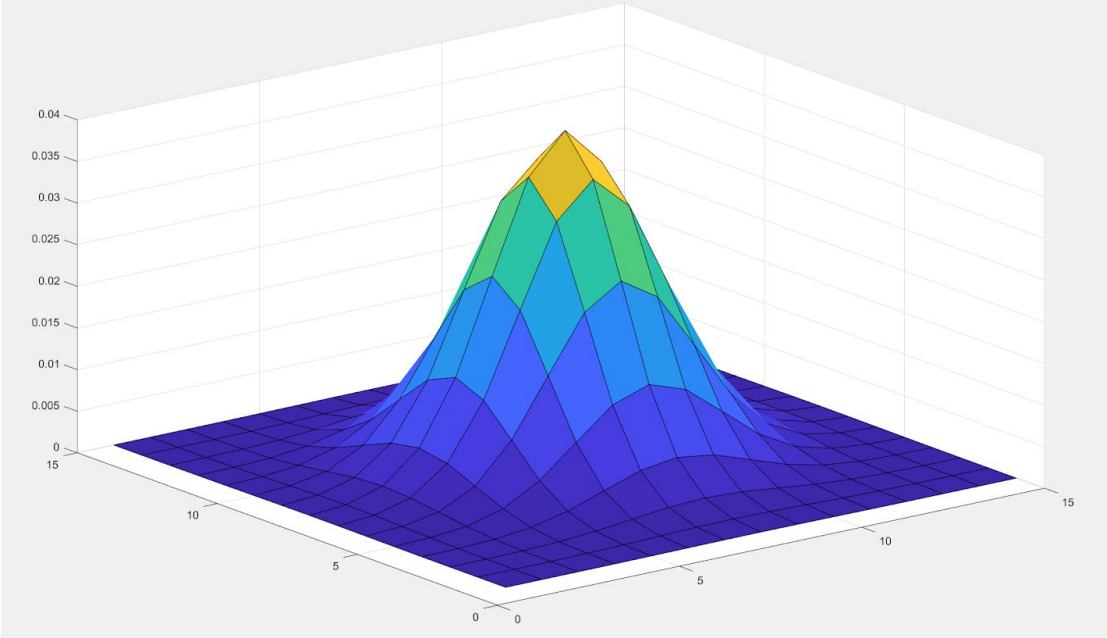
2. A.

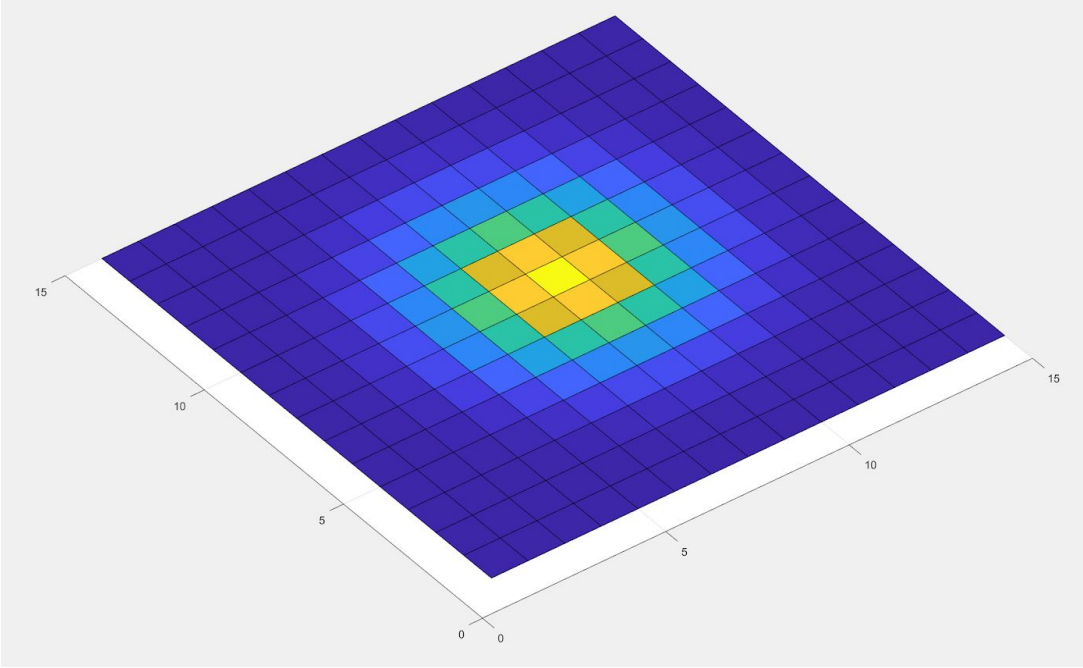
For reference, here is the Matlab function for creating a gaussian filter:
fspecial('Gaussian', 15, 2)



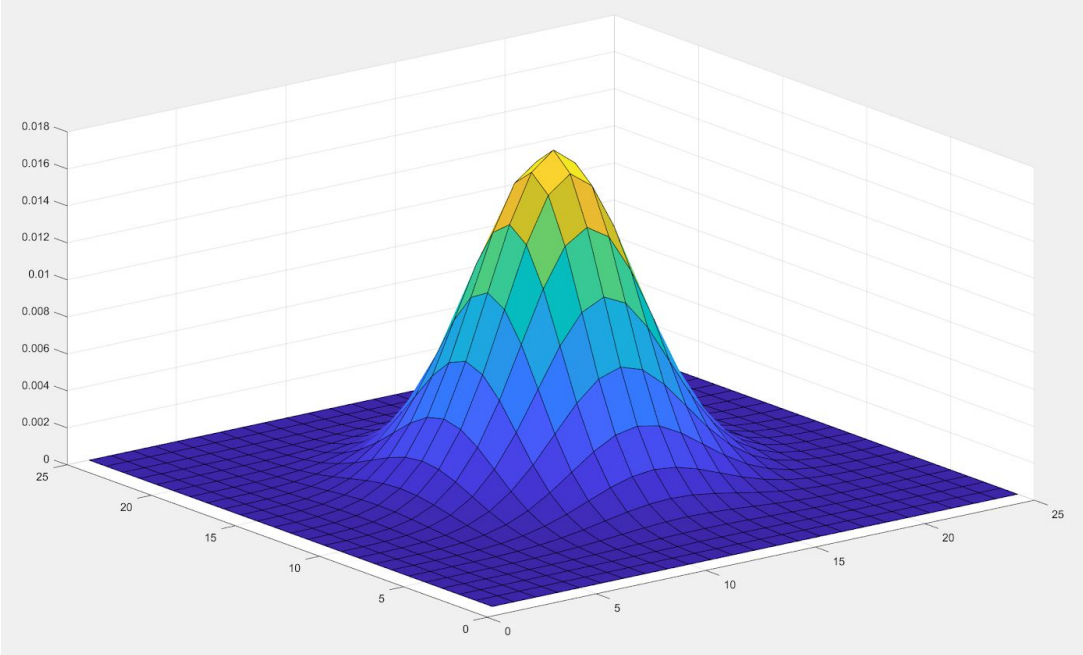


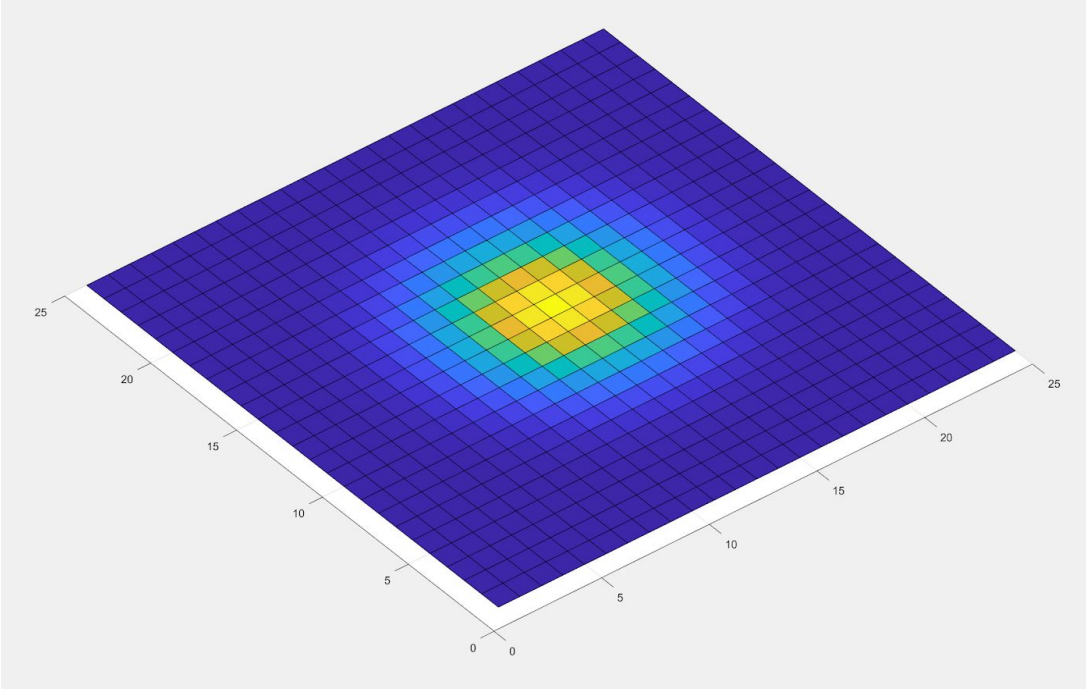
Here is my make2DGaussian(15, 2)



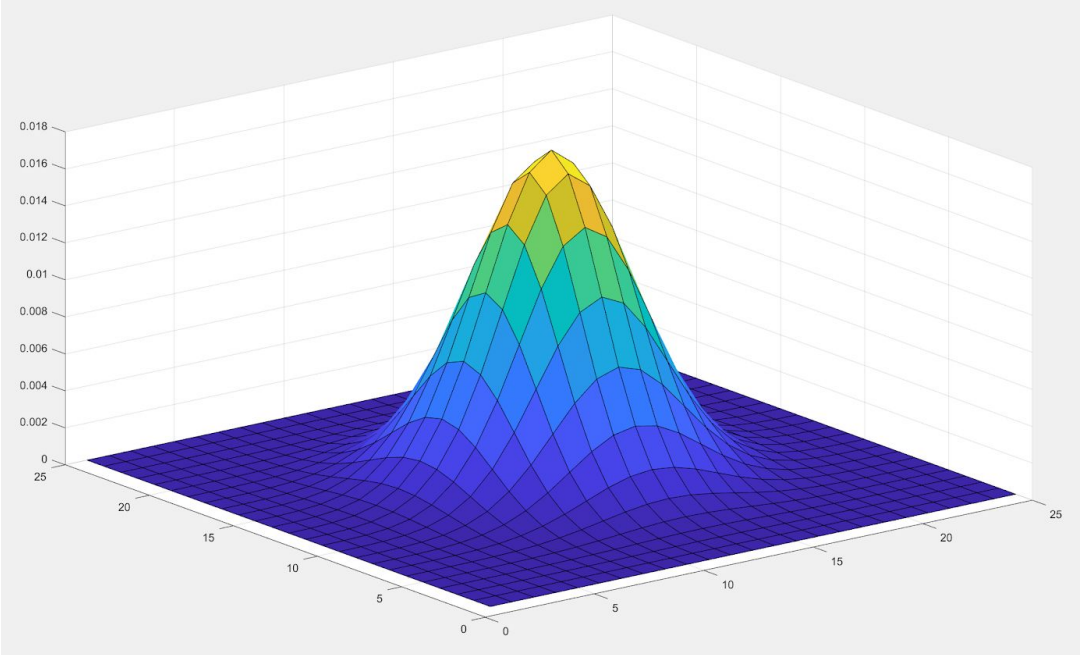


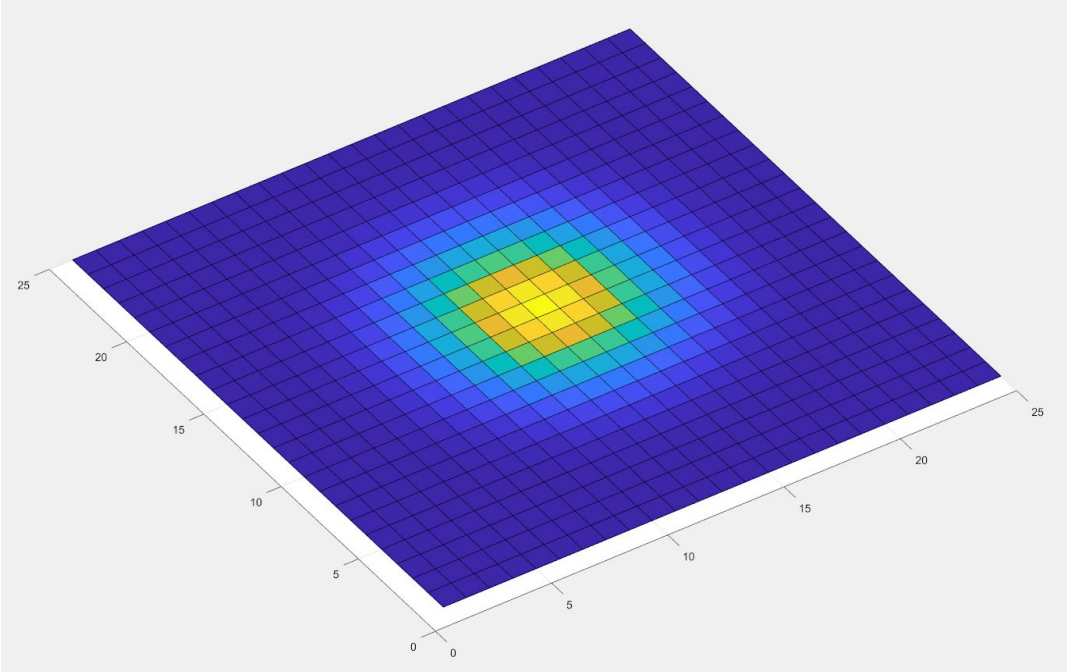
fspecial(‘Gaussian’, 25, 3):



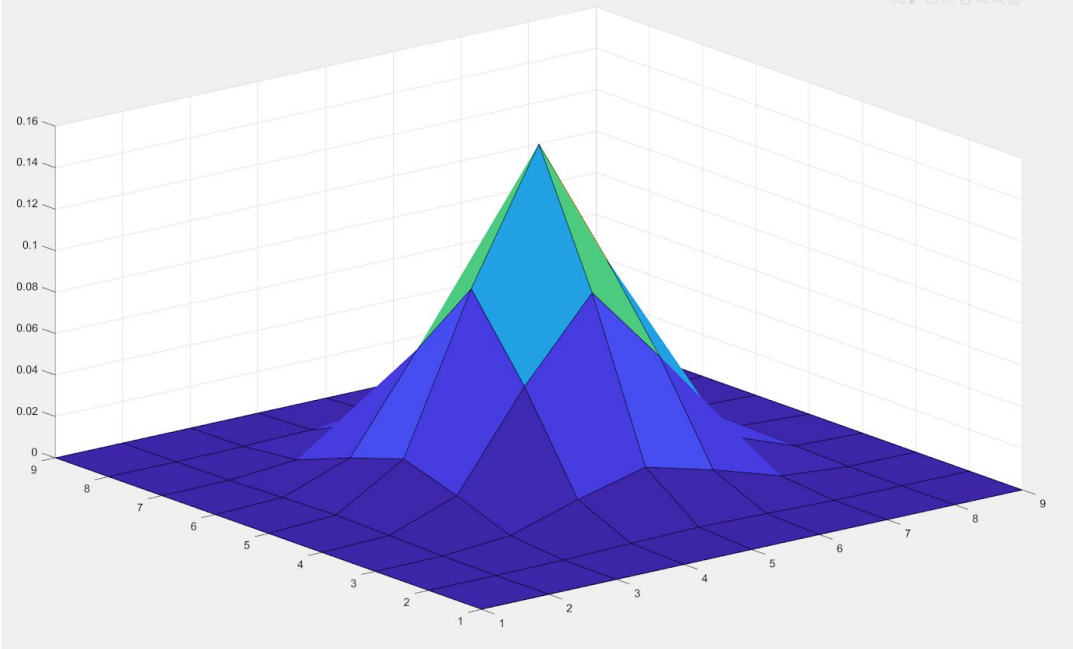


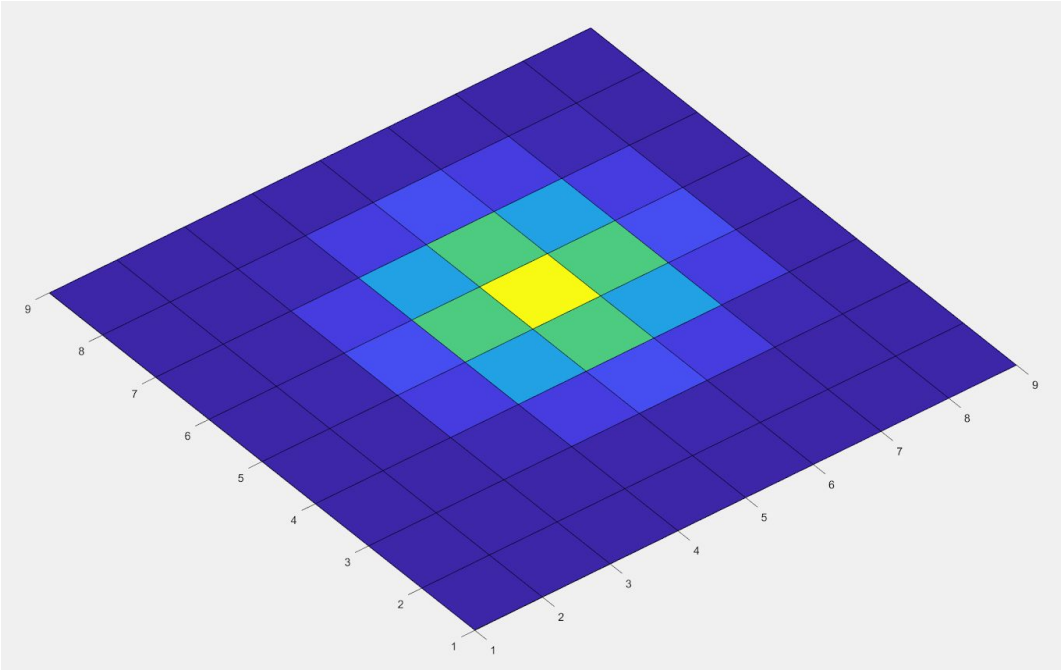
make2DGaussian(25, 3)





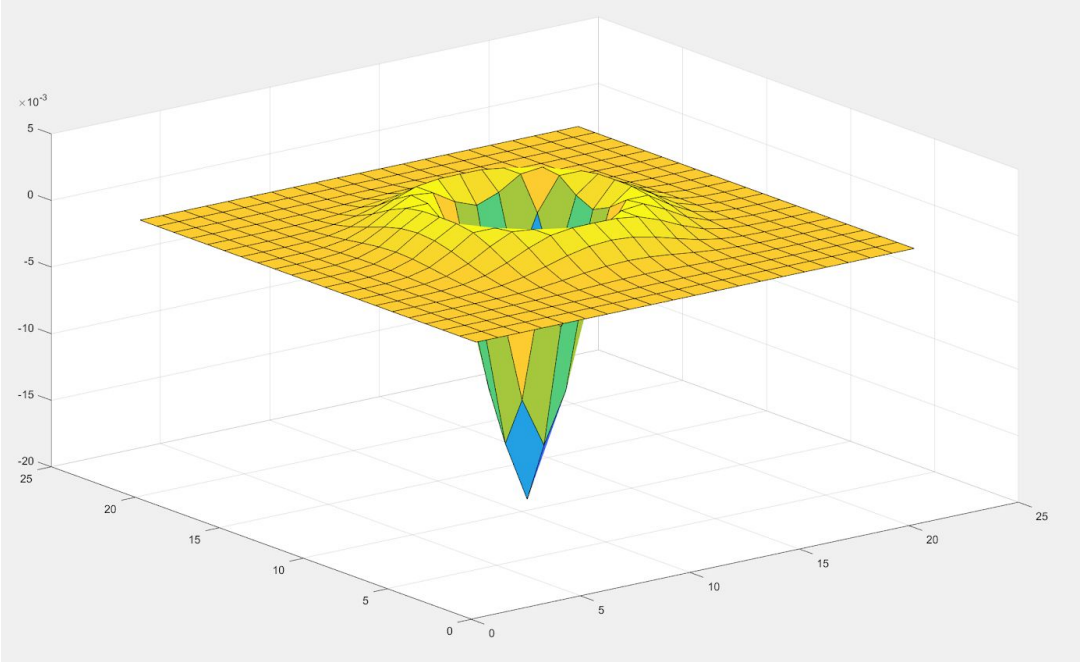
make2DGaussian(9, 1)



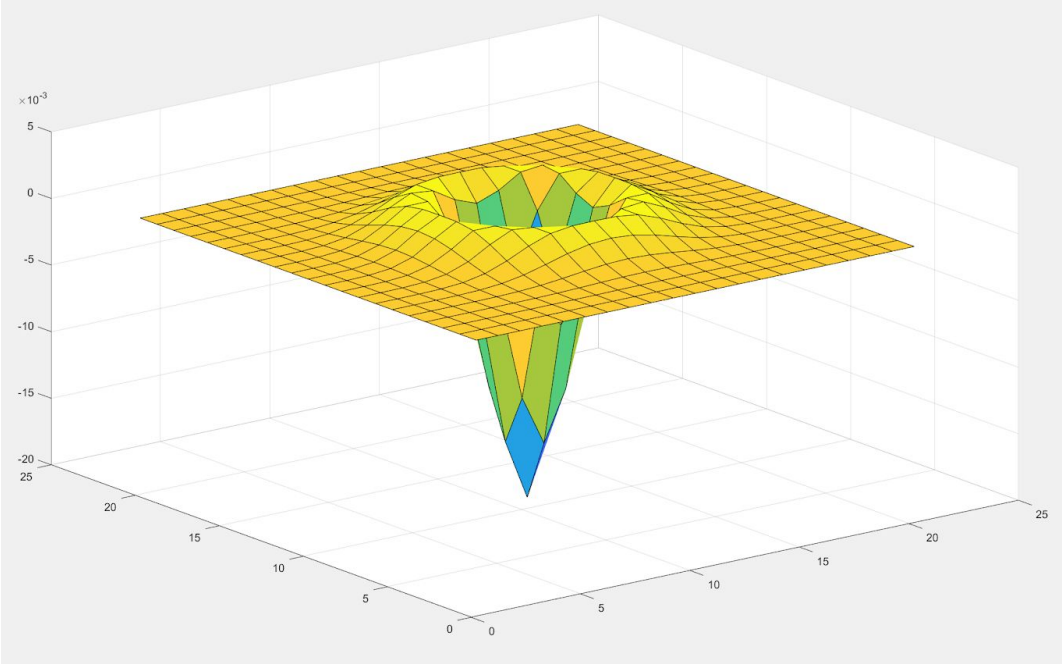


B.

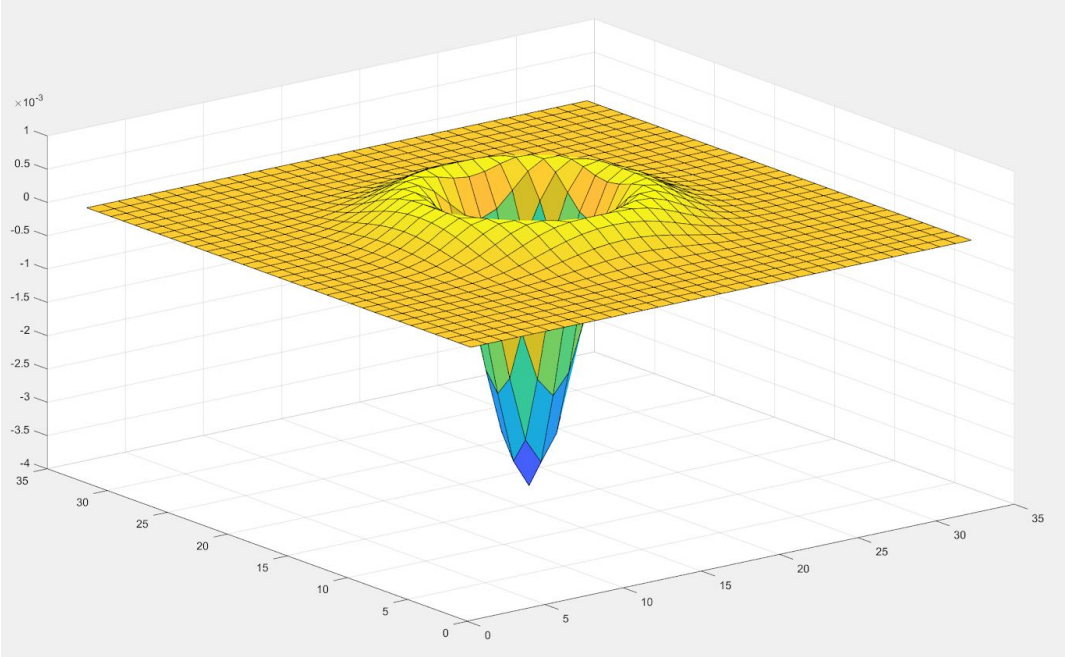
fspecial('LOG', 21, 2)



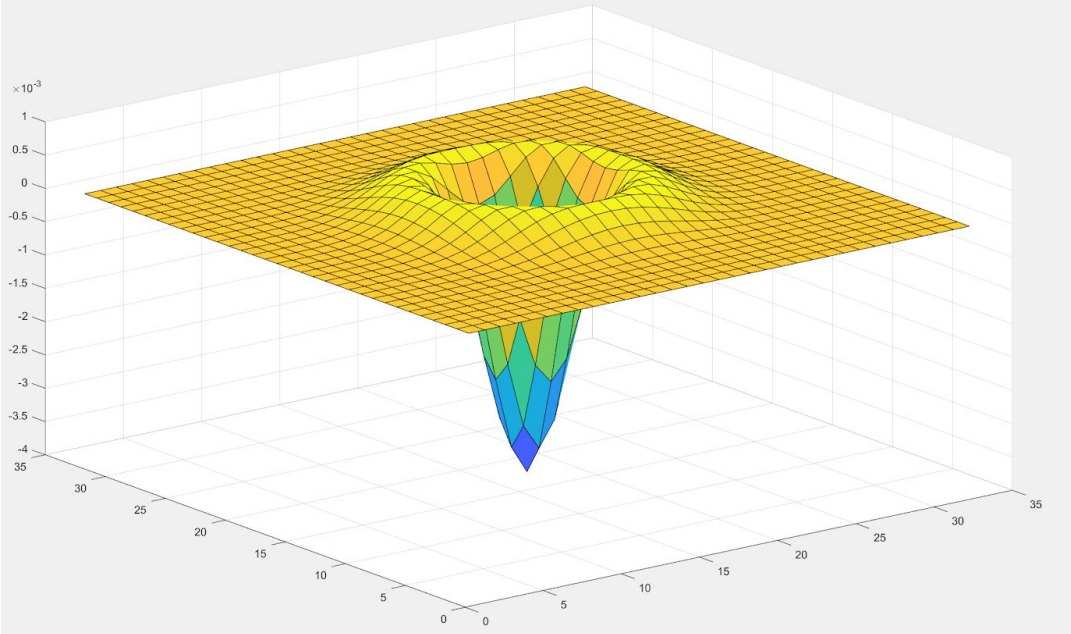
make2DLOG(21, 2)



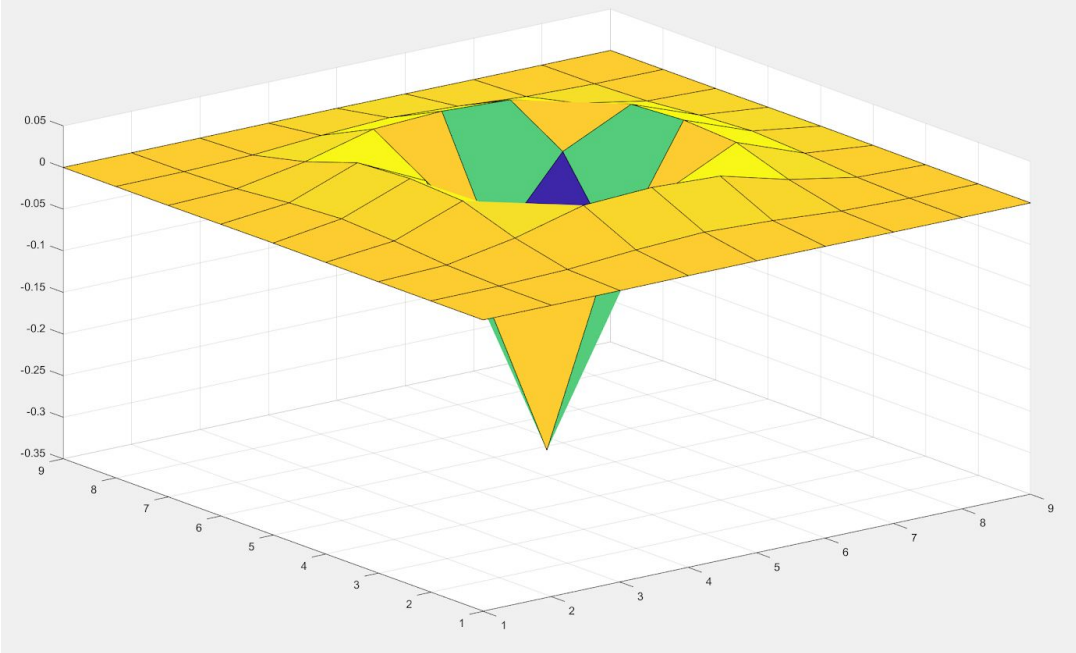
fspecial('LOG', 33, 3)



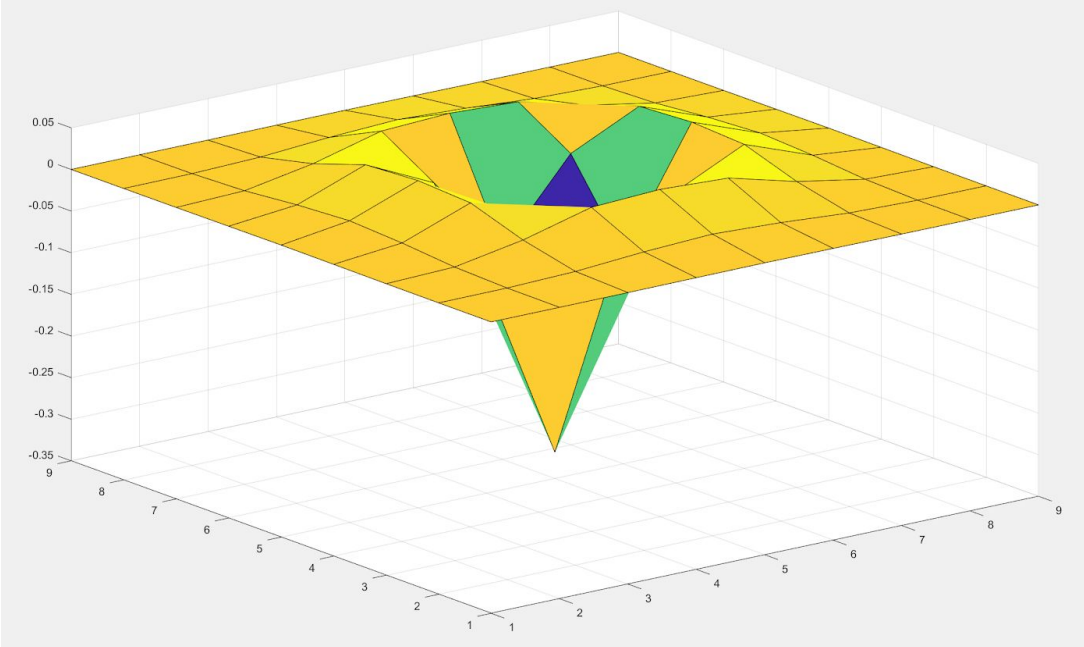
make2DLOG(33, 3)



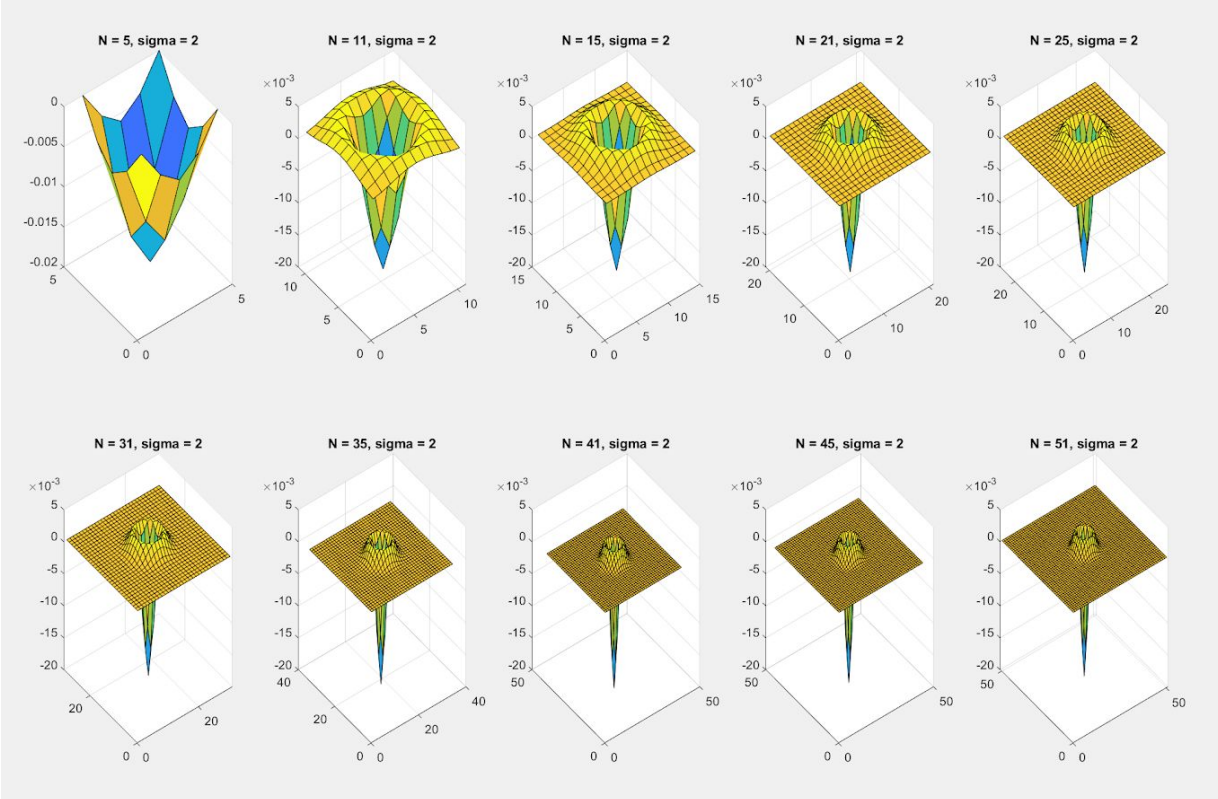
fspecial('LOG', 9, 1)

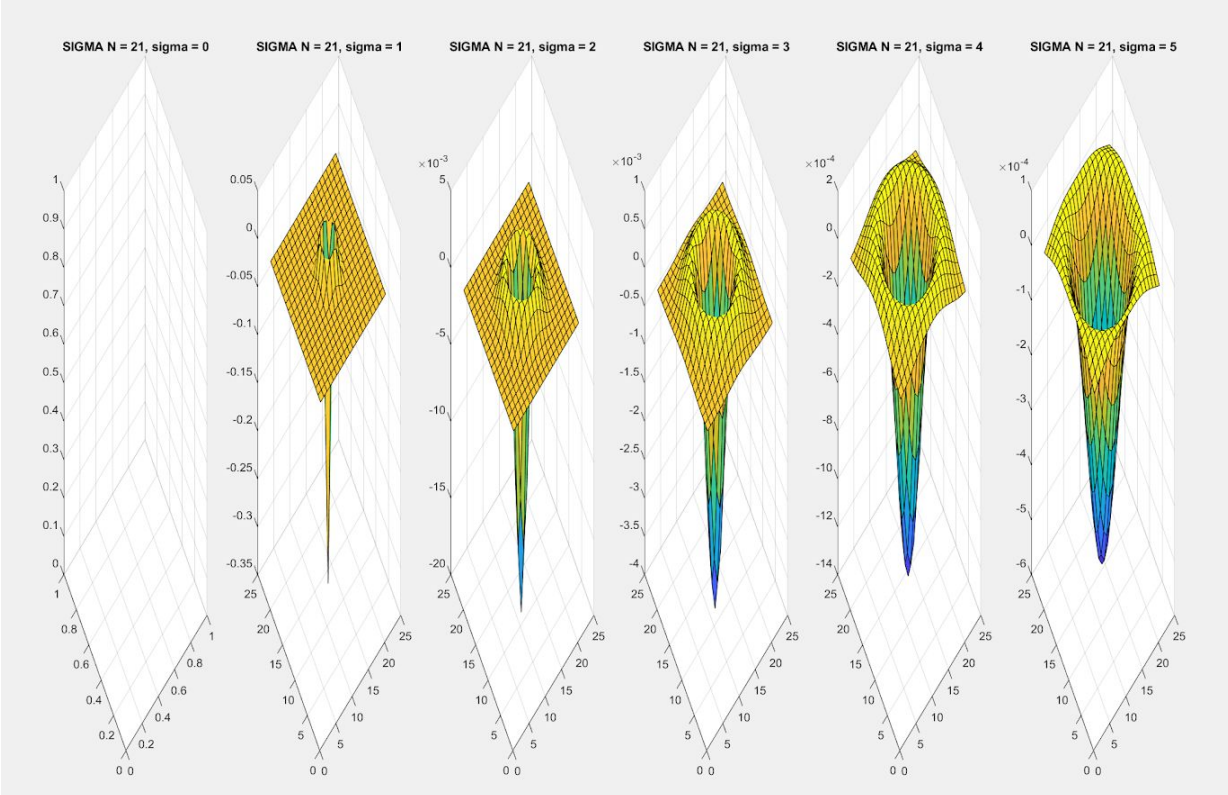


make2DLOG(9, 1)



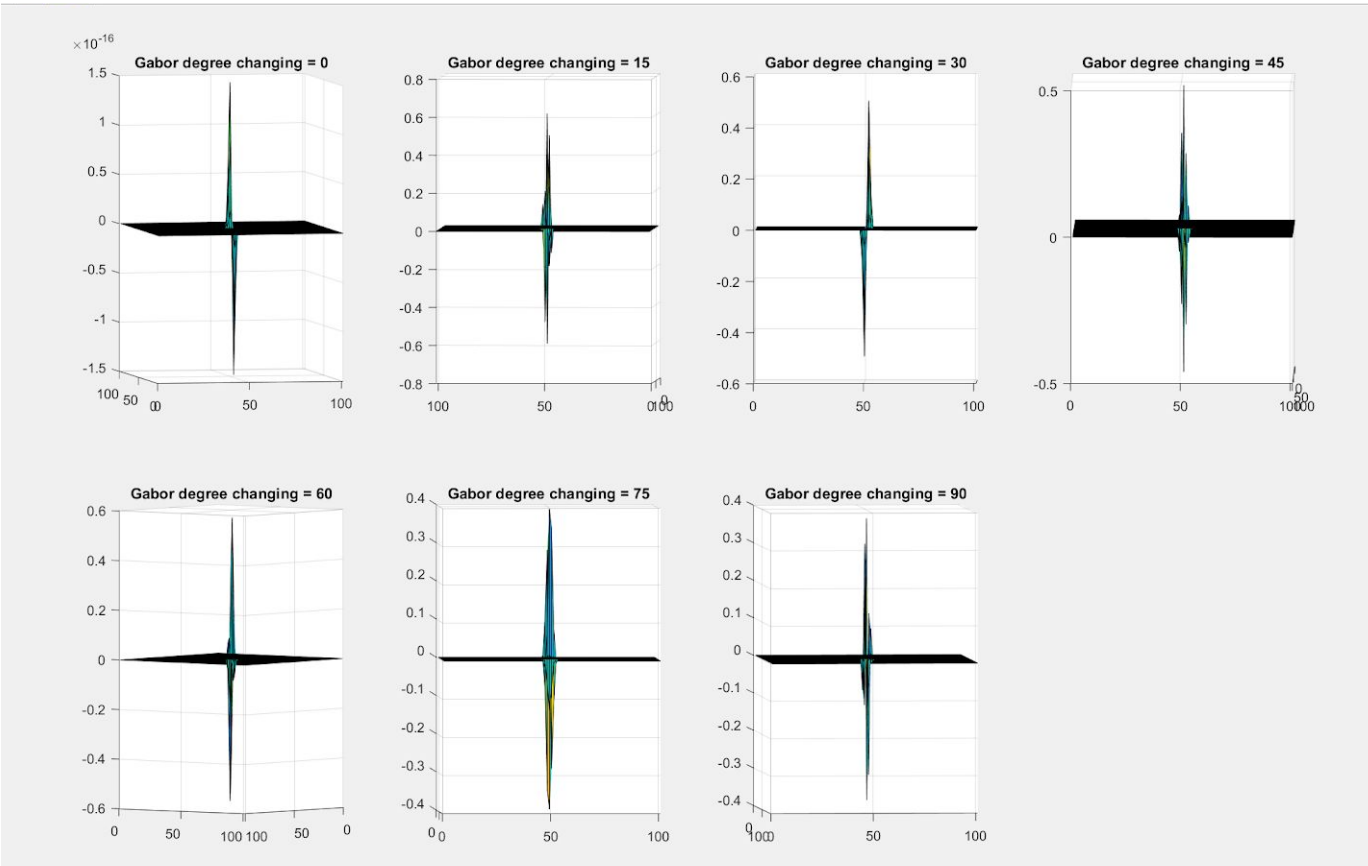
Visualizations for changing N values in the first image and sigma in the second.



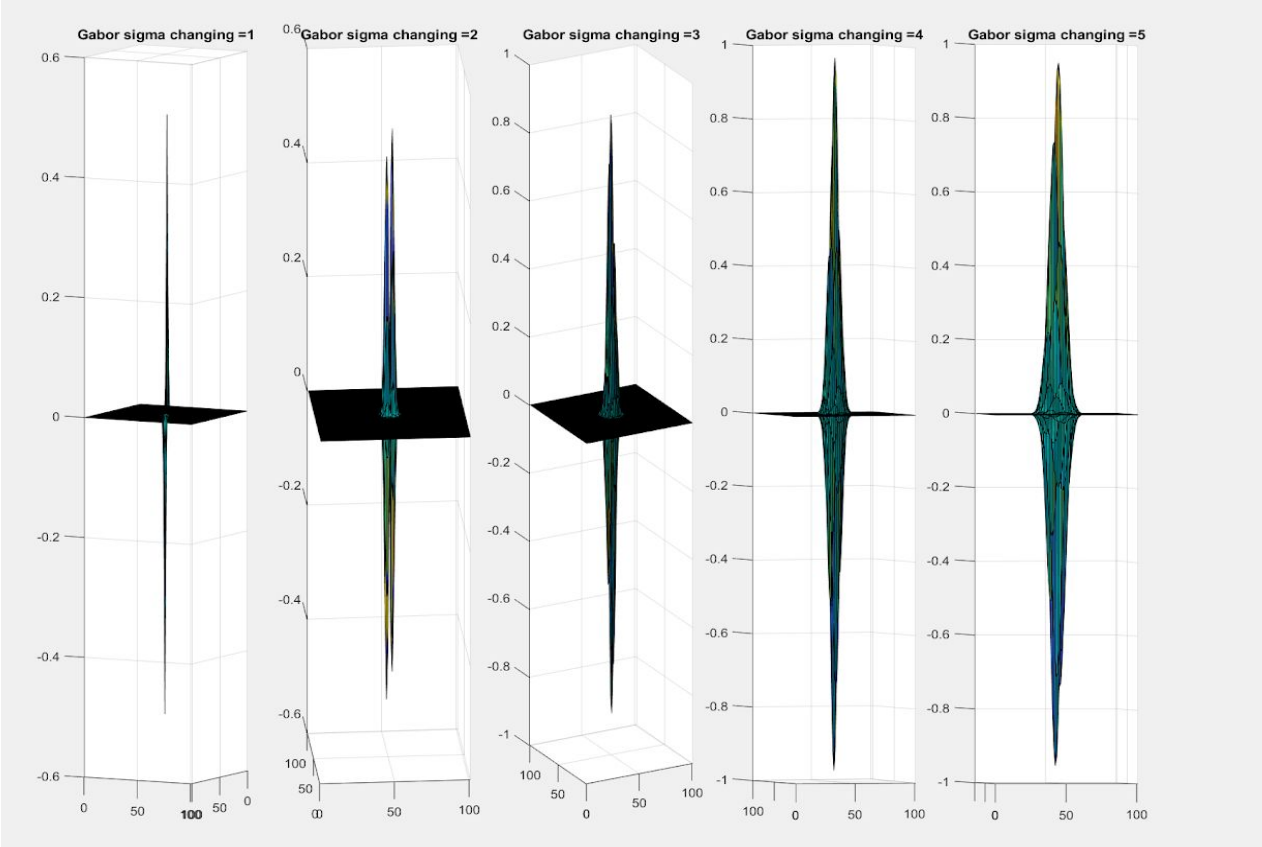


C.

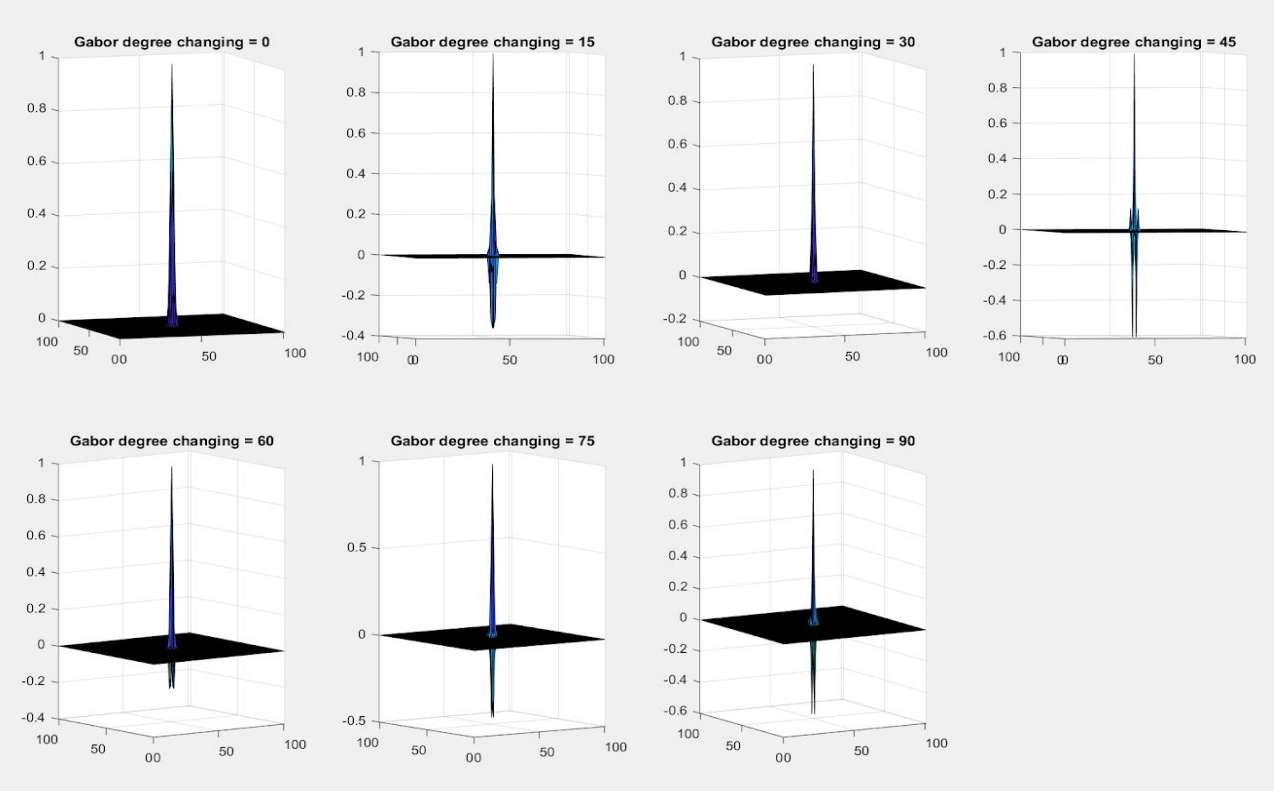
Gabor odd orientation changing:



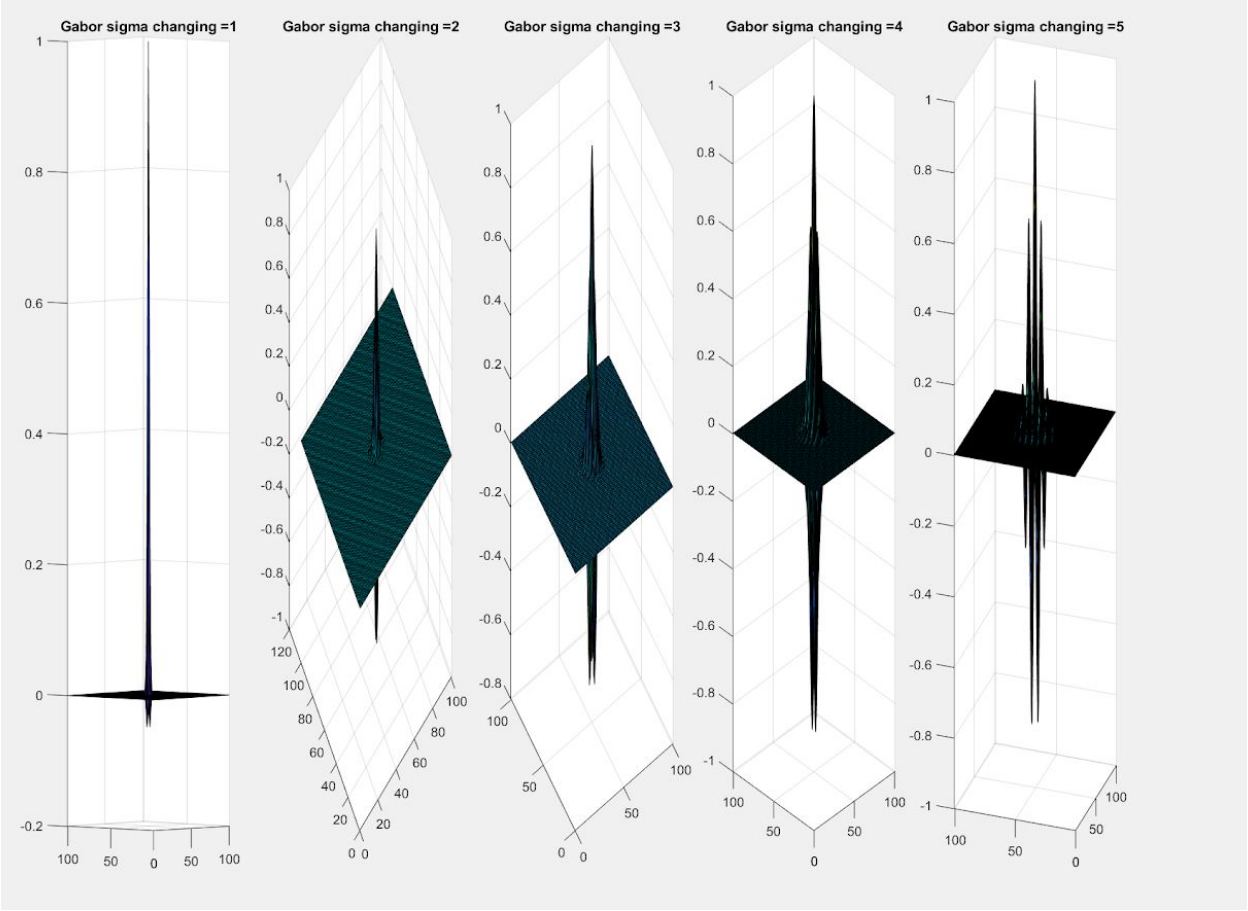
Gabor odd sigma/lambda changing:



Gabor even orientation changing:

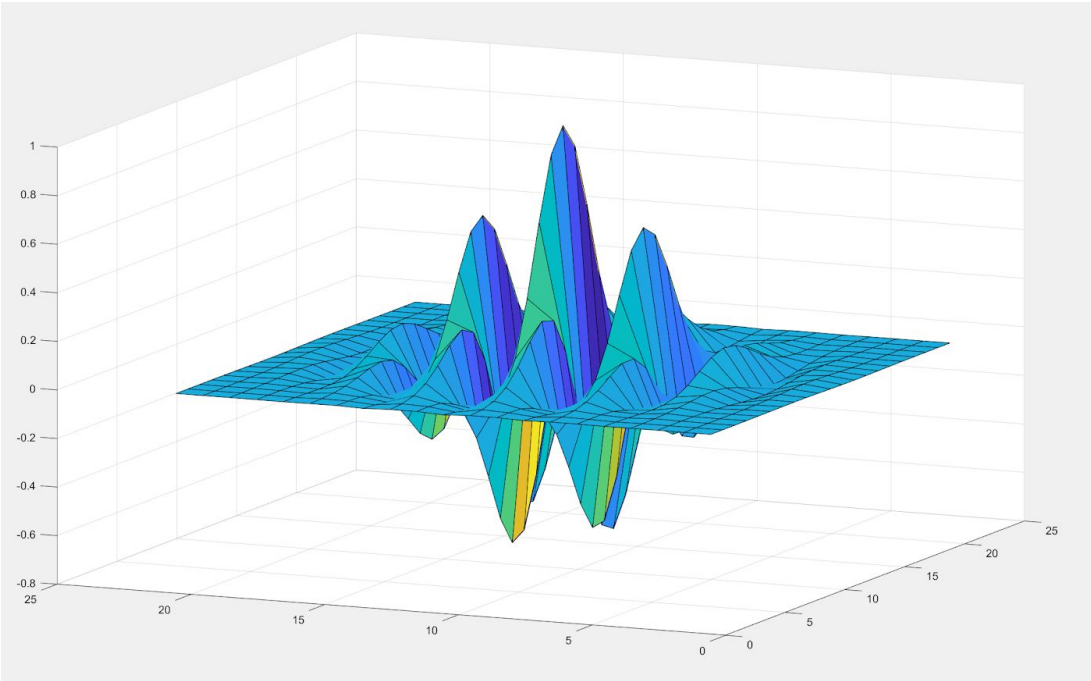


Gabor even wavelength changing (note sigma = lambda):

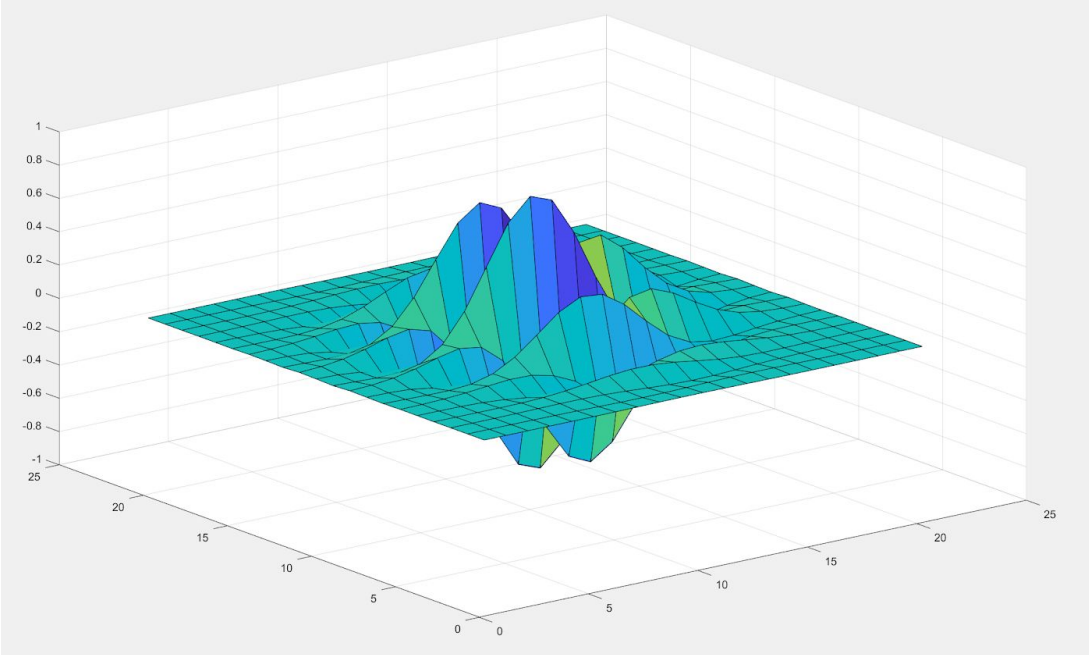


NOTE: These above are only for distant reference to the following:

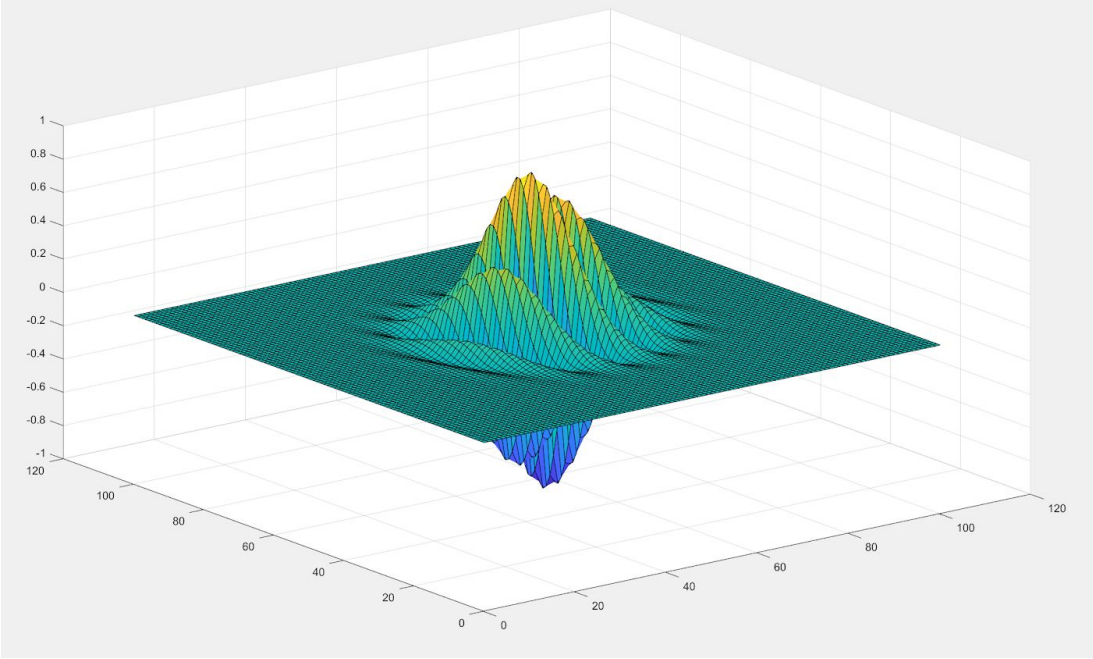
Even Gabor: (21, 3, 30)



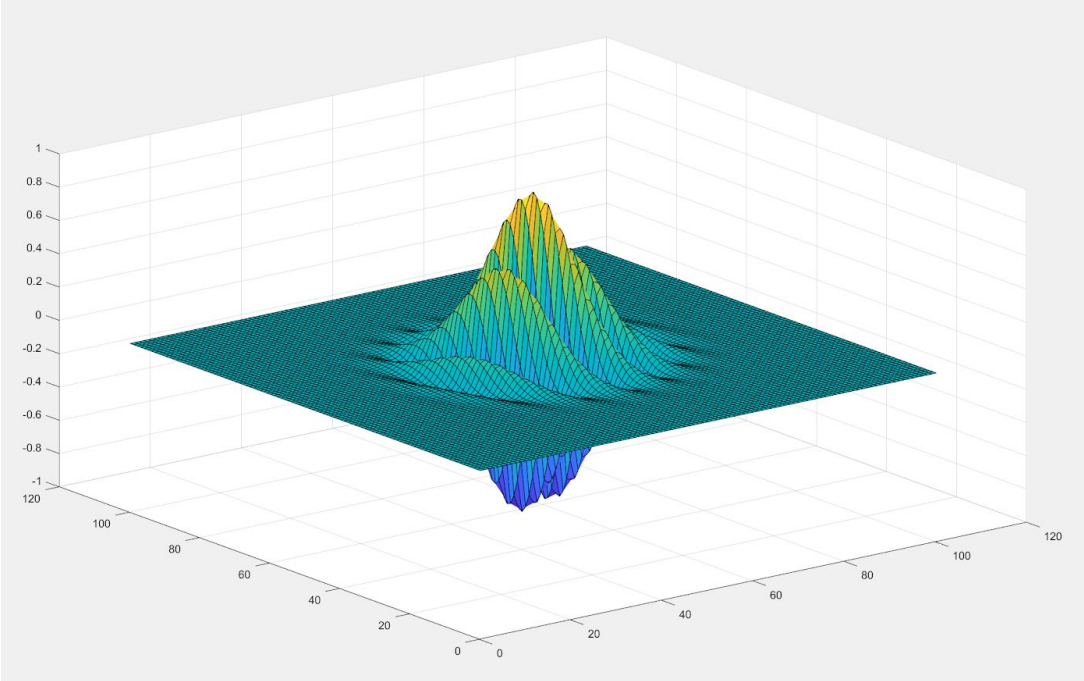
Odd Gabor: (21, 3, 30)



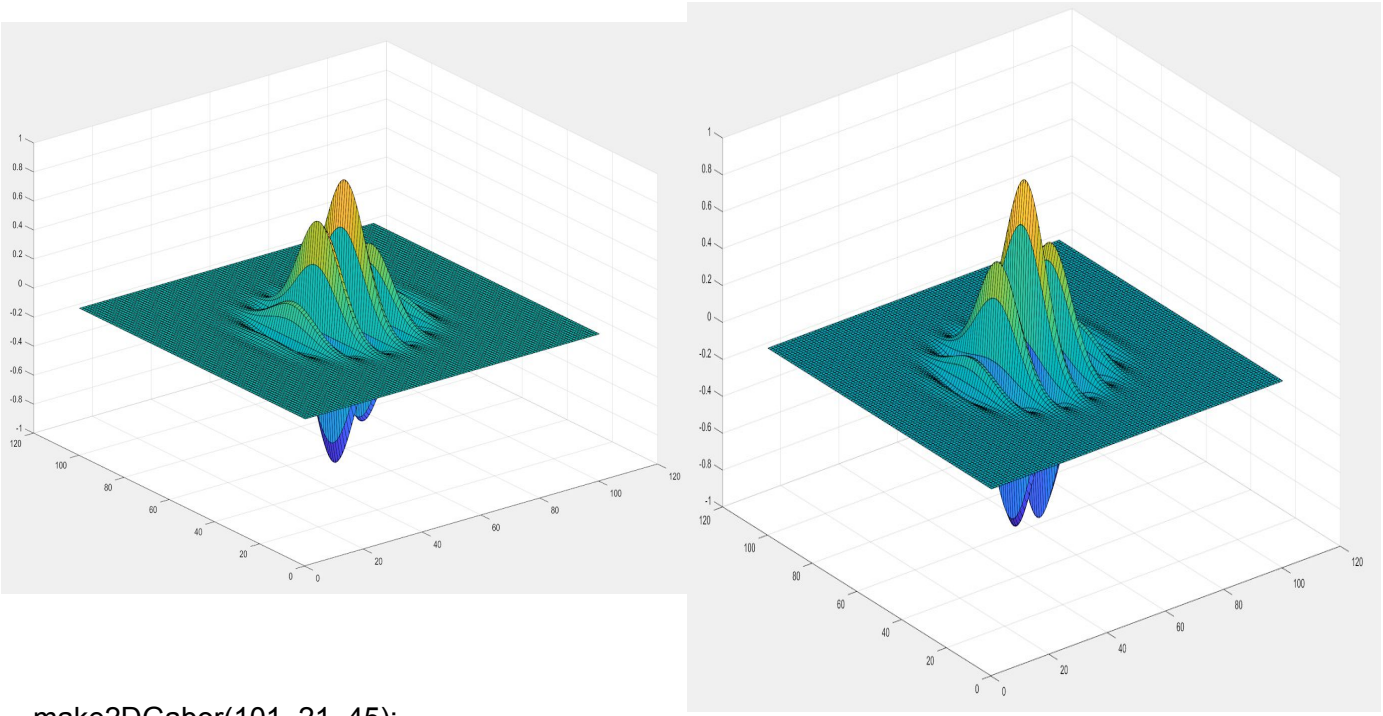
Odd Gabor: make2DGabor(101, 9, 60)



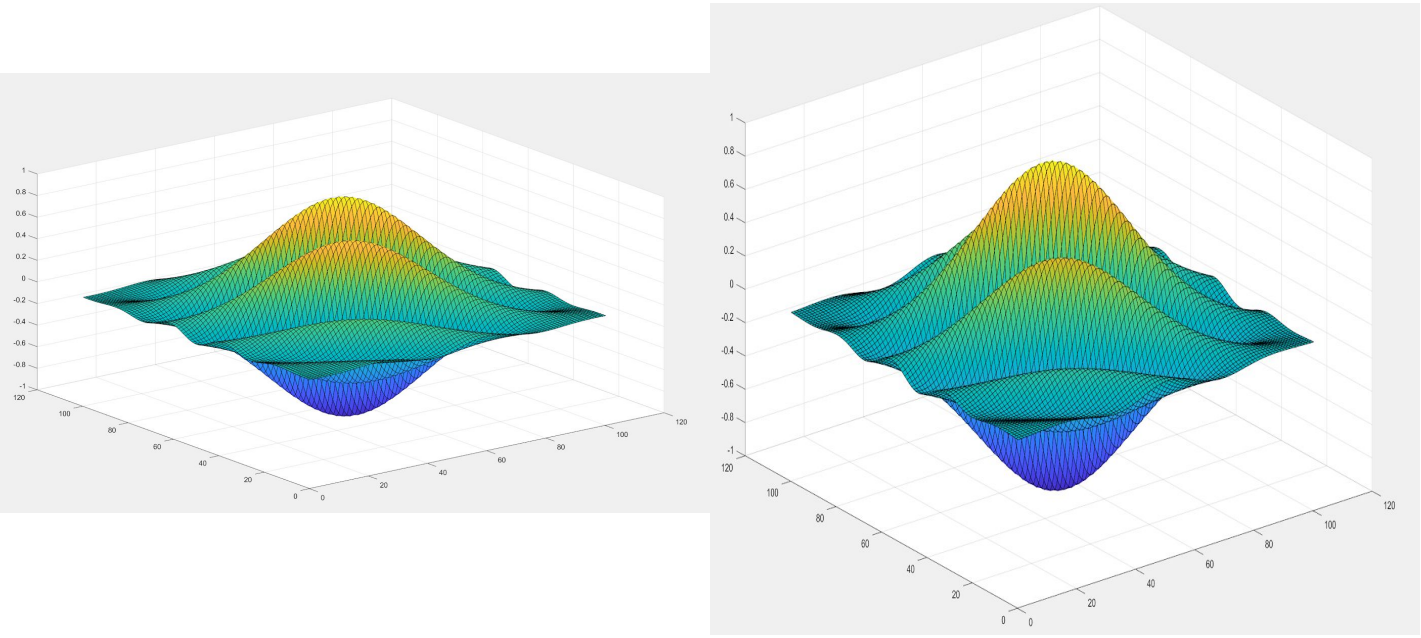
Even Gabor make2DGabor(101, 9, 60):



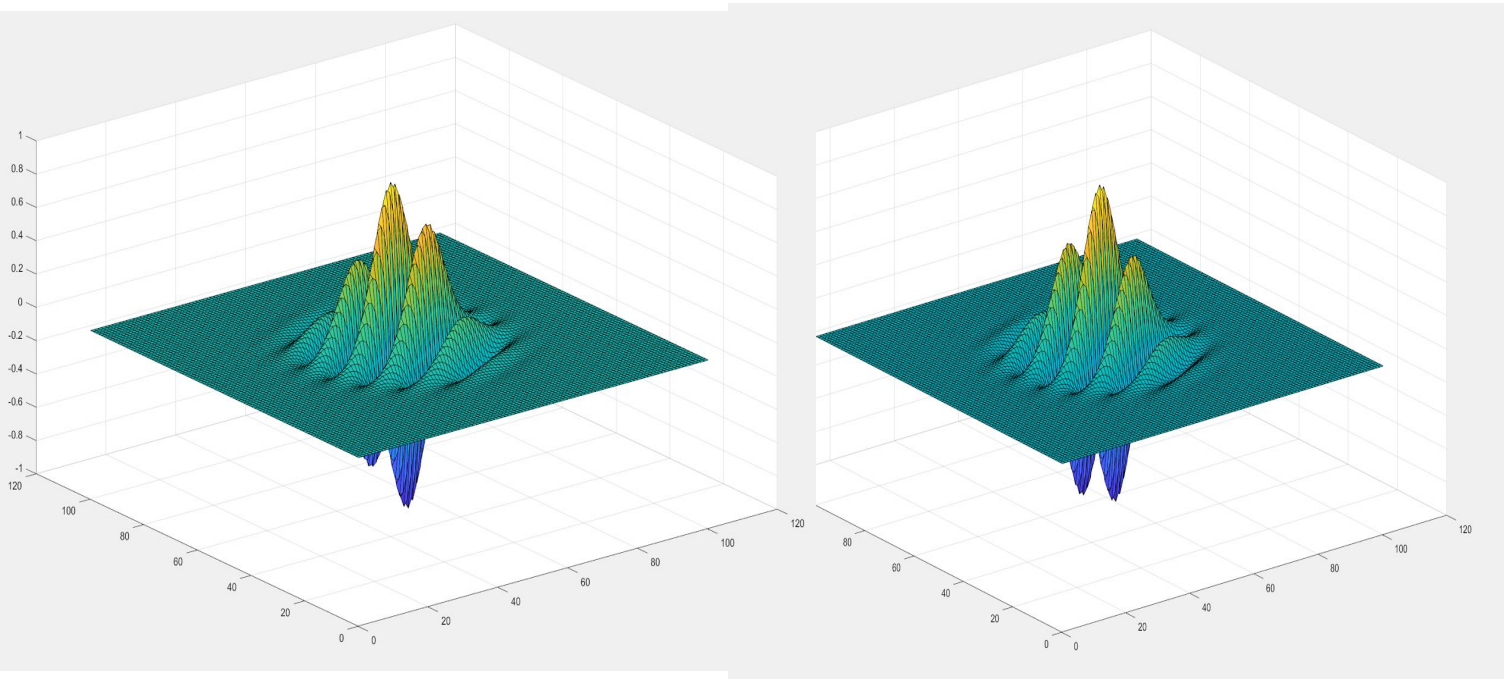
Degree: 0, 45, 90
Odd on left, even on right
make2DGabor(101, 9, 0):



make2DGabor(101, 21, 45):



make2DGabor(101, 9, 90):



++

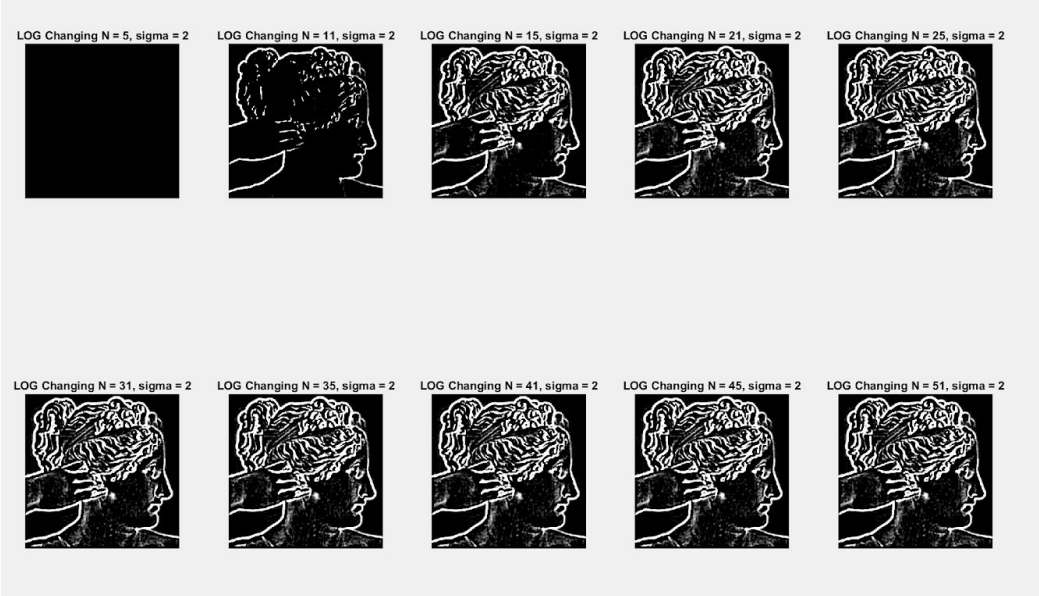
3. A.

Unless otherwise specified, the call of make2DLOG is:
make2DLOG(21, 2);

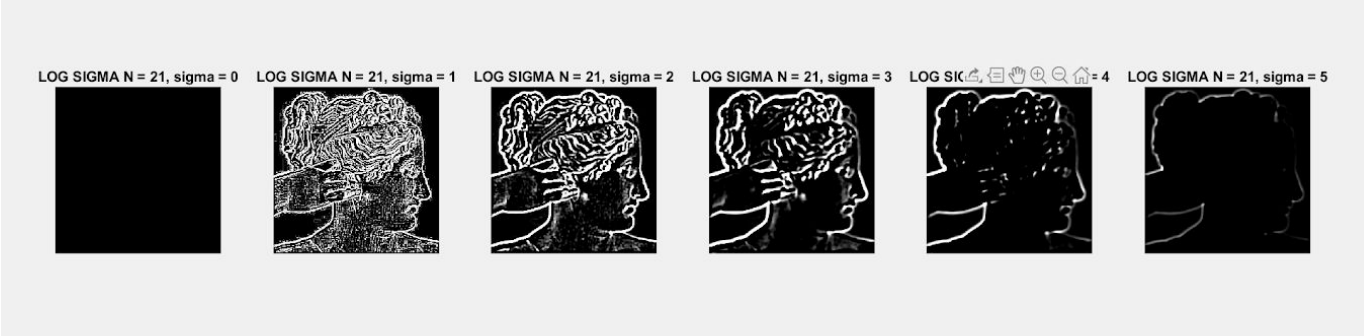
After some initially random trials, it was found that (21, 2) was the best filter arguments to use in my LOG, therefore I re-did my calculations using that as my “base” for the variations - something to check against to see improvements or if the results worsened. That second round of tests is what you see below. (21, 2) was detailed enough to show off the small edges, but not so detailed as to fill the image with noise.

Here, I show the different consequences of altering both sigma and N separately, in order to better aid my analysis.

Convolved with varying N's, but sigma = 2

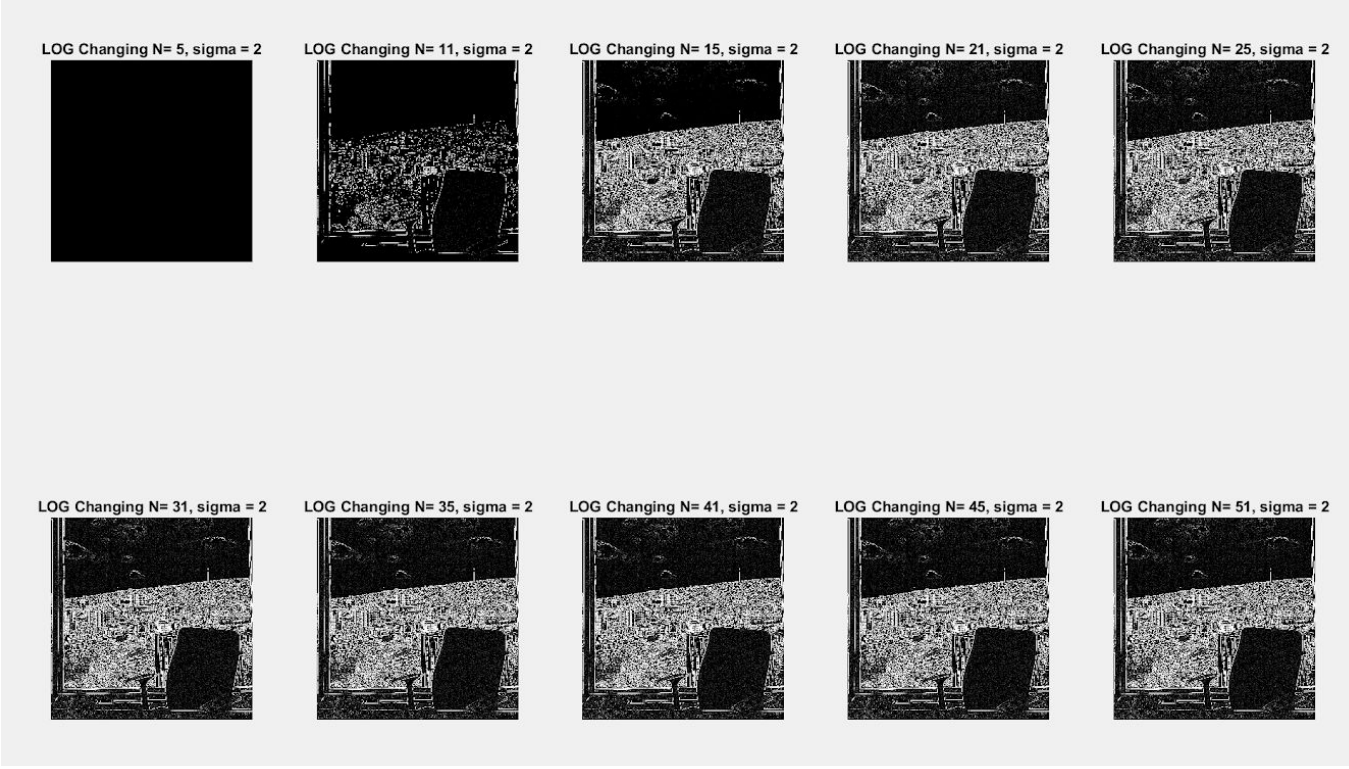


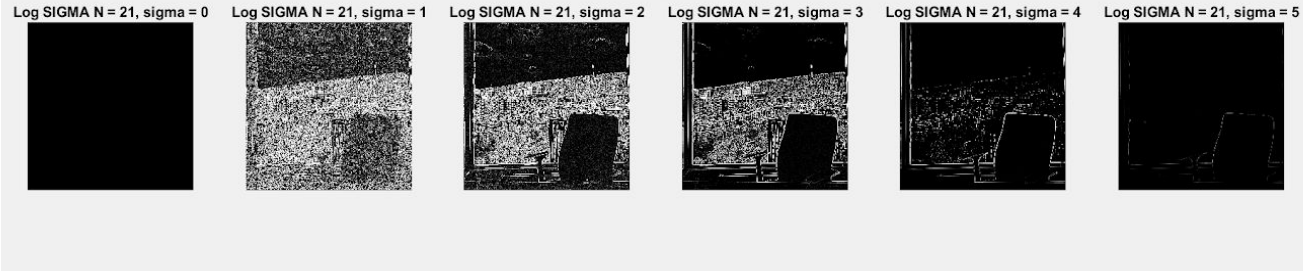
Convolved with N = 21, but sigma varying:



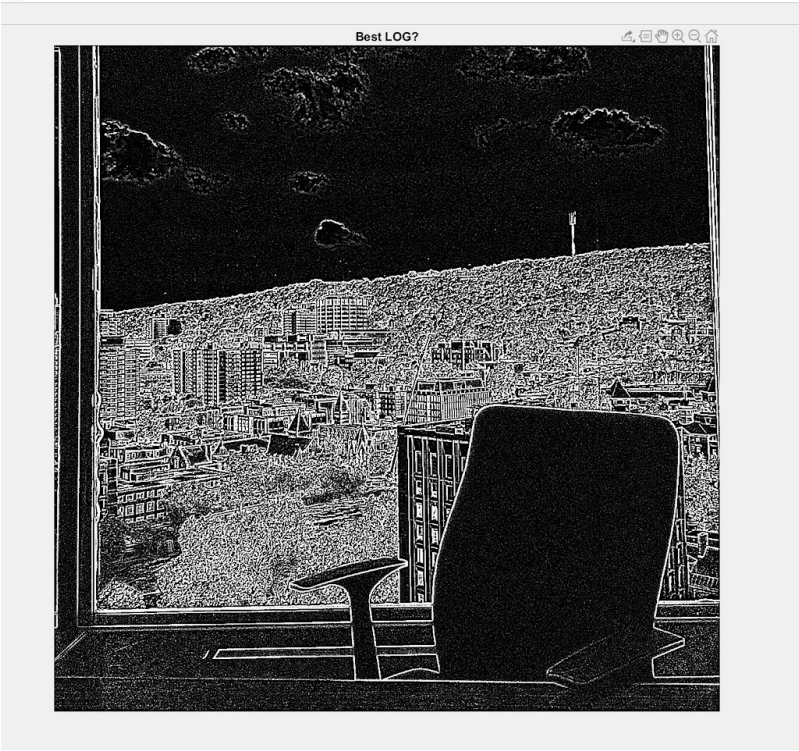
make2DLOG(21, 2) - image1

Now I conduct the same tests and loops on the second image:



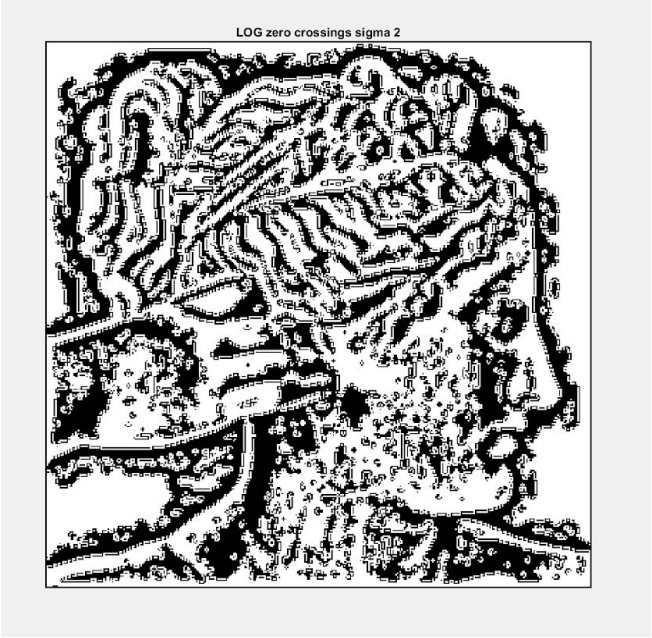


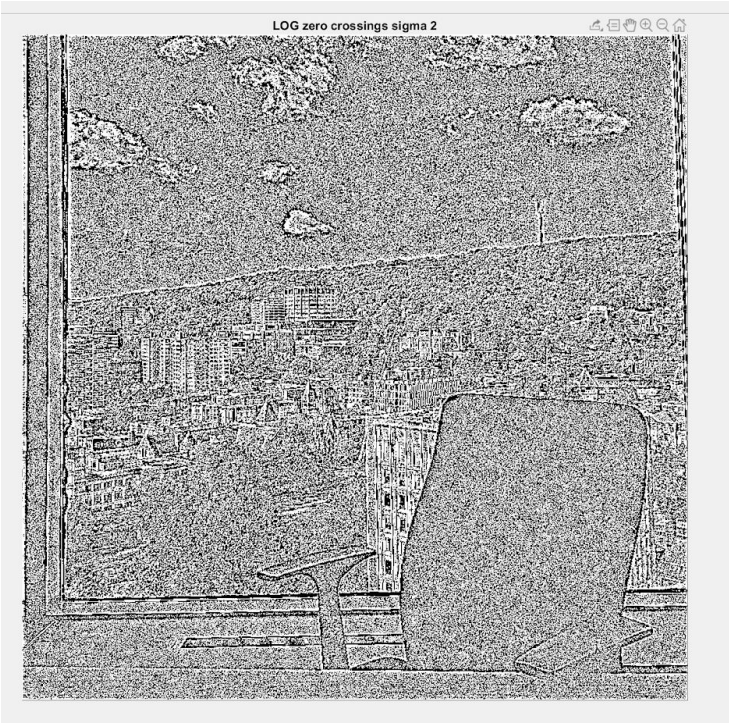
It was after the second round of testing that I noticed the changing of N does not really matter within quite a large range of a given sigma. For a given sigma, changing the N value in small ways appears to have almost no effect on the overall image.



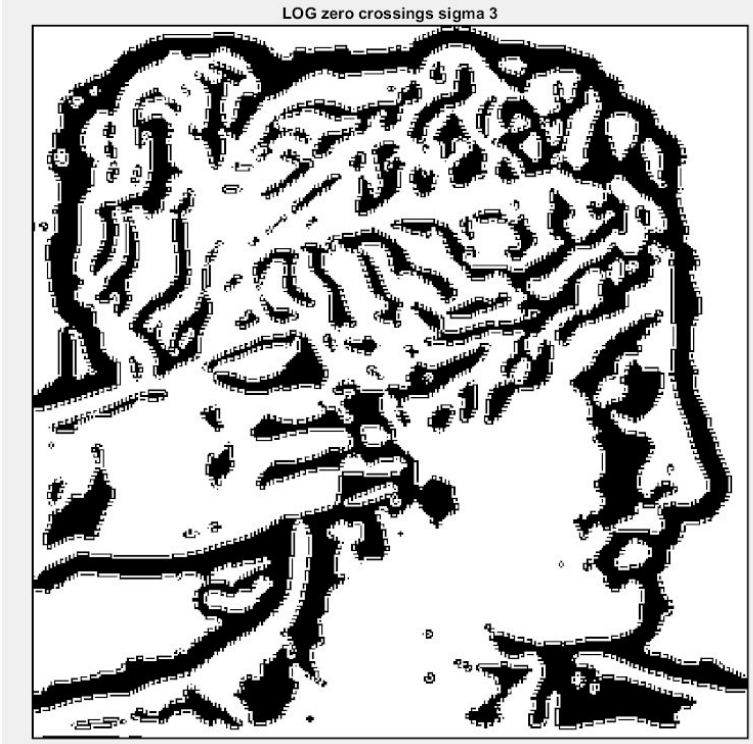
make2DLOG(21, 2) - image2

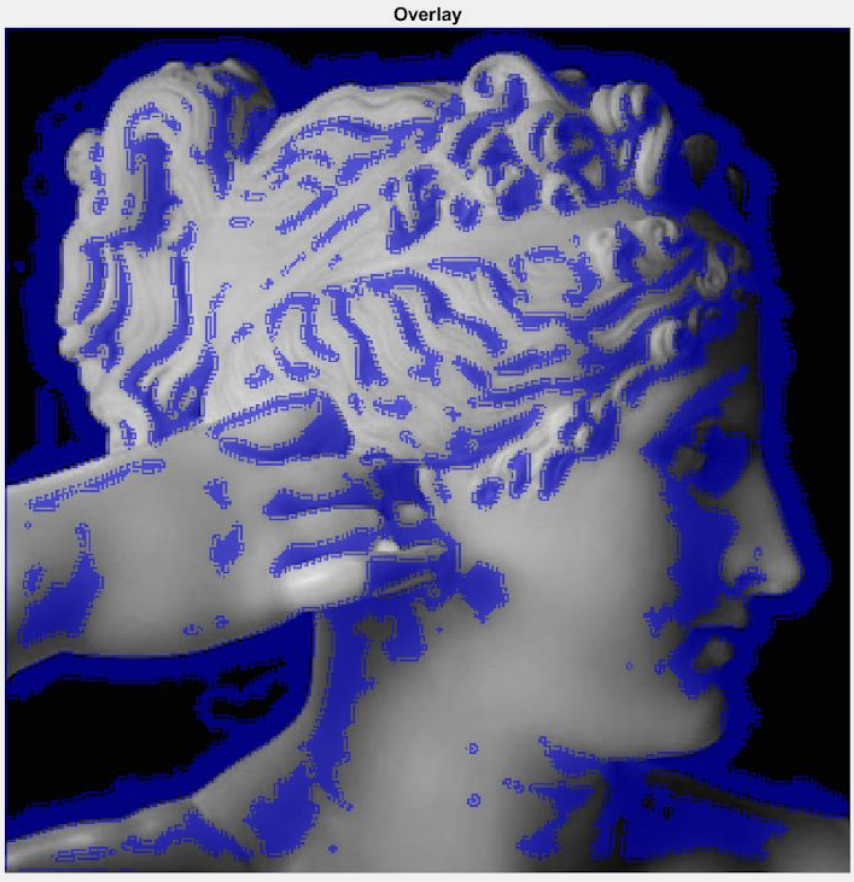
B.
Zero Crossings:
These images are the zero crossings on my colvoled images using:
make2DLOG(21,2)



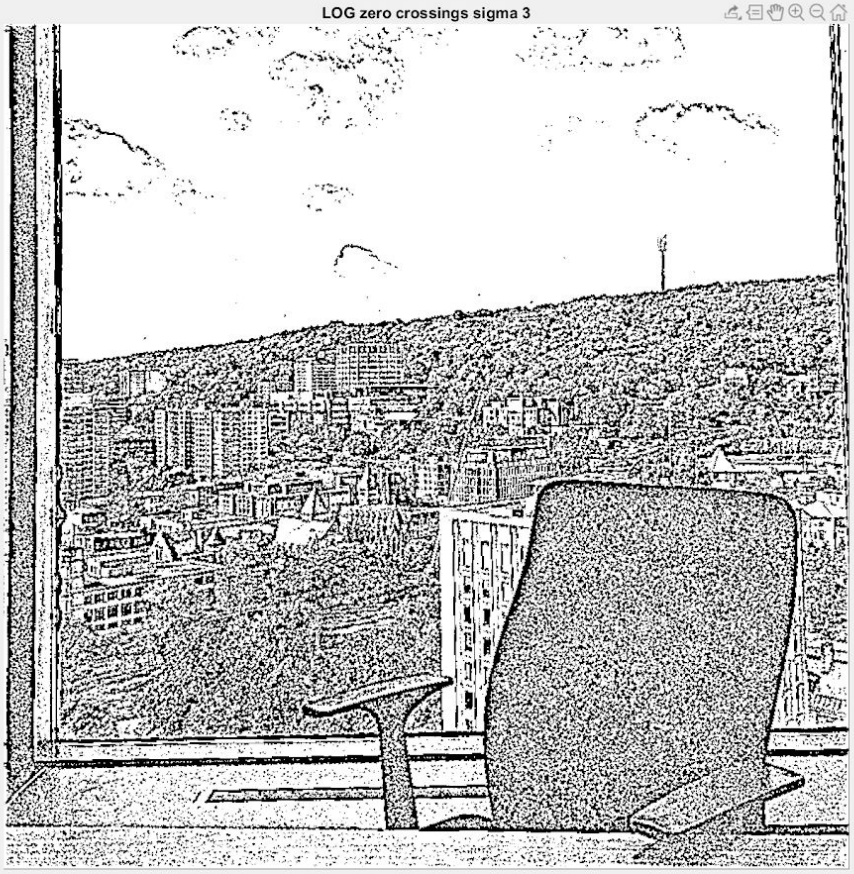


When the sigma was 2, the zero crossings seemed to pick up too much noise, so I decided to raise the sigma to 3, so that the zero crossings would have less noise-pickup.

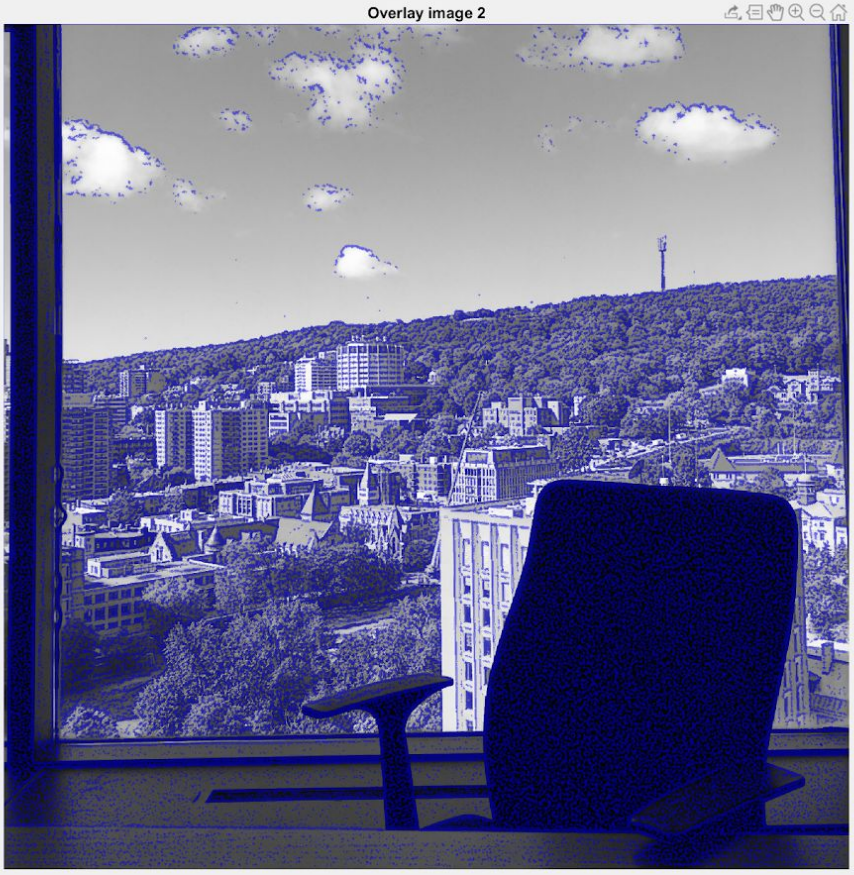

LOG zero crossings sigma 3


Overlay

LOG zero crossings sigma 3



Overlay image 2



As noted in my code comments, an inversion algorithm was required to apply the labeloverlay matlab function. Please see code notes for more information on why that was required.

C.
Analysis:

For the Laplacian of the Gaussian, I stared by simply making loops that alter each of the two arguments to generate different filters, by reasonable increments (as can be seen in my code). From there, I was able to quickly visualize which images had the best resolution, edge detection, noise reduction and so forth. It was from these trials that I found my typical testing parameters of make2DLOG(21, 2). That then became my “base case” where I then retested each parameter in order to see if improvements could be made.

When deciding on what N to choose, I knew I had to choose dimensions that took into account that +- 3 sigma is 98% of the area under a Gaussian, and 95% for +- 2. I wanted to have a large enough filter to move over the image quickly, regardless of size, while at the same time small enough to show discrepancies in dense areas. Sigma seemed much easier to choose as seen in the graph of images varying sigma 1:5. At 1 there is too much noise being detected, and at 5 the threshold is just too high for the filter to detect anything. 2 seemed to be the sweet spot.

However, it was not ideal in regards to the zero crossings. When I took the zero crossing of the Laplacian of the gaussian, my zero detection algorithm picked up too much noise again, so I decided to raise the sigma on my LOG before looking for its zero crossings - to cut down on unnecessary information within the image.

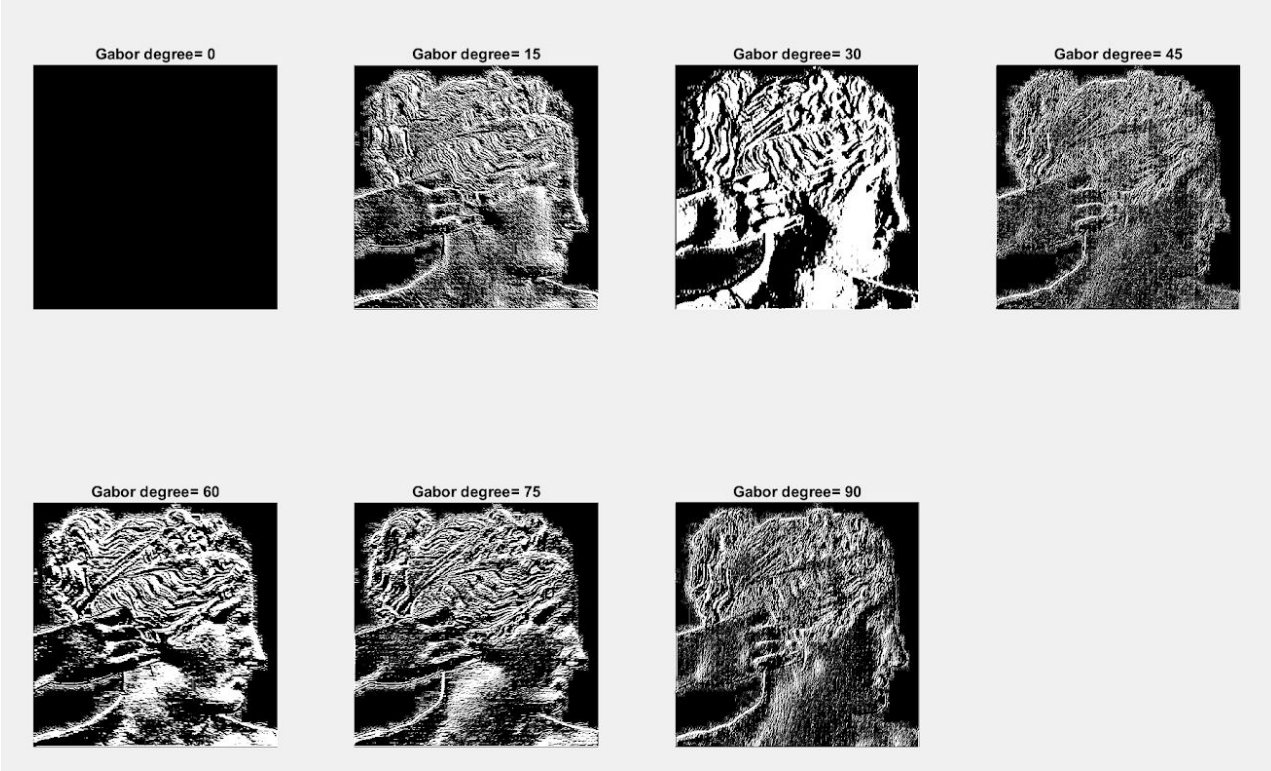
Overall, I was very pleased with the amount of detail the algorithm could detect in regards to edges. It is clear that shadows are commonly taken as separate objects to the LOG algorithm, and that overlapping objects may be mis-interpreted in an object recognizing program.

Also, as noted in 1c, this algorithm seems to only hold up when the condition of linear variation holds, as it is mostly based off the Marr-Hildreth method of edge detection (it is using the LOG operator after all).

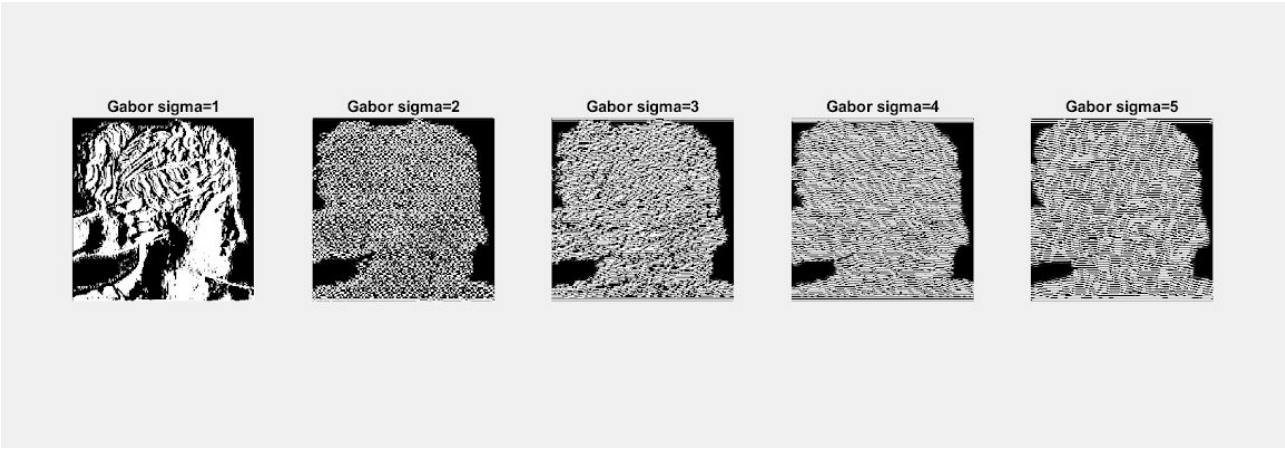
The imageFilter3.m file should execute everything seen above and below in case of any confusion or sourcing concerns.

D.
REPEAT 3A-3C
NOTE: The base case used varies a bit more in the Gabor tests than in the LOG, please refer to the code to take any notes of deviations. While N stayed about 21, the lambda changed between 1 and 2, because in some pictures/orientation combinations, 1 was favored over 2 and vice versa. ALSO, anytime sigma is in the header of a figure, sigma=lambda.

Gabor (21, 2, i)- Varying degrees:



Gabor (21, i, 30) - varying sigma/lambda:



Gabor (i, 2, 30)varying size of N:

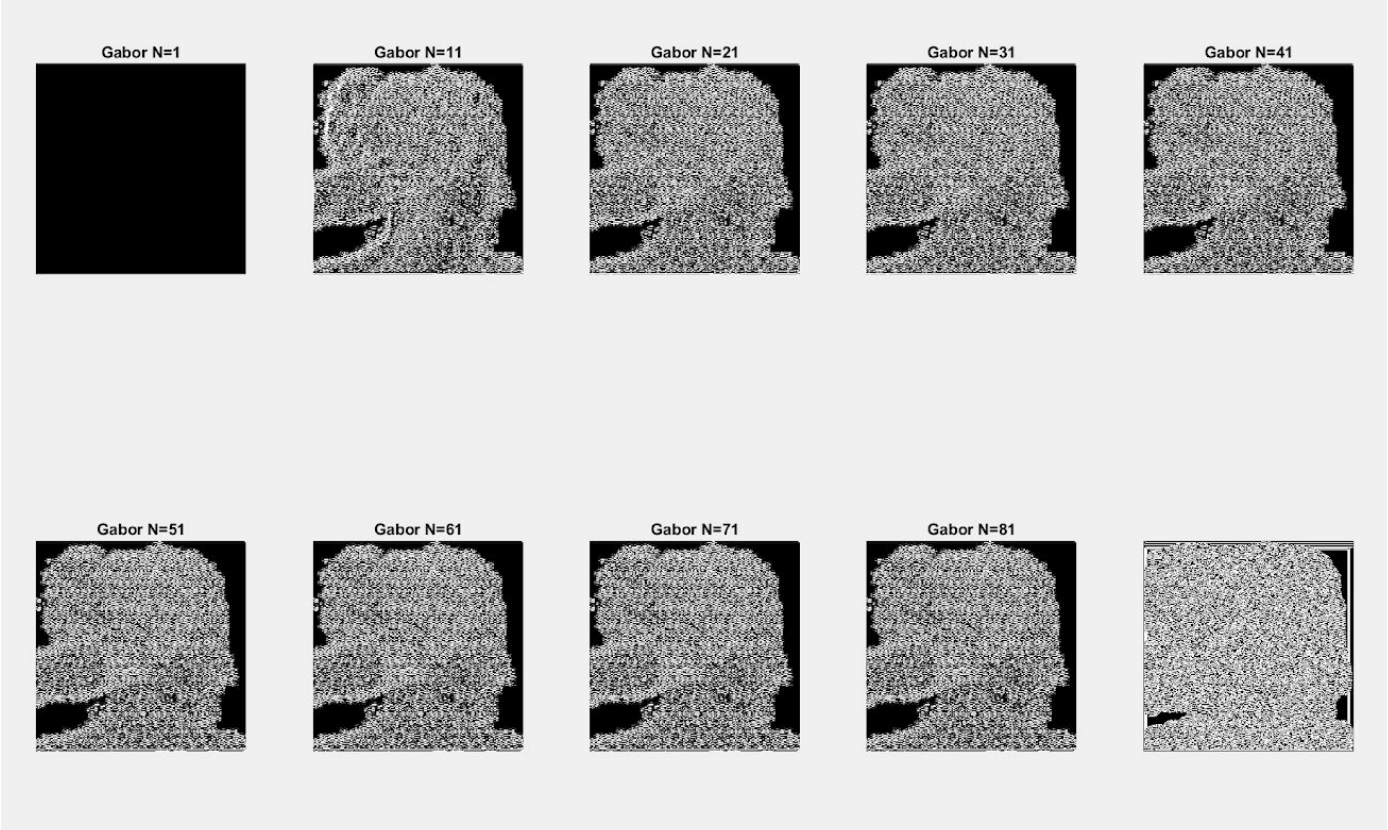
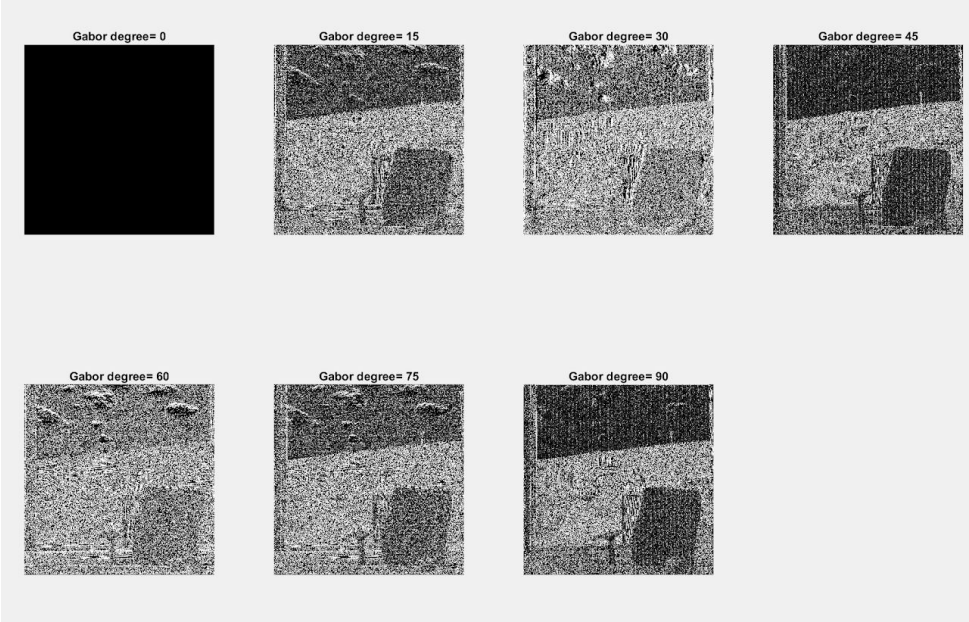
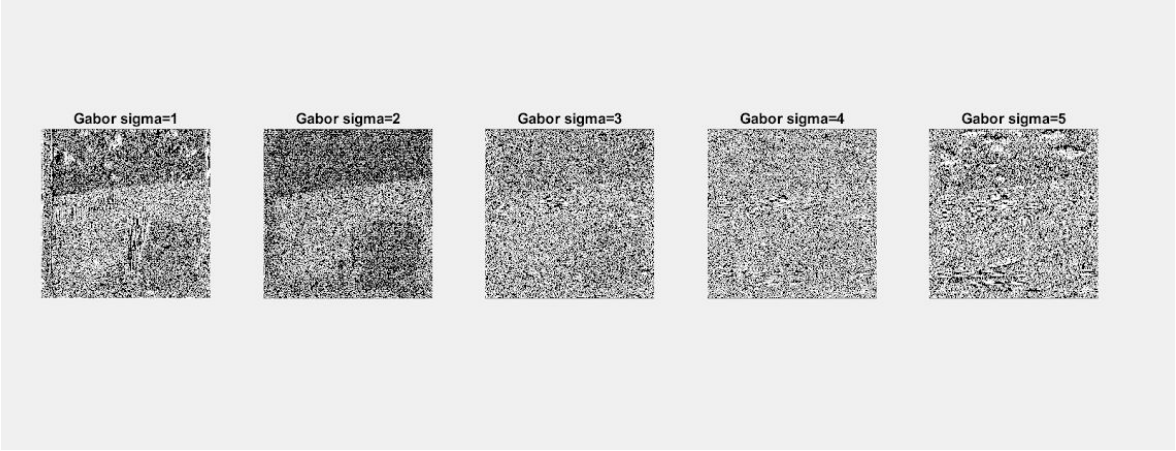


Image 2 (Same process as image 1 above):
Note, the standard angle used here was 30

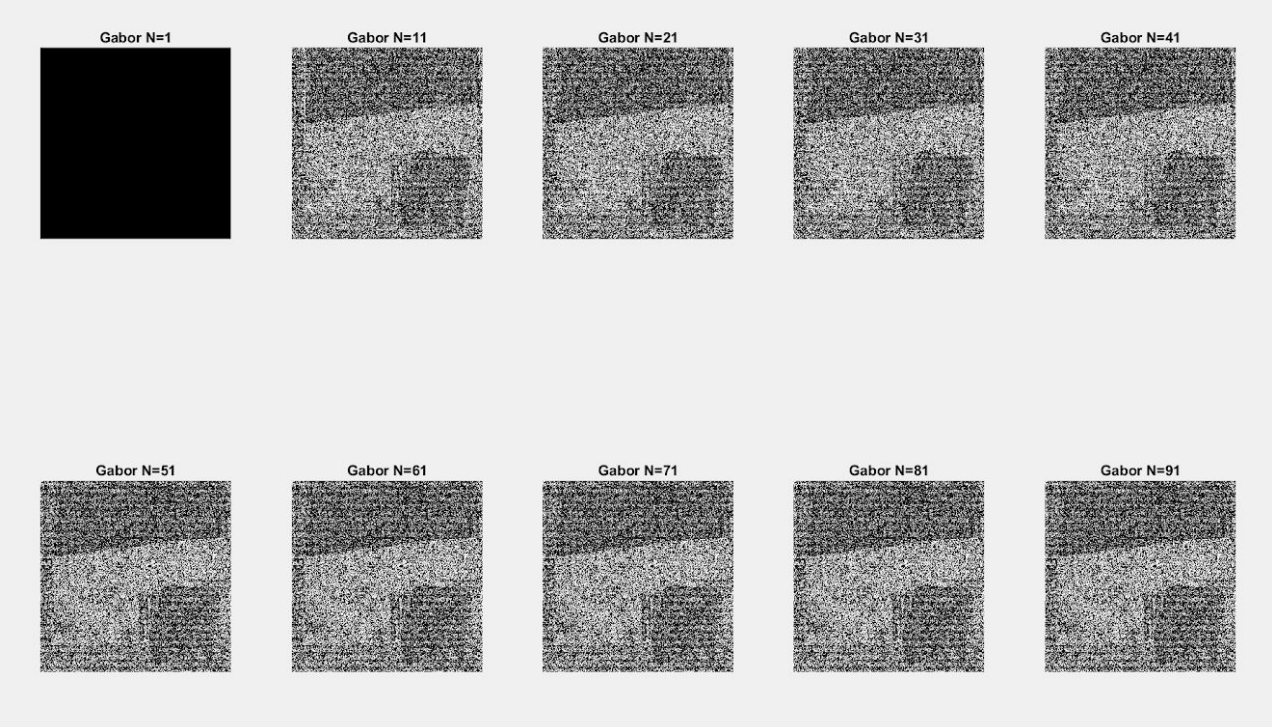
Orientation altering:



Lambda/sigma changing:

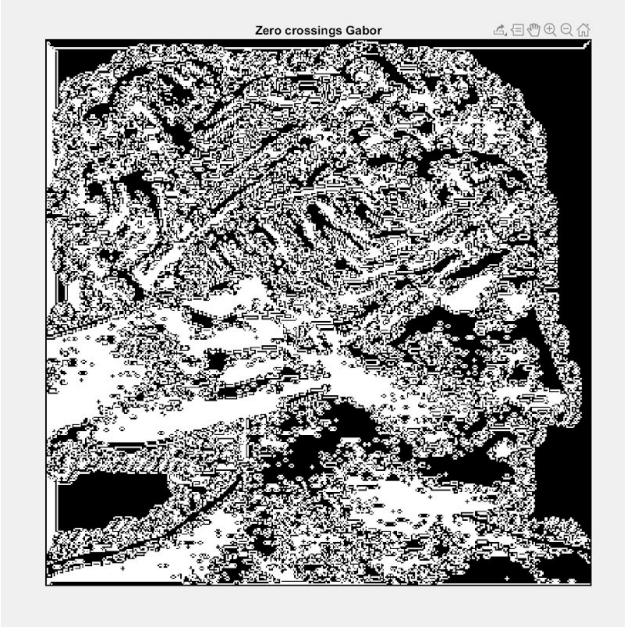


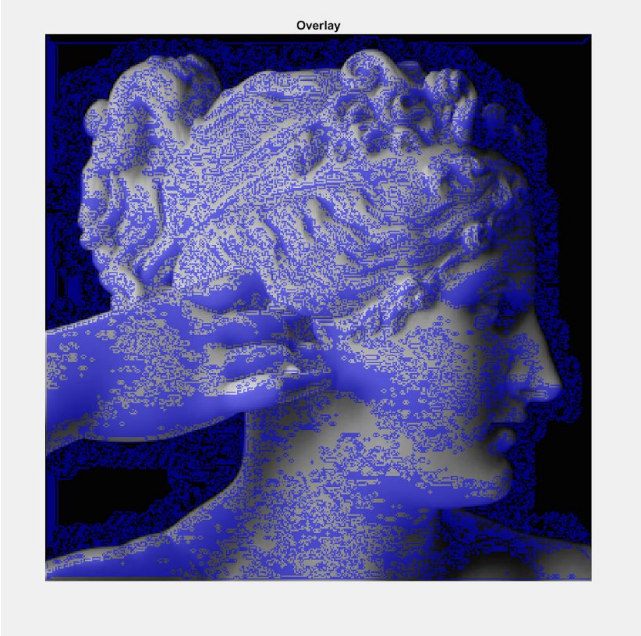
Altering N:



Best Gabor using following formula:
make2DGabor(21, 1, 60)
My 2nd round of testing basis point

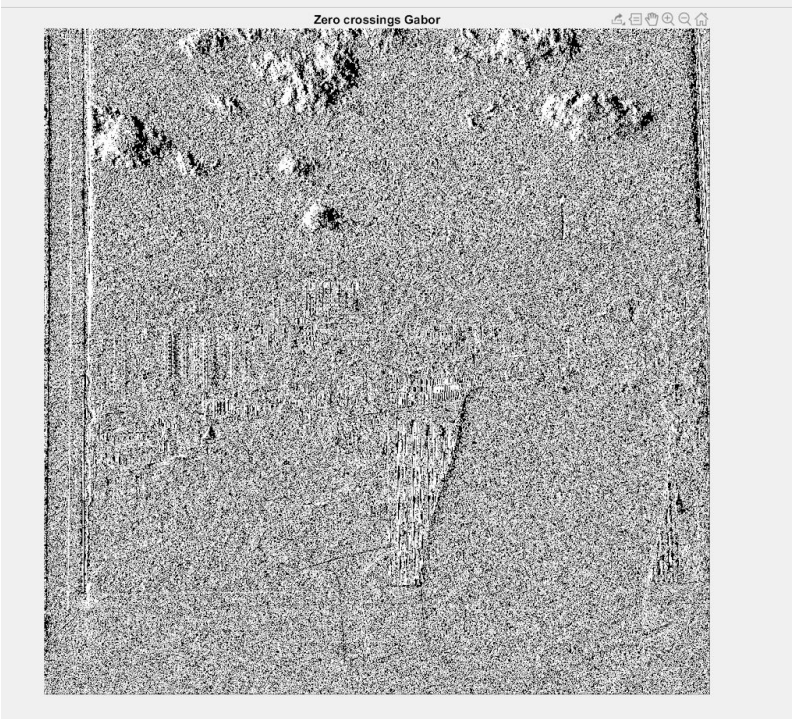
Zero Crossings:

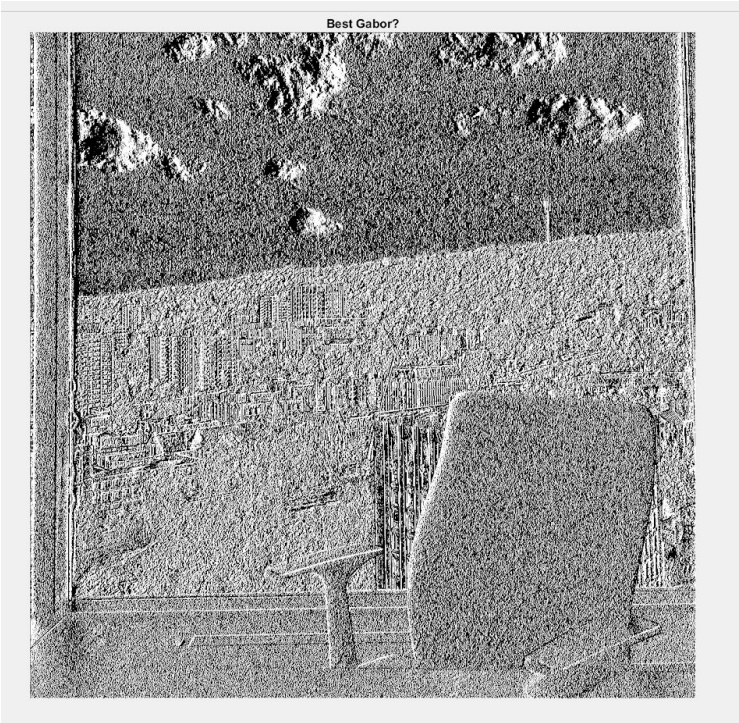




Gabor filter is much noisier than the LOG (in this single instance)

Degree was changed to 30 for image 2:





<-- make2DGabor(21, 1, 30);

D)
Analysis:

Similarly to my part A of this question, my base here was (21, 1, 30). However, due to the different angles used, there was no real “better” combination, hence my use of 60 and 30 for some orientation tests and 1 and 2 for lamba.

Overall I found the Laplacian to be better with edge detection alone, but the Gabor to be better when edges matched up with the inputted orientation (which makes sense). The Gabor filter would seem to improve over the Lapalcian if multiple angles were taken, then superimposed over one another, that would give a better result than one orientation alone. The Gabor also seemed to have more noisy outputs in general, possibly due to the “texture” of each image being picked-up on more than the Laplacian of the Gaussian. I’d imagine Gabor filters could be much more useful in object recognition, as you could use local orientations to better distinguish objects in a scene as compared to simply comparing edges. The LOG fails at detecting separate objects, it would appear, if objects were to overlap (one is continuous and one is hidden underneath), but having the orientation information would add more data to allow computers to better determine what is happening between objects in a given image.

The main relationship between the wavelength in the Gabor filter and sigma in the Laplacian is their main function. The higher the value its, the wider the main area under the curve for each gets. In the case of the Gabor, it widens the ellipsis’ used in the filter, so the higher you go, the more blurry the image gets, and less detail is achieved. Same with sigma in the Laplacian, the larger the sigma, the larger the area under the curve gets, creating a less precise edge detector, as it’s threshold for detection is so high it detects nothing. Whereas, having either of their functions too low allows each respective function to pick up too much of an image, leading to a mess of noise.