ECE 253, Solutions for Homework 2 (Fall 2022)

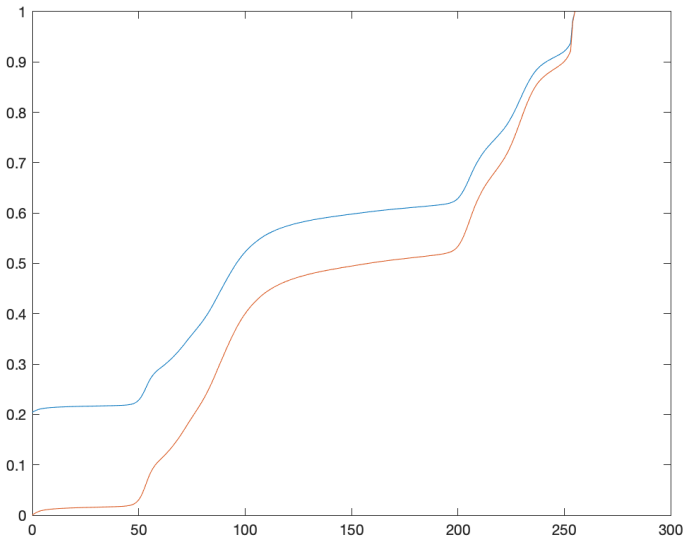
1) **Histogram Equalization**

a) The vector ”n” is the histogram of the image, and cumsum takes the cumulative sum. So ”map” is the CDF, computed from the histogram and then normalizing it so that it goes up to 1. The imshow command imshow(a, [map map map]) is using the CDF ”map” as the mapping function for the display, so this should look like ghe.

b) In the histogram, n(1) = 80655, so this bin represents a lot of background pixels. After setting newhist = n, we can set newhist(1) = 0 as a new ”fake” histogram to prevent these pixels from influencing the remapping function. We then compute the cdf for this modified histogram:

```
newmap = cumsum(newhist);
newmap = newmap / max(newmap);
imshow(a, [newmap newmap newmap])
```

and this has more contrast. We can plot both curves with
plot(y,map,y,newmap)



2) **Noise Cleaning**

a) For the image baby2 which has both noise types, the MSE after median filtering with the [1 3] filter is 6.7367, whereas the MSE is slightly better at 6.4710 with the [3 3] filter. On the other hand, for the babyS image which has only the Salt noise, the [1 3] filter comes out ahead, with MSE =4.4862, whereas the [3 3] filter achieves 5.5141.
Why does it turn out this way? Consider the case with Salt noise alone. The Salt noise comes as little vertical streaks, one pixel wide. So a horizontal 1x3 median filter is oriented correctly to clean this up, and it makes use only of two 4-neighbors. The 3x3 median filter will also

clean it up, but it involves more pixels in the operation, including the 8-neighbors, which are farther away than the 4 neighbors. Also, when the 3x3 filter is centered on a noise pixel, it has at least one other noise pixel in the window too (because they come as streaks). So when operating on the noisy pixels, the median operation is a little bit skewed, with at least 2 bad pixels. Most of the pixels of course are not noisy (in this image) and do not need to be altered at all. In those areas where nothing needs to be done, the 1x3 filter is smaller and will do less alteration than the 3x3. So the 1x3 filter wins for the Salt noise alone.

The baby2 image, on the other hand, has more noise, since it has the Gaussian noise as well. So here, all of the pixels are noisy and the larger size of the 3x3 filter is giving some benefit. For the pixels which have only the Gaussian noise (which is the vast majority of pixels), the 3x3 does not suffer from the skew issue and it works better than the 1x3 (although admittedly a median filter is not the best choice for cleaning this up). For the pixels with Salt noise, the 3x3 is still operating with some skew, but these are a relatively small set.

b) Sequential use of a median filter and a spatial averaging filter on the baby2 image that has both noise types.

By applying a [3 3] median filter followed by a [3 3] average filter, the MSE is reduced to 5.7028. This additional improvement likely comes mostly from the pixels which had Gaussian noise (because the median filter did not clean those well). However, applying the median filter first is important as the salt-and-pepper noise should be dealt with first. Using the averaging filter first would potentially blend the salt-and-pepper noise with the other pixels which would make the salt-and-pepper noise much harder to detect after that point.

```
>> tempoutput = medfilt2(baby2, [3 3]);
>> seqoutput = imfilter(tempoutput, ones(3)/9, 'replicate');
>> seqmse = immse(cleanbaby, seqoutput)
>> imshow(seqoutput)
```

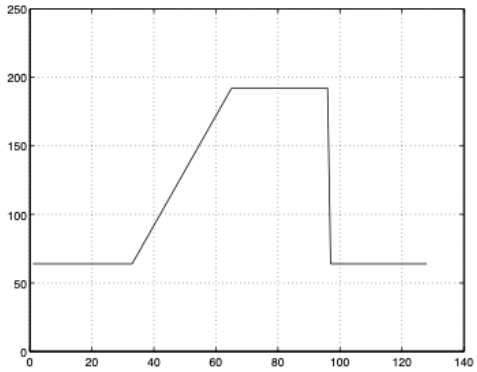
3) **Unsharp Masking**

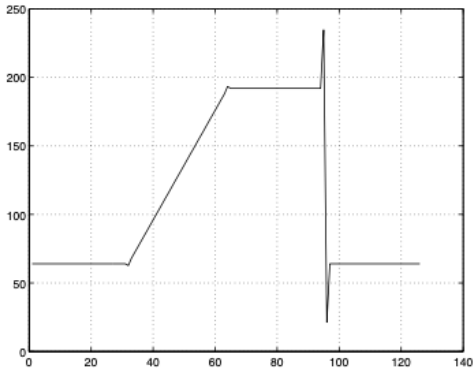
a) maskB is an identity filter that is the same size as maskA. It can also be viewed as an all-pass filter as it does not filter out certain frequency components. Therefore, filtering with maskB just gives you back the original image. MaskC is a highpass filter. It is not an unsharp mask. MaskD is either a highboost filter or an unsharp mask depending on the highboost weight. In this question, both labels are acceptable. MaskD is not a high-pass filter. Filtering with maskD results in an image which is the sum of the original image plus some weight times a highpass version of it. So this will have the result of boosting the highpass content, thereby sharpening the edges.

b) We try first a simple unweighted 3×3 averaging filter as our lowpass mask, with a fairly large extra weight (1) given to the highpass portion:

```
>> mask = 1/9 * ones(3);
>> tmp = unsharp(tst,mask,1);
>> plot(tmp(64,:))
```

Some people tried VERY large weights, like 5 or 10, in which case the 8-bit range of the image is exceeded, and you have to deal with the truncation and rescaling issues. But there is no need to use a weight that large. Even a weight of 1 produces an enormous edge enhancement effect, as can be seen from the cross section. After unsharp masking, the test image shows substantial undershoots and overshoots at the step edge, and there are very slight overshoots and undershoots at the beginning and end of the ramp. A cross-section through the test image is shown below left (plot(tst(64,:))), and the cross- section for the edge-sharpened version is shown below right:



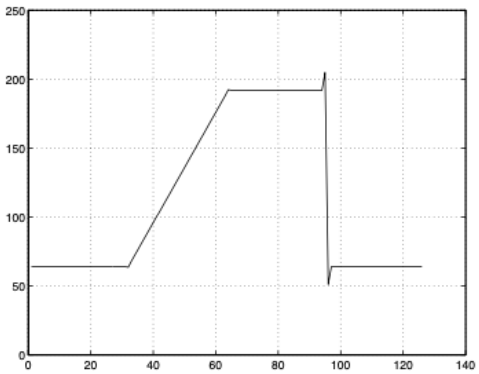


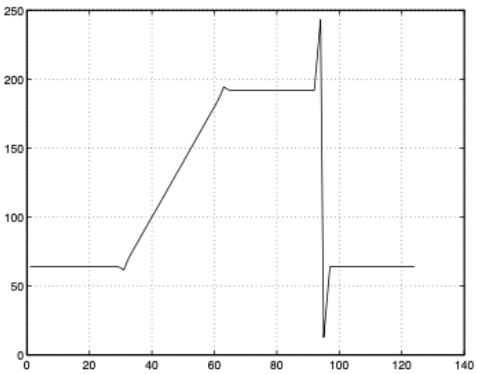
If the extra amount of highpass added in is not so much, the overshoots are less pronounced relative to the ones in the previous plot (graph shown below left):

```
>> tmp = unsharp(tst,mask,0.3);
>> plot(tmp(64,:))
```

If the lowpass filter is larger, then the overshoots get wider (graph shown below right):

```
>> mask = 1/25 * ones(5);
>> tmp = unsharp(tst,mask,1);
>> plot(tmp(64,:))
```





Lastly, if the lowpass filter is strongly center-weighted, then the lowpass filtered image will

be close to the original image. In this case, the highpass filtered image (which is the original minus the lowpass filtered) will be close to zero. So we expect that strong center-weighting will produce an unsharp masked image that doesn't look very different from the original.

```
>> mask = 1/18 * [1 1 1; 1 10 1; 1 1 1];
>> tmp = unsharp(tst,mask,1);
```

This is in fact the case; the overshoots are small. Basically, the various parameters one can play with are changing the heights and widths of the overshoots.

c) For a real-world example at larger size, read in the image "blurry-moon.tif" and use highboost filtering on it starting with some Gaussian lowpass filtering of your choice. You can do this easily using imgaussfilt. Try different boost values. Provide your code and your output image, and comment on your result.

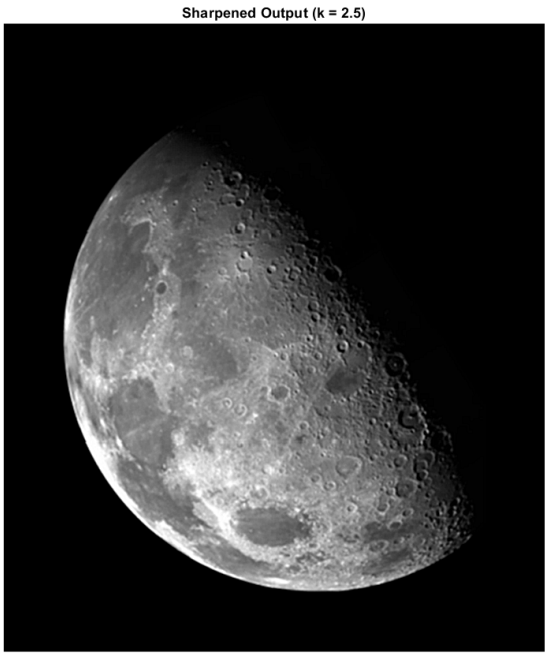
When we use the following Gaussian filter

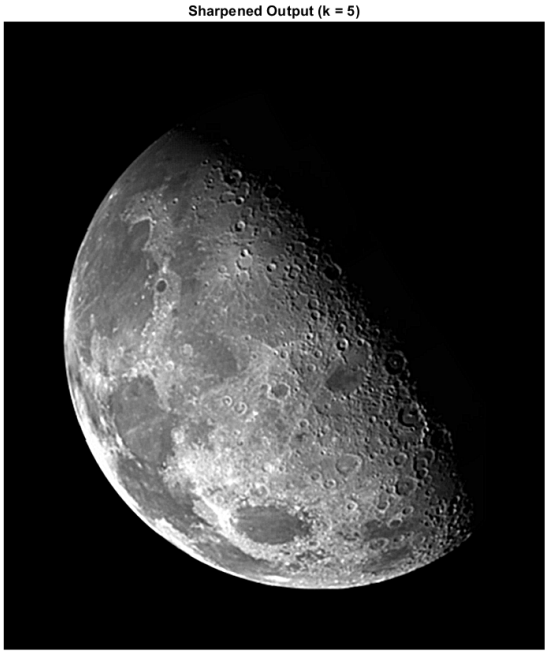
$$f = 1/159 \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

together with a fairly large weight like 1.5, we get the following output:

Although smaller weights do work, it is difficult to distinguish the sharpening effect in the output. Therefore, larger weights are more preferable for this particular sharpening task. Below

are the output images obtained when weights were set to be 2.5, 5, and 7.5, respectively.





The sharpening effect significantly increases as we increase the weight, however, the truncation and overshoot effects become more and more evident at the same time. Since this is undesirable, a weight around 5 seems to be good when sharpening this particular image.

```
>> in = imread('blurry_moon.tif');
>> gauss = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9,
12, 9, 4;2, 4, 5, 4, 2];
>> gauss = 1/159 .* gauss;
>> out = unsharp(double(in), gauss, 5);
>> figure
>> imshow(uint8(out))
>> title("Sharpened Output (k = 5)")
```

4) **Order of Operations (this is not a Matlab problem)**

a) A median filter puts elements in order and takes the middle one. Because the square root function is monotonic increasing, it will not change the order of those elements. Therefore doing the median filter right before the contrast stretching, or doing it right after, will make no difference. Of the 6 orders specified:

A: MF $\rightarrow$ SA $\rightarrow$ CS
B: CS $\rightarrow$ MF $\rightarrow$ SA
C: SA $\rightarrow$ CS $\rightarrow$ MF
D: MF $\rightarrow$ CS $\rightarrow$ SA
E: SA $\rightarrow$ MF $\rightarrow$ CS
F: CS $\rightarrow$ SA $\rightarrow$ MF

we can say that B is equivalent to D and also C is equivalent to E.

b) In general, when there is both impulse noise (salt-and-pepper noise) and Gaussian noise, and one is doing both median filtering and spatial averaging filtering, it is better to do the median filter first, because the spatial averaging filtering will blur out the outliers and then the median filter becomes ineffective at removing them, So MF should come before SA, and that principle by itself would suggest that of the 4 distinct systems (A, B, C, and F) we expect that A and B, which have MF before SA, will perform better than C and F, which put SA earlier.

A different consideration is that Gaussian noise has mean zero, and if some pixel has a noise value +N1, and some nearby pixel has a value -N1, then they will cancel each other out in the averaging operation. But if we take the square root of all pixel values first, then the noise on these two hypothetical pixels won't cancel each other out from the averaging. So there would be a general concept that doing SA before CS is a good idea.

Combining these two ideas, we expect that system A would be the best. But the first effect in this context is the more important one: that median filtering should come before spatial averaging filtering, to remove the impulse noise before averaging smears it.