Activity 4: Nonlinear Filters

Primary Goal: Learn how to denoise an image using different filters.
Secondary Goal: Learn how to compare the performance of algorithms.

1.) Gaussian Noise

Load the "cameraman" image and add Gaussian noise with variance 0.02 using the `imnoise` command.

```
A_ideal = imread('cameraman.tif');
A_gauss = imnoise(A_ideal, 'gaussian', 0, 0.02);
```

Let's compare the performance of the mean and median filters on removing this noise.

First let's run a 5x5 Mean Filter and store our first denoising attempt in image A1.

```
w = fspecial('average', [5,5]);
A_mean = imfilter(A_gauss, w, 'replicate');
```

The 'replicate' parameter will impose Neumann boundary conditions, which should result in slightly better performance at the image boundary.

Second let's run a 5x5 Median Filter and store this second denoising attempt in image A2. The Matlab command `medfilt2` performs a median filter with the specified neighborhood size.

```
A_median = medfilt2( A_gauss, [5,5] );
```

Now create an appropriate subplot to display the 3 images side-by-side:

- The noisy image A_gauss
- The mean filtered image A_mean
- The median filtered image A_median

As you can see from this problem, the hardest part is keeping track of all the different images. It definitely helps if you give the images clear names you can remember instead of just A, B, C, D. It involves a little more typing, but it can save a lot of frustration down the road.

2.) Image Statistics

Next let's compare the performance of the filters numerically. The Root Mean Square Error (RMSE) compares the denoising result to an ideal noise-free image:

$$RMSE = \frac{1}{N}\|A_{denoised} - A_{ideal}\|_F$$

where N is the number of pixels in the image and F denotes the Frobenius norm. To compute RMSE for our mean filter, we compare the mean filtered image A_mean to the original cameraman image A_ideal before we added noise. Note we have to cast the images to double format within the computation.

```
[m,n] = size(A_ideal);
RMSE = 1/(m*n)*norm( double(A_mean)-double(A_ideal),'fro');
```

Similarly we could compute the Signal-to-Noise Ratio (SNR):

$$SNR = 20\log\frac{\|A_{ideal}\|_F}{\|A_{ideal} - A_{denoised}\|_F}$$

Compute the RMSE and SNR for both your mean and median filtered images from #1 and write the values in the table below. Do the numbers comfirm your visual inspection?

	RMSE	**SNR**
5x5 Mean Filter	0.0775	38.0004
5x5 Median Filter	0.0776	37.8726

Note SNR decreases as the image gets more noisy.

Gaussian Noise σ=0.01

SNR=34.0206

Gaussian Noise σ=0.05

SNR=19.1825

Gaussian Noise σ=0.1

SNR=13.7121

3.) Salt & Pepper Noise

The median filter really shines on salt & pepper noise. Create a new noisy image using the command:

```
A_salt = imnoise (A_ideal, 'salt & pepper', 0.1);
```

This will reset the values of 10% of the pixels to random salt & pepper values.

Run the 5x5 mean and median filters on this new image. Note that you can use the up arrow in Matlab to quickly look up previous commands and change A_gauss to A_salt. If you remember how a command started, you could type in a few letters and then press the up arrow.

Display the results side-by-side in an appropriate subplot. Then compute the RMSE and SNR for the denoised salt & pepper images.

	RMSE	**SNR**
5x5 Mean Filter	0.0829	36.3701
5x5 Median Filter	0.0676	40.7499

Note SNR decreases as the image gets more noisy.

5% Salt & Pepper Noise

SNR=29.4225

10% Salt & Pepper Noise

SNR=21.7686

20% Salt & Pepper Noise

SNR=14.6724

4.) Algorithm Runtimes

You should have noticed that the median filter is generally slower than the mean filter. We can time processes in Matlab using the `tic` and `toc` commands. The command `tic` starts a stopwatch running. The command `toc` records the time (in seconds) that has elapsed since the last `tic` was entered.

To time a process, we could type on a single line:

```
tic;  MY ALGORITHM; toc
```

Note we do not put a semi-colon on the final `toc` command, since we want to see a display of how long it took to the run the algorithm.

For example, to time the mean filter in #1 we would type something like:

```
tic;  A_mean = imfilter(A_gauss, w, 'replicate');  toc
```

Now use a similar tic-toc to time the median filter. Write the runtimes in the middle column of the table at the bottom of the page.

Since the original cameraman image was only 256x256, the mean filter runs almost instantaneously on this image. It would be interesting to try timing our algorithms on a larger image. The Matlab command `imresize` can change the size of an image to anything we like. The basic format is:

```
NEW = imresize (IMAGE, NEW SIZE or MAGNIFICATION FACTOR);
```

In our cameraman image, we want to resize the original image before we add noise. If we resized after adding noise, it would only make the noise points into bigger noise blobs and we would get a mess.

Let's make the image 2 times larger to create a 512x512 cameraman image. Note resizing by a factor 2 in both the image width and height creates 4 times as many pixels. The two commands below do the exact same thing. I just wanted to show you the two ways that you can call the `imresize` command.

```
A_large = imresize(A_ideal, 2);
A_large = imrezize(A_ideal, [512,512]);
```

Now add Gaussian noise to the large image and time the mean filter again.

```
A_gauss = imnoise (A_large, 'gaussian', 0, 0.2);
tic;  A_mean = imfilter(A_gauss, w, 'replicate');  toc
```

Run the 5x5 median filter on this large image and time it. Write the runtimes in the table below. Can you see how the growth rates of the runtimes differ?

	Runtime (sec)	
	256x256 Image	**512x512 Image**
5x5 Mean Filter	0.000701	0.001794
5x5 Median Filter	0.001485	0.002261