BIOS 7718 Assignment 1

Piper Williams

March 8, 2019

Problem 1, Part 1

Figure 1 from the Assignment 1 description shows two different methods for fitting a straight line by minimizing errors. In Figure 1a, the line was fit by minimizing the sum of squared vertical distances between data points and the line. In Figure 1b, the line was fit by minimizing the sum of squared perpendicular distances between data points and the line.

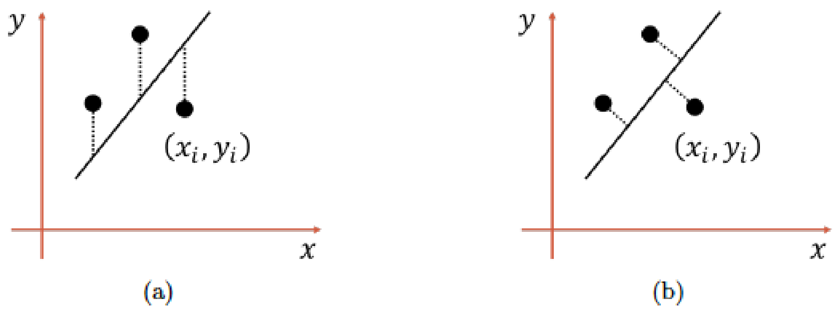


Fig. 1. **Illustration of distances. (a) Vertical (i.e., *y*-direction) distances between data points and a line. (b) Perpendicular distances between data points and a line.**

Minimizing the sum of squared vertical distances between the data points and the line $y = ax + b$ can be written as:

$$\sum_{i=1}^{n} (y_i - ax_i - b)^2$$

In James R Carr's *Orthogonal regression: a teaching perspective*, Carr includes the following figure to display the four ways we can fit a straight line by minimizing errors.

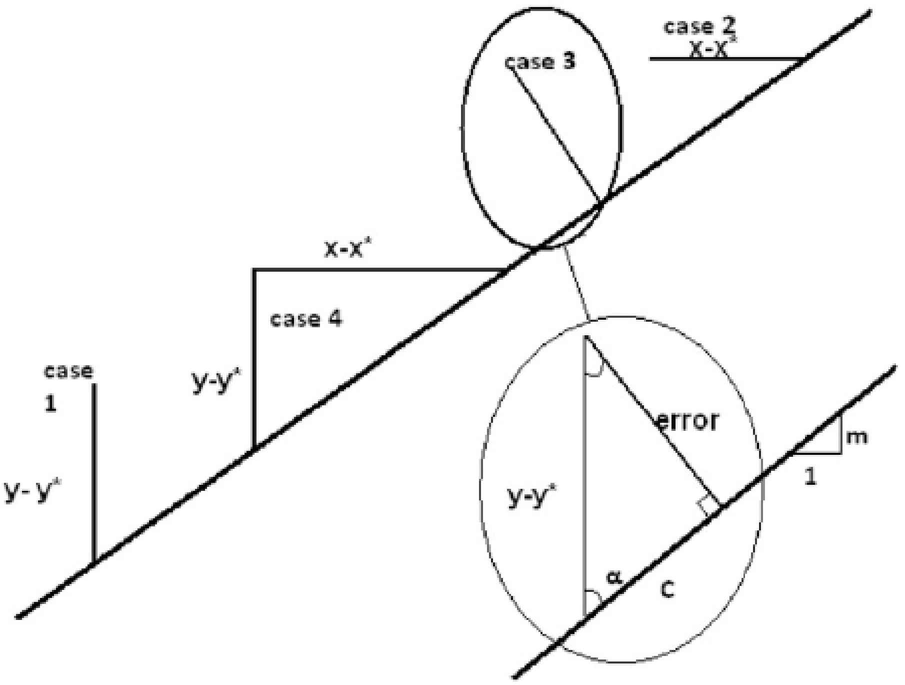


Fig. 2: Four approaches to linear regression. Case 3 shows the approach in which the sum of squared perpendicular distances between the data points and the line are minimized. This yields what is known as the major axis line. Here, Carr uses the variable m to describe the slope. However, for our purposes, we will use the term a to describe the slope

In comparing Figure 1a with that of case 3 in Figure 2, these two approaches are the same, and this is the approach used in orthogonal regression modeling. The equation for the line that follows the Case 3 approach can then be written as follows:

$$y^* = ax + b$$

...where y^* represents the estimated, or predicted, value of y given x. In the orthogonal regression approach, one of the key assumptions is that both x and y are subject to measurement error. Thus, the goal of orthogonal regression is to solve for both a and b such that the *total* sum of squared error, e^2, is minimized. From Figure 2, and from our understanding of the Pythagorean theorem, we can solve for e^2 as follows:

$$\begin{aligned}
e^2 &= (y_i - y_i^*)^2 - C^2 \\
&= (y_i - y_i^*)^2 - [(y_i - y_i^*)^2 - cos^2(\alpha)] \\
&= (y_i - y_i^*)^2 - [(y_i - y_i^*)^2 - (\frac{a}{\sqrt{a^2+1}})^2] \\
&= \frac{a^2}{a^2+1}
\end{aligned}$$

Again, as a reminder, Carr used the term m to describe the slope. Here, we are using a to describe the slope. It can then be seen that the squared error, e^2, can be simplified to:

$$e^2 = \frac{(y_i - y_i^*)^2}{a^2+1}$$

Thus, total squared error (TSE) is equal to:

$$\begin{aligned} TSE &= \sum_{i=1}^{n} \frac{(y_i - y_i^*)^2}{a^2+1} \\ &= \sum_{i=1}^{n} \frac{(y_i - ax_i - b)^2}{a^2+1} \end{aligned}$$

Now, let's differentiate TSE with respect to the intercept, b, and set that equal to zero. This will give us the solution for b that will minimize TSE.

$$\begin{aligned} \frac{d}{db}TSE &= \frac{d}{db}\sum_{i=1}^{n} \frac{(y_i - ax_i - b)^2}{a^2+1} \\ &= \frac{d}{db}\sum_{i=1}^{n} \frac{(y_i^2 + a^2x_i^2 - 2ax_iy_i + b^2 - 2by_i + 2ax_ib)}{a^2+1} \\ &= \frac{d}{db}\sum_{i=1}^{n}\frac{y_i^2}{a^2+1} + \frac{d}{db}\sum_{i=1}^{n}\frac{a^2x_i^2}{a^2+1} - \frac{d}{db}\sum_{i=1}^{n}\frac{2ax_iy_i}{a^2+1} + \frac{d}{db}\sum_{i=1}^{n}\frac{b^2}{a^2+1} - \frac{d}{db}\sum_{i=1}^{n}\frac{2by_i}{a^2+1} + \frac{d}{db}\sum_{i=1}^{n}\frac{2ax_ib}{a^2+1} \\ &= \sum_{i=1}^{n}\frac{d}{db}(\frac{y_i^2}{a^2+1}) + \sum_{i=1}^{n}\frac{d}{db}(\frac{a^2x_i^2}{a^2+1}) - \sum_{i=1}^{n}\frac{d}{db}(\frac{2ax_iy_i}{a^2+1}) + \sum_{i=1}^{n}\frac{d}{db}(\frac{b^2}{a^2+1}) - \sum_{i=1}^{n}\frac{d}{db}(\frac{2by_i}{a^2+1}) + \sum_{i=1}^{n}\frac{d}{db}(\frac{2ax_ib}{a^2+1}) \\ &= 0 + 0 - 0 + \sum_{i=1}^{n}(\frac{2b}{a^2+1}) - \sum_{i=1}^{n}(\frac{2y_i}{a^2+1}) + \sum_{i=1}^{n}(\frac{2ax_i}{a^2+1}) \\ &= \frac{2}{a^2+1}(nb - \sum_{i=1}^{n} y_i + a\sum_{i=1}^{n} x_i) = 0 \end{aligned}$$

$$nb - \sum_{i=1}^{n} y_i + a\sum_{i=1}^{n} x_i = 0$$

$$\begin{aligned} nb &= \sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i \\ b &= \sum_{i=1}^{n}\frac{y_i}{n} - a\sum_{i=1}^{n}\frac{x_i}{n} \\ b &= \overline{y} - a\overline{x} \end{aligned}$$

Therefore, the final solution is $b = \overline{y} - a\overline{x}$. By substituting this into our original TSE equation, we get:

$$\begin{aligned} TSE &= \sum_{i=1}^{n}\frac{[y_i - ax_i - (\overline{y} - a\overline{x})]^2}{a^2+1} \\ &= \frac{1}{a^2+1}\sum_{i=1}^{n}[(y_i - \overline{y}) - a(x_i - \overline{x})]^2 \end{aligned}$$

Now, let's set $V_i = y_i - \overline{y}$ and $U_i = x_i - \overline{x}$. We can then rewrite the TSE equation as such:

$$\begin{aligned} TSE &= \frac{1}{a^2+1}\sum_{i=1}^{n}(V_i - aU_i)^2 \\ &= \frac{1}{a^2+1}\sum_{i=1}^{n} V_i^2 - 2aU_iV_i + a^2U_i^2 \end{aligned}$$

From here, we can differentiate the TSE equation with respect to the slope, a, and set this equal to zero to solve for the solution of a that will minimize TSE.

$$
\begin{aligned}
\frac{d}{da}TSE &= \frac{d}{da}(\frac{1}{a^2+1}\sum_{i=1}^{n} V_i^2 - 2aU_iV_i + a^2U_i^2) \\
&= \frac{d}{da}\sum_{i=1}^{n}\frac{V_i^2}{a^2+1} - \frac{d}{da}\sum_{i=1}^{n}\frac{2aU_iV_i}{a^2+1} + \frac{d}{da}\sum_{i=1}^{n}\frac{a^2U_i^2}{a^2+1} \\
&= \sum_{i=1}^{n}\frac{d}{da}(\frac{V_i^2}{a^2+1}) - \sum_{i=1}^{n}\frac{d}{da}(\frac{2aU_iV_i}{a^2+1}) + \sum_{i=1}^{n}\frac{d}{da}(\frac{a^2U_i^2}{a^2+1}) \\
&= \sum_{i=1}^{n}\frac{-2aV_i^2}{(a^2+1)^2} - \sum_{i=1}^{n}\frac{[(a^2+1)(2U_iV_i)] - [(2aU_iV_i)(2a)]}{(a^2+1)^2} \\
&+ \sum_{i=1}^{n}\frac{[(a^2+1)(2aU_i^2)] - [(a^2U_i^2)(2a)]}{(a^2+1)^2} \\
&= \frac{1}{a^2+1}\sum_{i=1}^{n} -2aV_i^2 - 2a^2U_iV_i - 2U_iV_i + 4a^2U_iV_i + 2a^3U_i^2 + 2aU_i^2 - 2a^3U_i^2 \\
&= \frac{1}{a^2+1}\sum_{i=1}^{n} -2aV_i^2 + 2a^2U_iV_i - 2U_iV_i + 2aU_i^2 \\
&= \frac{2}{a^2+1}\sum_{i=1}^{n} -aV_i^2 + a^2U_iV_i - U_iV_i + aU_i^2 = 0
\end{aligned}
$$

$$\sum_{i=1}^{n} -aV_i^2 + a^2U_iV_i - U_iV_i + aU_i^2 = 0$$

$$a^2\sum_{i=1}^{n} U_iV_i - a(\sum_{i=1}^{n} V_i^2 - \sum_{i=1}^{n} U_i^2) - \sum_{i=1}^{n} U_iV_i = 0$$

From here, we can use the quadratic equation to solve for a. As we remember, the general form of the first root of the quadratic equation is equal to:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Thus...

$$
\begin{aligned}
a &= \frac{(\sum_{i=1}^{n} V_i^2 - \sum_{i=1}^{n} U_i^2) + \sqrt{(\sum_{i=1}^{n} V_i^2 - \sum_{i=1}^{n} U_i^2)^2 - 4(\sum_{i=1}^{n} U_iV_i)^2}}{2\sum_{i=1}^{n} U_iV_i} \\
&= \frac{\sum_{i=1}^{n}(y_i - \overline{y})^2 - \sum_{i=1}^{n}(x_i - \overline{x})^2 + \sqrt{[\sum_{i=1}^{n}(y_i - \overline{y})^2 - \sum_{i=1}^{n}(x_i - \overline{x})^2]^2 - 4(\sum_{i=1}^{n}[(x_i - \overline{x})(y_i - \overline{y})]^2)}}{2\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}
\end{aligned}
$$

We have now shown the two solutions for the intercept b and the slope a that results in the minimized TSE. Moving forward, we now would like to compare two minimization approaches described earlier (vertical versus perpendicular error) when there is error added to both x and y variables. For this problem, we carried out 100 simulations (n = 30 per simulation) in which we added normally distributed error terms to both x and y variables. The data points for each simulation were generated using the following equation: $y^o = 6x^o + 1$, where $x_i^0 = 0, 1, ..., 29$. Then, v_i^o was defined as $v_i^o = (x_i^o, y_i^o)\epsilon\mathbf{R}^2$ and v_i was defined as $v_i = (x_i, y_i)$. Thus, v_i^o was defined as the clean data points, and v_i was defined as the noisy data points. v_i was, thus, equal to:

$$v_i = v_i^o + \delta v_i, \delta v_i \sim N(0, 5^2 I), i = 1, 2, ..., n$$

The data points $\{v_i\}_{i=1}^{n}$ were then rounded to the nearest integer. For each simulation, we were then asked to save the parameter estimates for a and b for each estimation procedure and plot these parameter estimates in two separate scatter plots (Figure 3).

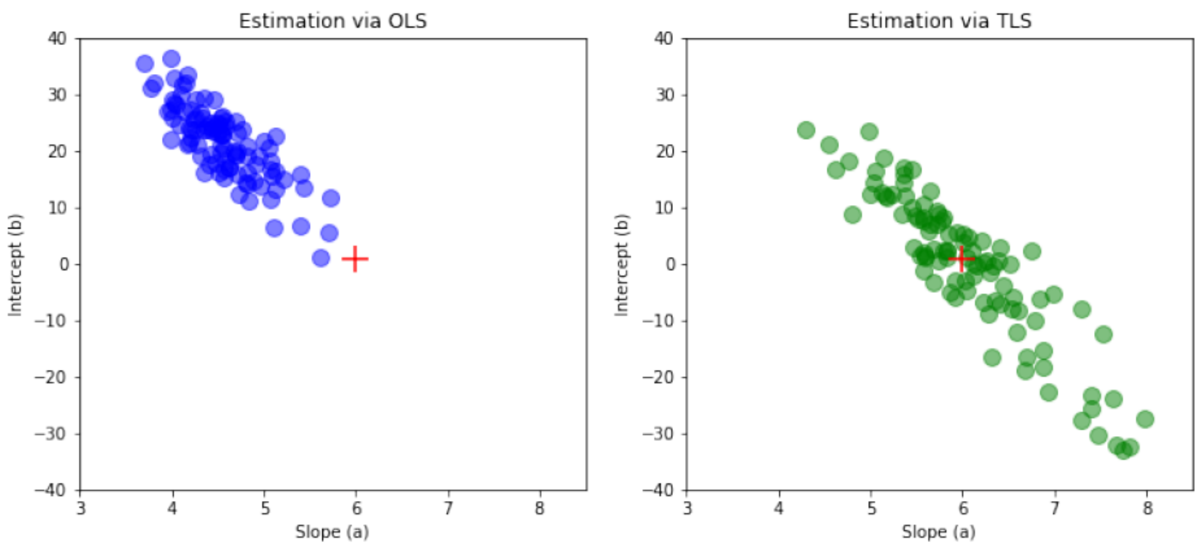


Fig. 3: The scatter plot on the left shows the estimated slope and intercept parameters for the ordinary least squares approach, which minimizes the sum of the squared vertical distance between the data points and the line, for each simulation. The scatter plot on the right shows the estimated slope and intercept parameters for the total least squares approach, which minimizes the sum of the squared perpendicular distance between the data points and the line, for each simulation. The red crosses indicate the true a and b. OLS = ordinary least squares, TLS = total least squares.

In comparing the slope and intercept estimates for both estimation approaches, it seems that 1.) the OLS estimation method consistently underestimates a and overestimates b. Logically, this makes sense. In adding noise to the original data, the slope via the OLS estimation becomes more shallow and the intercept then becomes larger than the true intercept to further minimize the vertical error. In comparison, the slope and intercept estimates via the TLS estimation method seem to be more evenly spread around the true slope and intercept value. We then were asked to take the mean of the 100 slope and intercept values for each estimation method and plot this line to then compare with the true fit line, which follows the equation $y^o = 6x^o + 1$ (Figure 4).

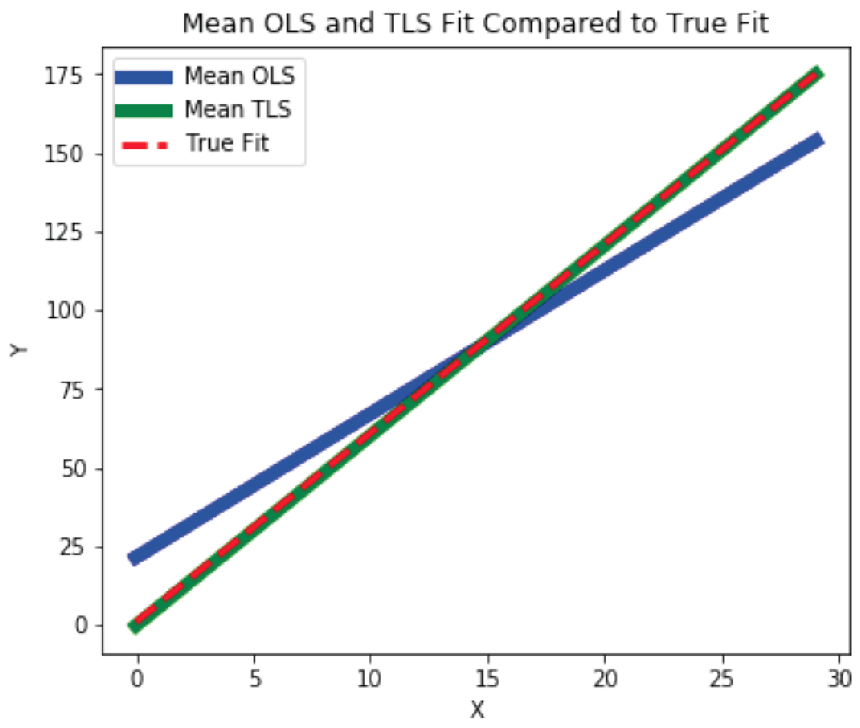


Fig. 4: Comparison of the mean OLS, mean TLS, and true fit. The true fit was equal to $y^o = 6x^o + 1$.

In Figure 4, it is obvious that the estimation method in which the perpendicular error was minimized (TLS) is a much better approximation of the true fit line compared to the estimation method in which the vertical error was minimized (OLS). In fact, the mean TLS line is almost identical to that of the true fit line.

Problem 1, Part 2

For this next problem, we were asked to repeat the same data generation procedure as was done in Problem 1, Part 1. We were then asked to generate another data point by sampling from a multivariate normal distribution with $\mu = (50, 2)$ and $\Sigma = 5^2 I$. Thus, there would be 31 data points per simulation (with 100 simulations). We then plotted the lines of the mean parameter estimates for the OLS and TLS estimation approaches, along with the true fit line (Figure 5.)

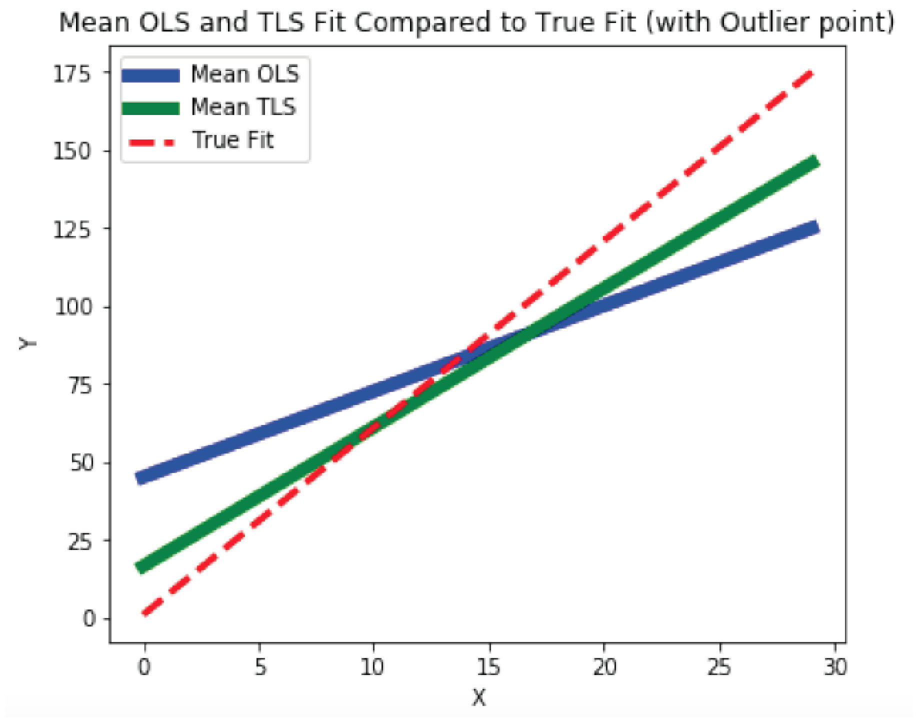


Fig. 5: Comparison of the mean OLS, mean TLS, and true fit with outlier data point. The true fit was equal to $y^o = 6x^o + 1$.

Here, we can see that the OLS estimation procedure is more sensitive to the outlier data point in comparison to the TLS estimation procedure. In comparison to the mean OLS and TLS lines fit in Problem 1, Part 1, the OLS and TLS lines from this problem deviate more from the line of true fit. Again, this is due to the fact that we added a positive outlier data point. However, it's worth noting again that the TLS estimation procedure seems to be more robust in maintaining a closer approximation to the line of true fit compared to the OLS estimation procedure. This seems to be the case in both Part 1 and Part 2 of Problem 1.

Problem 2, Part 1

For this problem, we were asked to apply a Gaussian filter to the image with different standard deviation values (σ = 1, 3, 5, 7, and 9). For each σ value, were were then asked to create a kernel size of $(6\sigma + 1) \times (6\sigma + 1)$. The original image, along with the five filtered images, are shown below (Figure 6).

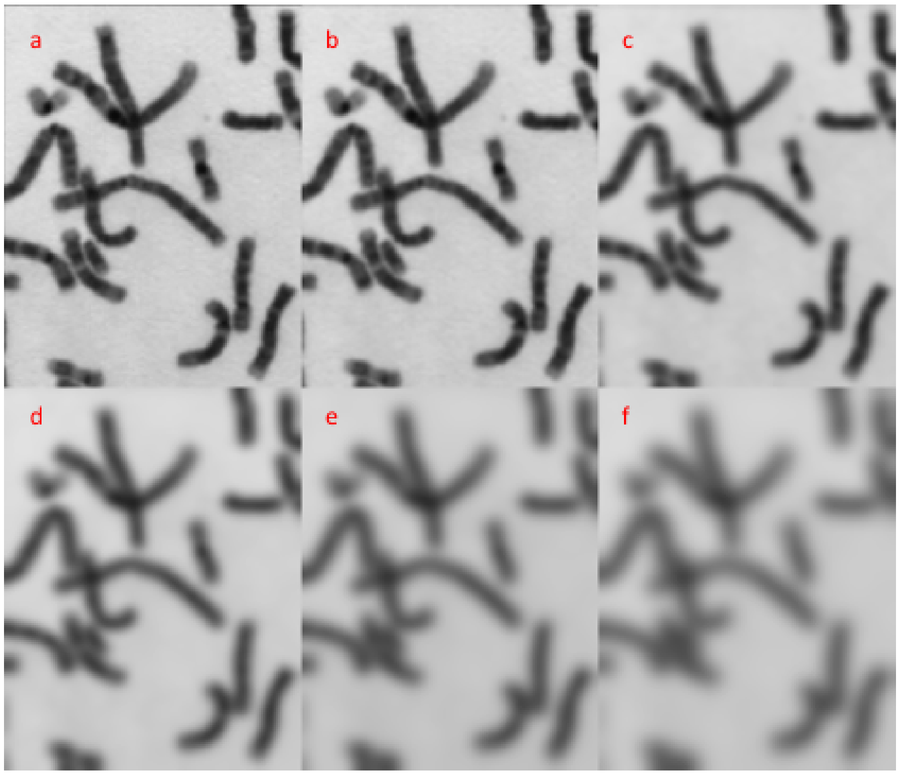


Fig. 6: Gaussian-filtered images with $(6\sigma + 1) \times (6\sigma + 1)$ kernel sizes. a.) original image, b.) $\sigma = 1$, c.) $\sigma = 3$, d.) $\sigma = 5$, e.) $\sigma = 7$, f.) $\sigma = 9$.

We can see that, as the standard deviation increases, the image itself becomes more blurry. This is in line with what we would expect. Based on our knowledge about the Gaussian distribution, as the standard deviation increases, the spread of the distribution about the mean increases. We also know that the kernel used in Gaussian filtering calculates a weighted average of the surrounding pixels following a Gaussian distribution. The increase in standard deviation means that the pixels surrounding the center pixel within a kernel will be weighted more heavily in comparison to a kernel with a smaller standard deviation. Also, increasing the kernel size with the standard deviation means that the kernel area is large and, thus, more pixels will be averaged over. Ultimately, increasing the standard deviation and the kernel dimensions results in a de-emphasizing of sharp gradient changes, which then results in a blurrier/smoother image.

We were then asked to apply a median filter to the image with different kernel sizes $w \times w$ ($w = 3$, 5, 7, 9, and 11) (Figure 7).

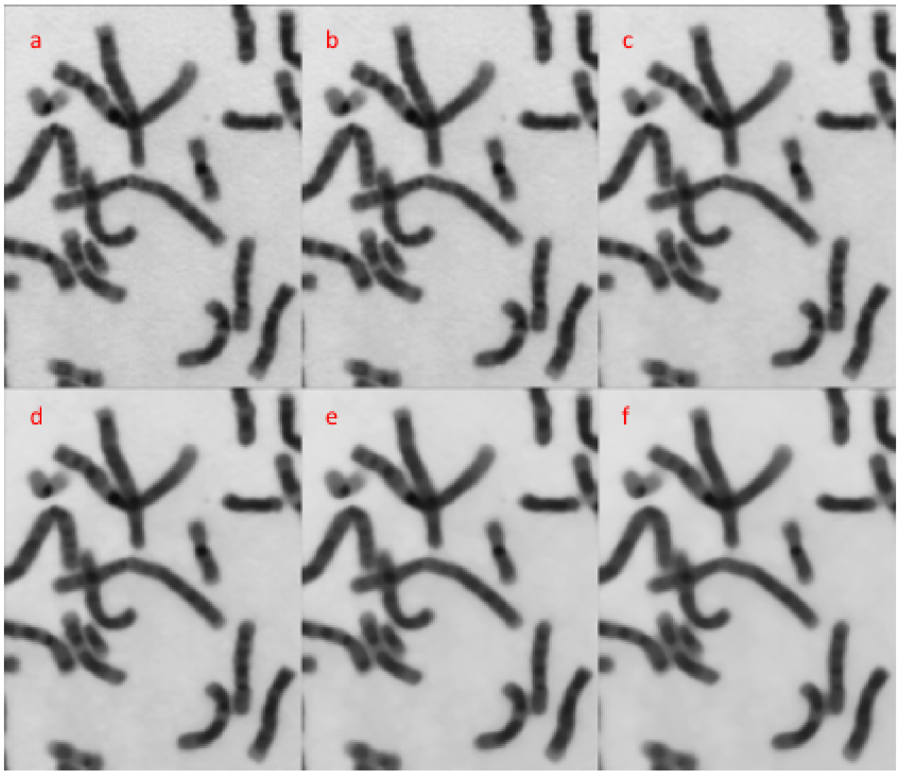


Fig. 7: Median-filtered images with $w \times w$ kernel sizes. a.) original image, b.) $w = 3$, c.) $w = 5$, d.) $w = 7$, e.) $w = 9$, f.) $w = 11$.

As the kernel size increases, there appears to be some smoothing of image details. However, the sharp edges seem to be retained. In comparison to the Gaussian-filtered images in Figure 6, the median-filtered images appear to be less blurry compared to those image. We know that the median filter is a non-linear filter, unlike the Gaussian filter. In median filtering, the pixel values are replaced with median value within the local neighborhood/kernel. The median filter is known to be an edge-preserving filter, meaning that sharp edges are retained. Based on this information, it makes sense that the median-filtered images are clearer/less blurry compared to the Gaussian-filtered images.

Problem 2, Part 2

Similarly to Problem 2, Part 1, we were asked to apply a Gaussian filter to the image with the same standard deviation values. However, the kernel sizes were set to be $(12\sigma + 1) \times (12\sigma + 1)$ instead of $(6\sigma + 1) \times (6\sigma + 1)$ (Figure 8).

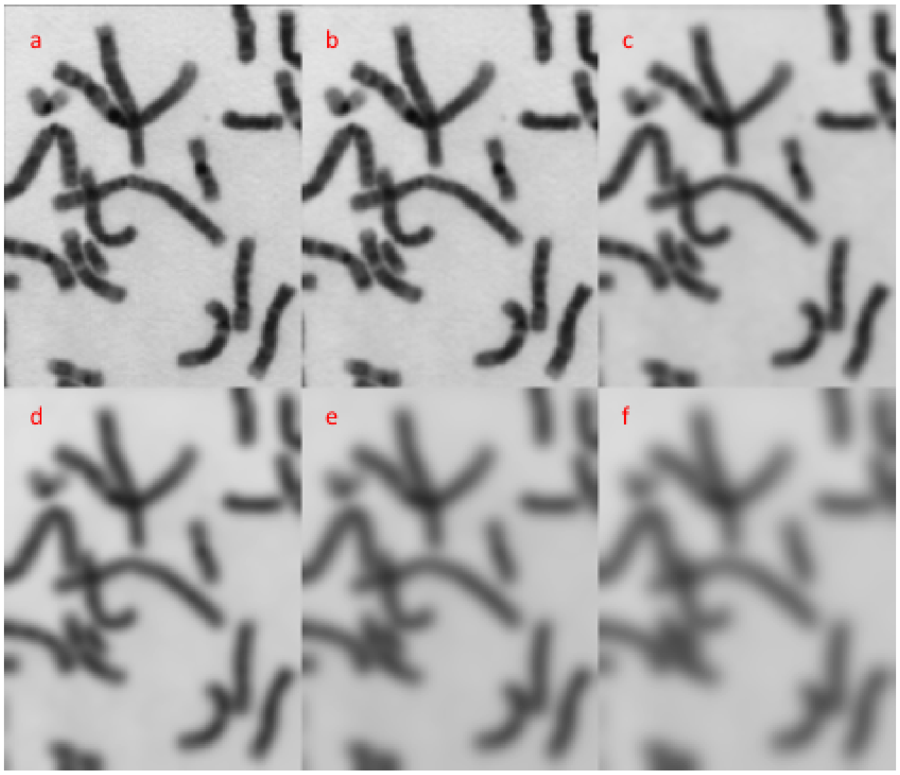


Fig. 8: Gaussian-filtered images with $(12\sigma + 1) \times (12\sigma + 1)$ kernel sizes. a.) original image, b.) $\sigma = 1$, c.) $\sigma = 3$, d.) $\sigma = 5$, e.) $\sigma = 7$, f.) $\sigma = 9$.

In comparing these images to those from Figure 6, there does not appear to be any obvious visual differences between the images. To further investigate, the absolute value of the difference of the maximum intensity pixel values for images with a matching standard deviation were calculated (Table 1).

σ	| Difference |
1	0
3	4
5	0
7	0
9	0

Table 1: Absolute value of the difference of maximum pixel intensity values for Gaussian-filtered image with kernel sizes of $(6\sigma + 1) \times (6\sigma + 1)$ and $(12\sigma + 1) \times (12\sigma + 1)$.

Based on this table, four of the five pairs of images have an absolute difference of zero between maximum pixel intensity values. The only pair of images that had a difference were the $\sigma = 3$ images, and this difference was minimal. In comparing the Gaussian filtering method in this problem with that of Problem 2, Part 1, the only difference between the two filtering methods were the difference in kernel sizes. However, the standard deviations remained the same. In Gaussian filtering, we know that the outer edges of the kernel have smaller (potentially zero) weighting coefficients compared to the central components of the kernel. Thus, unless the standard deviation increases with the increasing kernel size, the weighted average of pixel values may not change. This would result in pairs of images that are very similar, or identical, as we've seen in this case here.

Problem 3, Part 1

Here, we applied a first derivative of Gaussian filter to an image with different standard deviation values ($\sigma = 1$, 3, 5, and 7) and kernel sizes (kernel dimensions equal $(6\sigma + 1) \times (6\sigma + 1)$). I first filtered the image with a Gaussian smoothing function, where I defined the varying standard deviation and kernel sizes. I then applied a Sobel operator to the Gaussian-filtered images, with a 3×3 kernel. The Sobel kernel can be thought of as a 3×3 approximation to the first-derivative-of-Gaussian kernel. From the x and y-derivatives generated from the Sobel operator, we then calculated the gradient magnitude for each image using the following equation: $\sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$. Finally, the edge images were generated by thresholding the gradient magnitude images (threshold was calculated as 20% of the maximum gradient magnitude for each image). Below are the x-derivative, y-derivative, gradient magnitude, and edge images for each standard deviation value (Figures 9-12, respectively).

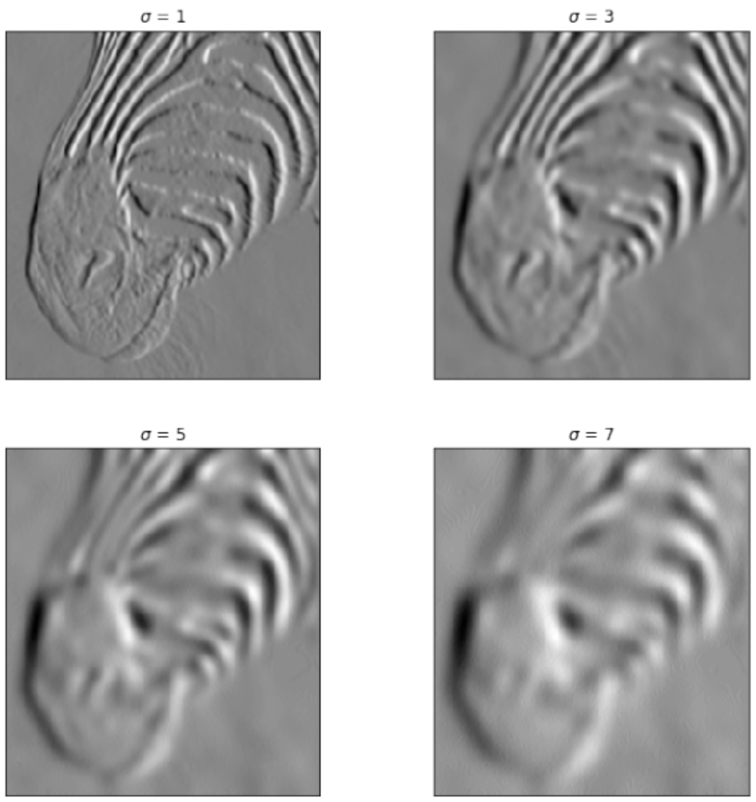


Fig. 9: X-direction derivative of Gaussian-filtered images for varying levels of σ. Kernel dimensions were $(6\sigma + 1) \times (6\sigma + 1)$.

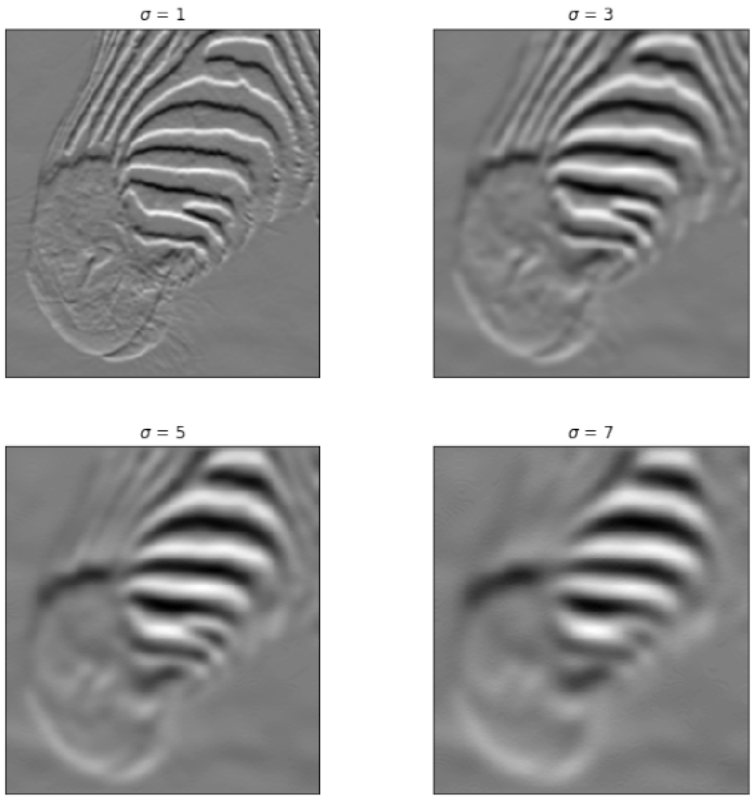


Fig. 10: Y-direction derivative of Gaussian-filtered images for varying levels of σ. Kernel dimensions were $(6\sigma+1)\times(6\sigma+1)$.

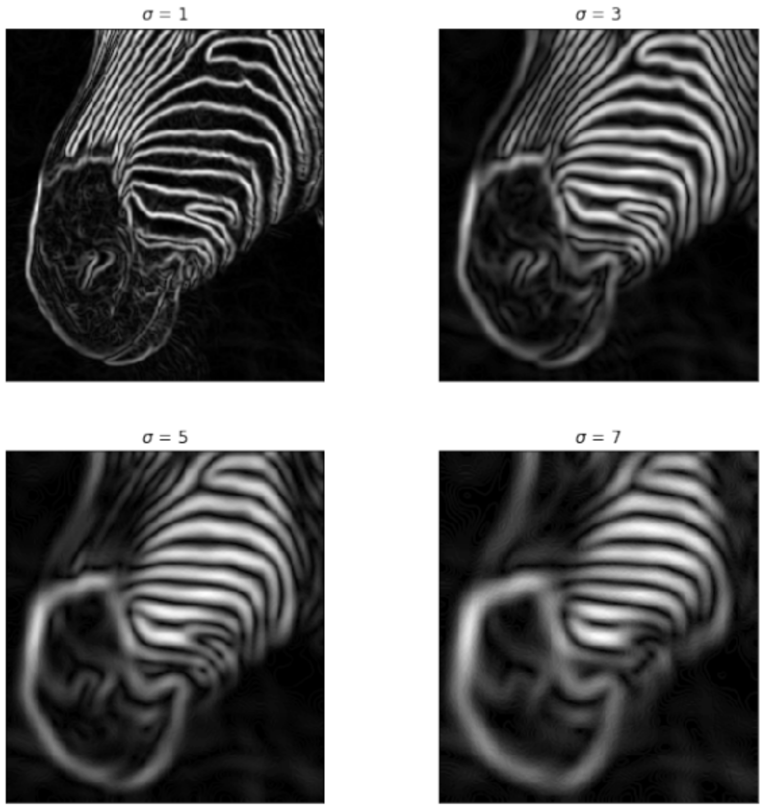


Fig. 11: Gradient magnitude images for varying levels of σ. Gradient magnitude $= \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$.

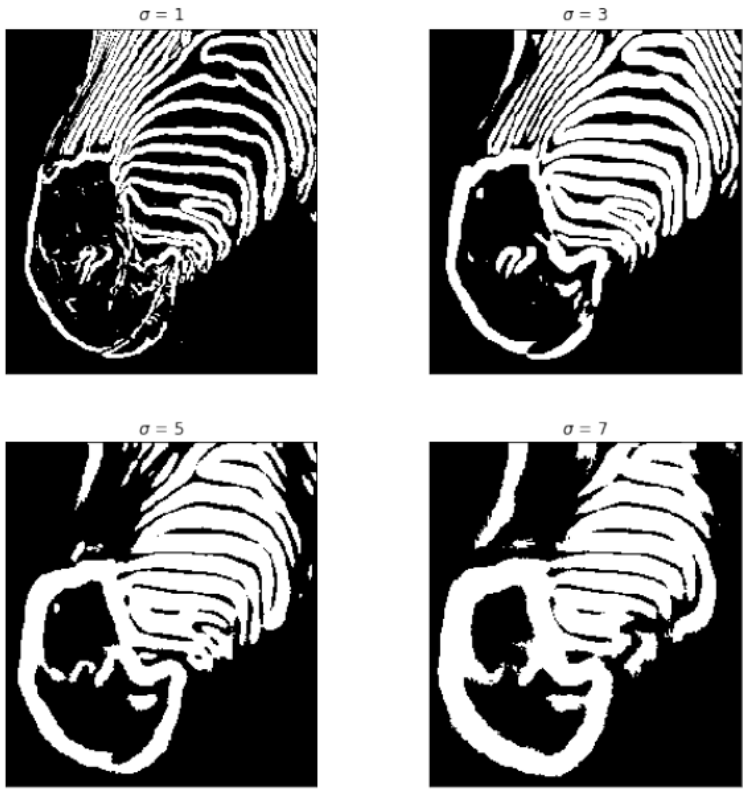


Fig. 12: Edge images for varying levels of σ. The threshold was calculated as 20% of the maximum gradient magnitude value for that image.

In Figures 9-11, we can see that increasing the standard deviation (and, thus, the corresponding kernel size) results in more blurring of the image. Ultimately, we can then see that increasing the standard deviation results in poorer edge detection performance, as seen in Figure 12. We lose more detail in the edge detection process. The edges also become thicker as the standard deviation increases. This is likely due to the fact that, at higher standard deviations and kernel sizes, there are more pixels that are being averaged over in the calculation of the weighted average from the Gaussian filter. This would result in a blurrier image and poorer edge detection capabilities.

Problem 3, Part 2

We were then asked to use a Canny edge detector with varying standard deviation values (σ = 1, 3, 5, and 7) (Figure 13). The process of Canny edge detection is as follows: 1.) smooth the image with the Gaussian filter, 2.) compute the gradient magnitude and angle images, 3.) apply non-maximum suppression to the gradient magnitude images, 4.) use double thresholding to detect strong and weak edges, and 5.) reject weak edges that are not connected with strong edges.

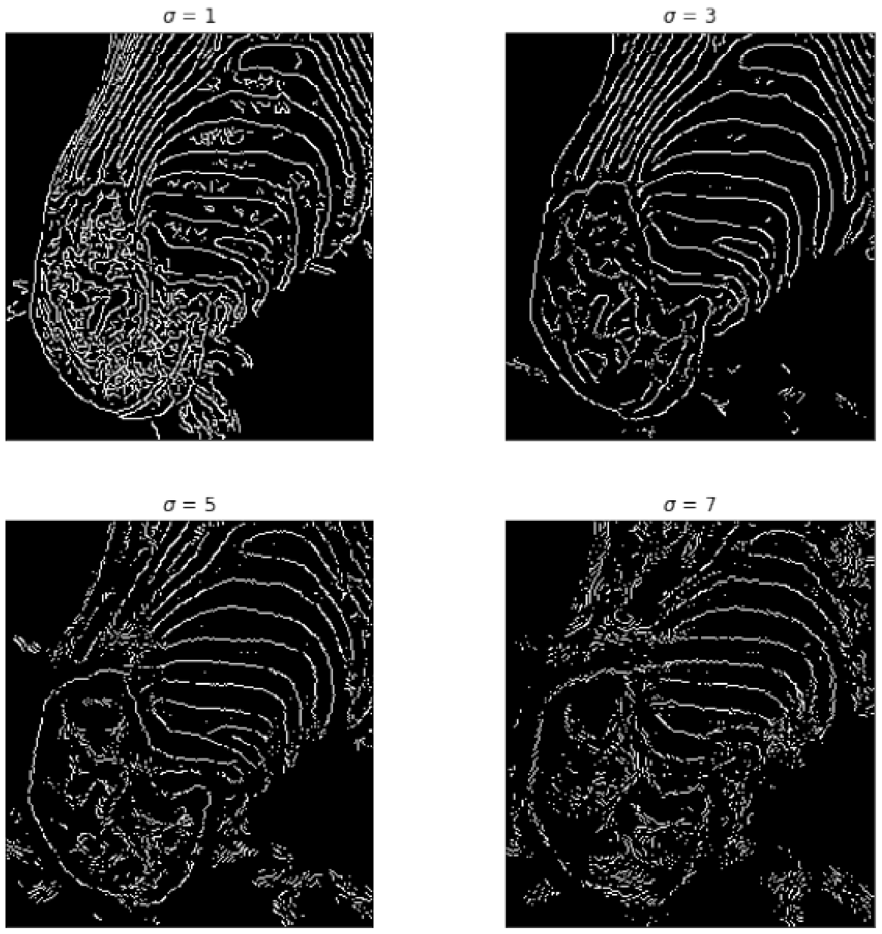


Fig. 13: Canny edge detected images for varying levels of σ. The upper and lower thresholds for Canny edge detection were calculated as 10% and 4% of the maximum gradient magnitude, respectively.

In comparing the Canny edge detection images with the first-derivative-of-Gaussian images, the edges themselves are much thinner in the Canny edge detection images. However, there also seems to be a little more noise/artifact that is picked-up in the Canny edge detection process. This could be due to the fact that both the upper and lower thresholds are fairly small, thus leading to an increase in the noise that is picked-up in the Canny edge detector. As σ is increased, it appears that it becomes more difficult to distinguish between weak edges to keep versus weak edges to disregard, as the edges seem to appear to be somewhat frayed. In particular, for the $\sigma = 7$ image in Figure 13, there seem to be less consistently strong edges. Thus, determining which edges to keep and which to reject may be less straightforward and result in the pattern observed in that image.

Finally, we applied a Canny edge detector to an image with a standard deviation equal to 3. The upper thresholds were then calculated as 10%, 20%, 30%, 40%, and 50% of the maximum gradient magnitude, and the lower thresholds were calculated as 40% of the corresponding upper threshold (Figure 14).

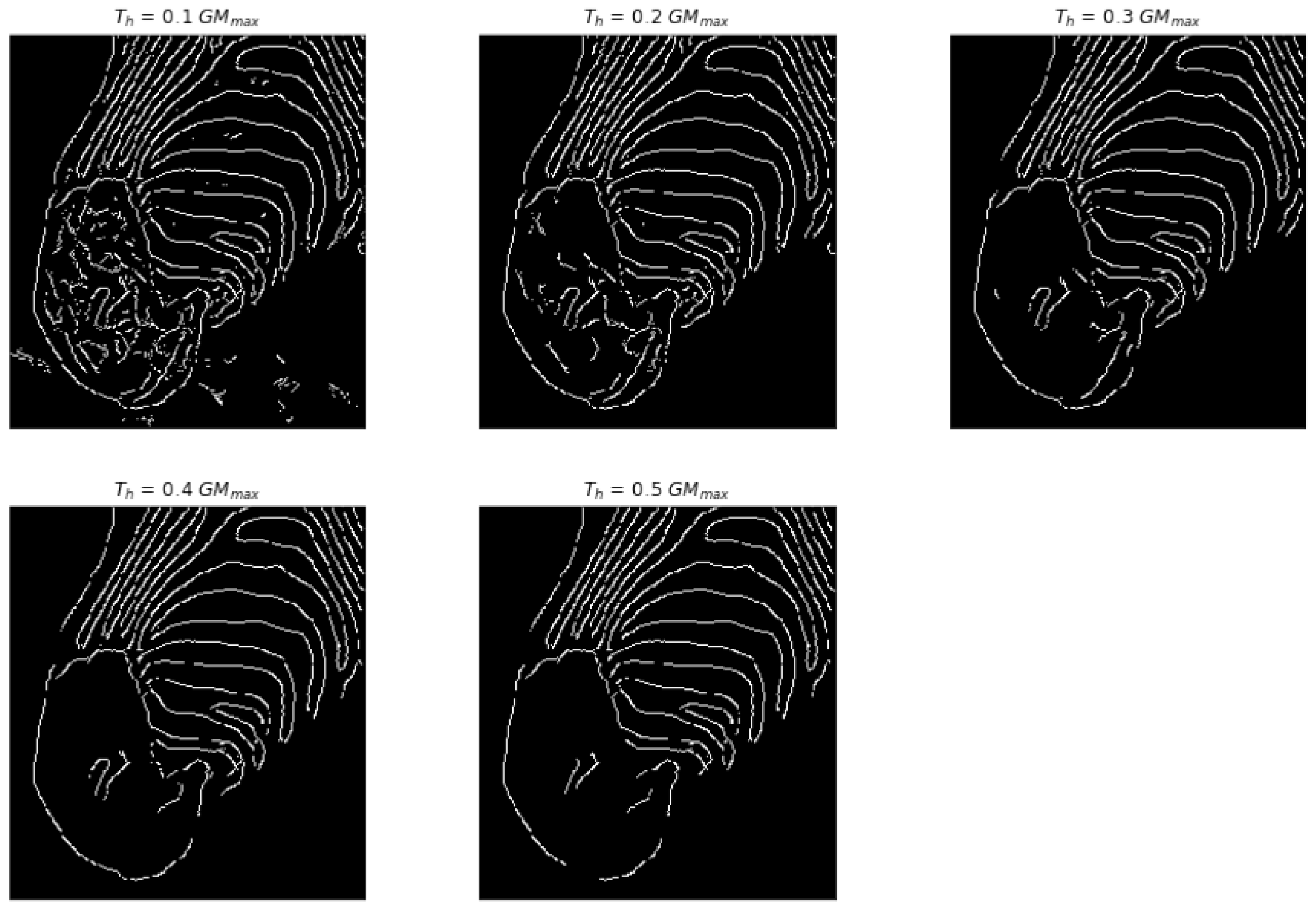


Fig. 14: Canny edge detection ($\sigma = 3$) with varying upper and lower thresholds. $T_h =$ upper threshold, $GM_{max} =$ maximum gradient magnitude.

In this process, we are widening the range of the upper and lower thresholds and shifting the threshold range upwards as we move from left to right across the images in Figure 14. In doing so, it seems that the noise in the edge-detected images is reduced dramatically. We can also see that some of the detail in the edge detection is lost as we move across images from left to right. At first, this is not the most intuitive observation. It is true that as the range between the upper and lower threshold increases, edges become noisier. However, we have to remember that we are also shifting the entire range upwards. In doing so, less noise will be picked-up in the Canny edge detection process.

References

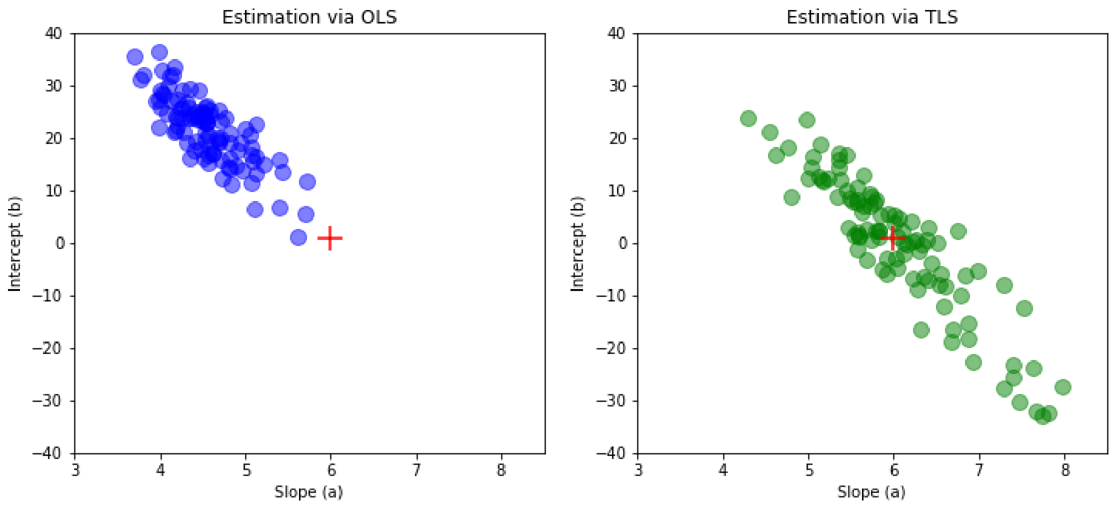
James R. Carr (2012) Orthogonal regression: a teaching perspective, International Journal of Mathematical Education in Science and Technology, 43:1, 134-143, DOI: 10.1080/0020739X.2011.573876

BIOS 7718-Assignment 1 Source Code

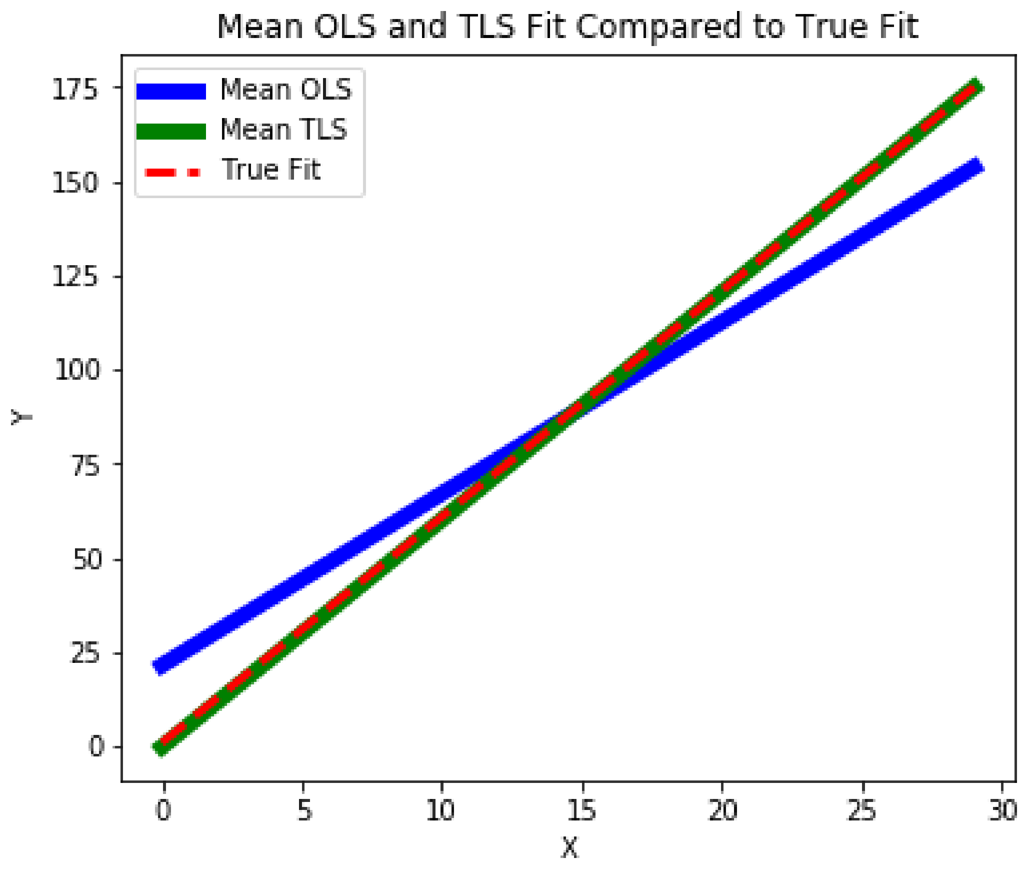
March 8, 2019

```
In [13]: ##### Problem 1.1 ####
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.linear_model import LinearRegression
         import scipy.odr as odr
         from scipy.stats import linregress

         # set seed
         np.random.seed(1212)

         results_a1 = []
         results_b1 = []
         results_a2 = []
         results_b2 = []

         for i in range(1,101):
             n = 30
             x = np.array(range(30)).reshape(-1,1)
             y = 6*x + 1
             v = np.concatenate((x, y), axis = 1)

             # generate noise
             mu = (0, 0)
             cov = [[25,0],[0,25]]
             err = np.random.multivariate_normal(mu, cov, n)
             v_final = np.round(v + err)

             # estimation method 1: vertical error (OLS)
             x_bar = np.mean(v_final[...,0])
             y_bar = np.mean(v_final[...,1])
             sq_diff_x = (v_final[...,0] - x_bar)**2
             ssd_x = np.sum(sq_diff_x)
             a1 = (np.sum((v_final[...,0] - x_bar)*(v_final[...,1] - y_bar)))/ssd_x
             b1 = y_bar - a1*x_bar

             # estimation method 2: orthogonal error (TLS)
             sq_diff_y = (v_final[...,1] - y_bar)**2
```

```
    ssd_y = np.sum(sq_diff_y)
    w = ssd_y - ssd_x
    r = 2*np.sum((v_final[...,0] - x_bar)*(v_final[...,1] - y_bar))
    a2 = (w + (w**2 + r**2)**(1/2))/r
    b2 = y_bar - a2*x_bar

    # save results
    results_a1.append(a1)
    results_b1.append(b1)
    results_a2.append(a2)
    results_b2.append(b2)

# all a and b values from the 100 simulations for each estimation method
a1_all = np.hstack(results_a1)
b1_all = np.hstack(results_b1)
a2_all = np.hstack(results_a2)
b2_all = np.hstack(results_b2)

# scatterplots of a and b values for each estimation method
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12,5))
ax1.scatter(a1_all, b1_all, color = 'b', alpha = 0.5, s=100)
ax1.scatter(6, 1, s=250, marker='+', color='r')
ax1.set_xlim([3,8.5])
ax1.set_ylim([-40,40])
ax1.set_xlabel('Slope (a)')
ax1.set_ylabel('Intercept (b)')
ax1.set_title('Estimation via OLS')
ax2.scatter(a2_all, b2_all, color = 'g', alpha = 0.5, s=100)
ax2.scatter(6, 1, s=250, marker='+', color='r')
ax2.set_xlim([3,8.5])
ax2.set_ylim([-40,40])
ax2.set_xlabel('Slope (a)')
ax2.set_ylabel('Intercept (b)')
ax2.set_title('Estimation via TLS')
plt.show()
```



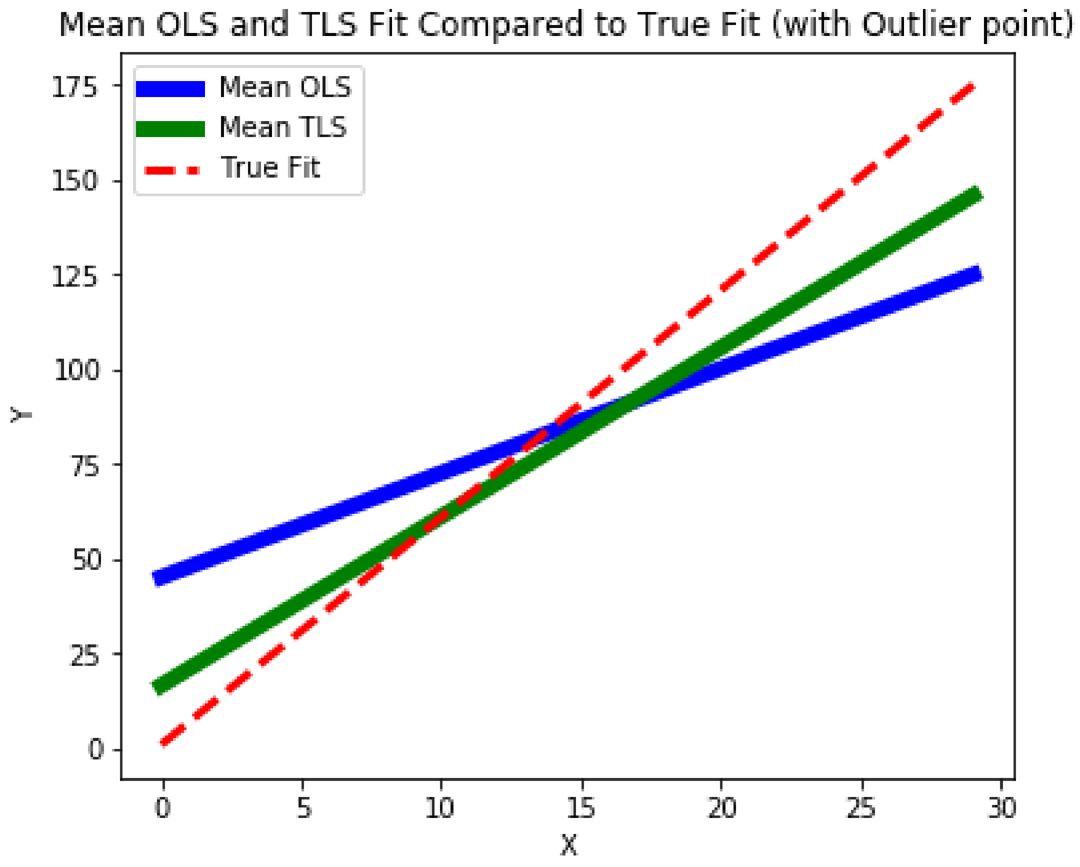
```
In [2]: # linear plots of mean and and b values for each estimation method
        mean_a1 = np.mean(a1_all)
        mean_b1 = np.mean(b1_all)
        mean_a2 = np.mean(a2_all)
        mean_b2 = np.mean(b2_all)

        ols_y = mean_a1*x + mean_b1
        tls_y = mean_a2*x + mean_b2
        true_y = 6*x + 1

        plt.figure(figsize = (6,5))
        plt.plot(x, ols_y, color='b', linewidth = 6, label='Mean OLS')
        plt.plot(x, tls_y, color='g', linewidth = 6, label='Mean TLS')
        plt.plot(x, true_y, color='r', linestyle = '--', linewidth = 3, label='True Fit')
        plt.legend()
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Mean OLS and TLS Fit Compared to True Fit')
        plt.show()
```



```
In [3]: ##### Problem 1.2 #####
        # set seed
        np.random.seed(1212)

        results_a1 = []
        results_b1 = []
        results_a2 = []
        results_b2 = []

        for i in range(1,101):
            n = 30
            x = np.array(range(30)).reshape(-1,1)
            y = 6*x + 1
            v = np.concatenate((x, y), axis = 1)

            # generate noise
            mu = (0, 0)
            cov = [[25,0],[0,25]]
            err = np.random.multivariate_normal(mu, cov, n)
            v_final = np.round(v + err)
```

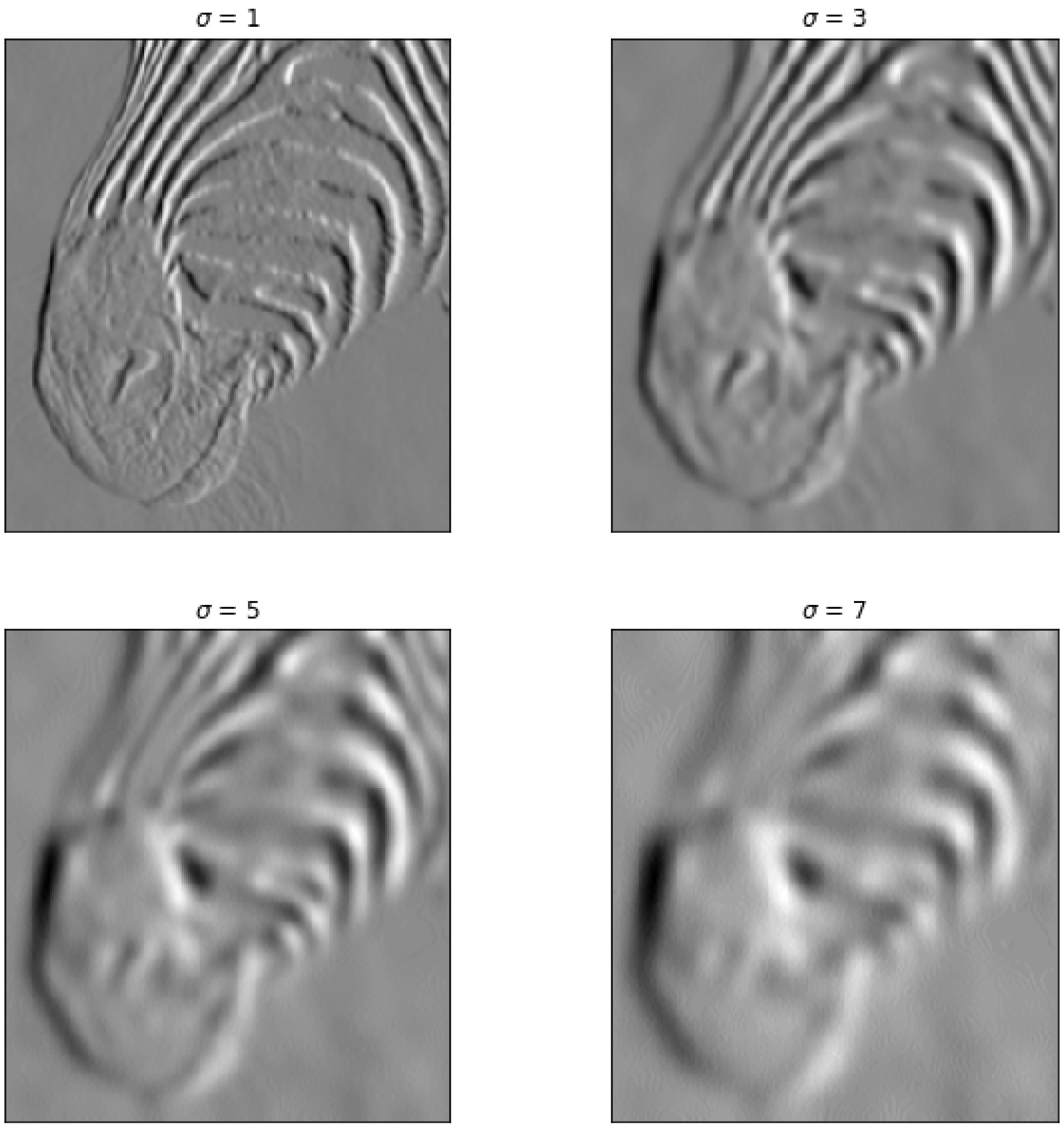
```
    # generate single data point with noise from different MVN distribution
    x_31 = np.array(31).reshape(-1,1)
    y_31 = 6*x_31 + 1
    v_31 = np.concatenate((x_31, y_31), axis = 1)
    mu_31 = (50, 2)
    err_31 = np.random.multivariate_normal(mu_31, cov, 1)
    v_final_31 = np.round(v_31 + err_31)

    # add to existing v_final and err vectors
    v_final_new = np.vstack([v_final, v_final_31])

    # estimation method 1: vertical error (OLS)
    x_bar = np.mean(v_final_new[...,0])
    y_bar = np.mean(v_final_new[...,1])
    sq_diff_x = (v_final_new[...,0] - x_bar)**2
    ssd_x = np.sum(sq_diff_x)
    a1 = (np.sum((v_final_new[...,0] - x_bar)*(v_final_new[...,1] - y_bar)))/ssd_x
    b1 = y_bar - a1*x_bar

    # estimation method 2: orthogonal error (TLS)
    sq_diff_y = (v_final_new[...,1] - y_bar)**2
    ssd_y = np.sum(sq_diff_y)
    w = ssd_y - ssd_x
    r = 2*np.sum((v_final_new[...,0] - x_bar)*(v_final_new[...,1] - y_bar))
    a2 = (w + (w**2 + r**2)**(1/2))/r
    b2 = y_bar - a2*x_bar

    # save results
    results_a1.append(a1)
    results_b1.append(b1)
    results_a2.append(a2)
    results_b2.append(b2)

# all a and b values from the 100 simulations for each estimation method
a1_all = np.hstack(results_a1)
b1_all = np.hstack(results_b1)
a2_all = np.hstack(results_a2)
b2_all = np.hstack(results_b2)

# linear plots of mean and and b values for each estimation method
# (with outlier 31st data point)
mean_a1 = np.mean(a1_all)
mean_b1 = np.mean(b1_all)
mean_a2 = np.mean(a2_all)
mean_b2 = np.mean(b2_all)

ols_y = mean_a1*x + mean_b1
```

```
tls_y = mean_a2*x + mean_b2
true_y = 6*x + 1

plt.figure(figsize = (6,5))
plt.plot(x, ols_y, color='b', linewidth = 6, label='Mean OLS')
plt.plot(x, tls_y, color='g', linewidth = 6, label='Mean TLS')
plt.plot(x, true_y, color='r', linestyle = '--', linewidth = 3, label='True Fit')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Mean OLS and TLS Fit Compared to True Fit (with Outlier point)')
plt.show()
```



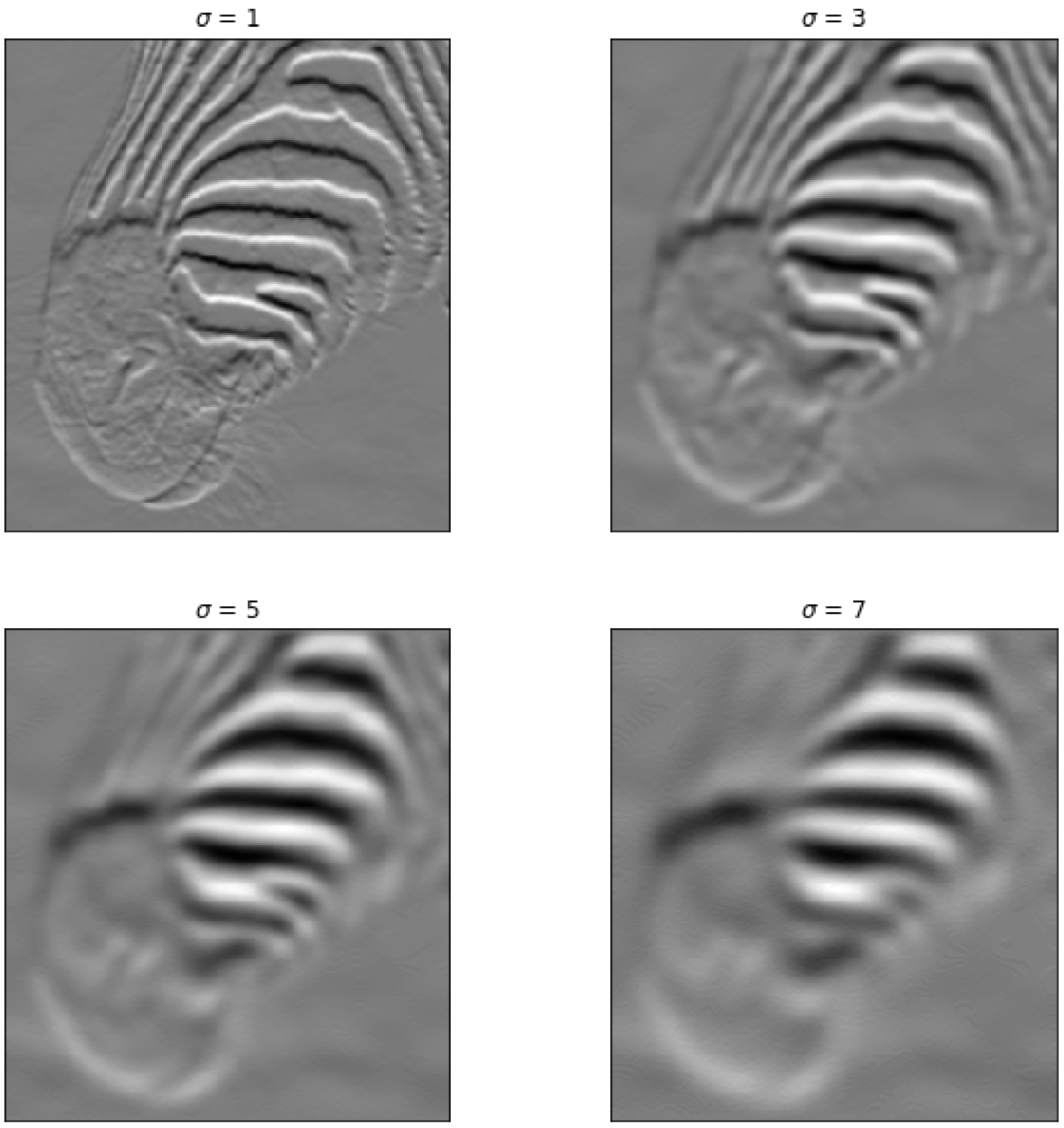
```
In [4]: ##### Problem 2.1 ####
        import cv2

        # read image
        img = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignme

        # Gaussian filtering (kernel size equals 6*sigma + 1)
        img_gaus1 = cv2.GaussianBlur(img, (7,7), 1)
        img_gaus3 = cv2.GaussianBlur(img, (19,19), 3)
        img_gaus5 = cv2.GaussianBlur(img, (31,31), 5)
        img_gaus7 = cv2.GaussianBlur(img, (43,43), 7)
        img_gaus9 = cv2.GaussianBlur(img, (55,55), 9)

        compare_filt1 = np.hstack((img, img_gaus1, img_gaus3))
        compare_filt2 = np.hstack((img_gaus5, img_gaus7, img_gaus9))
        compare_filt_gaus = np.vstack((compare_filt1, compare_filt2))
        cv2.imwrite('compare_filt_gaus.png', compare_filt_gaus)

        # median filtering
        img_med3 = cv2.medianBlur(img, 3)
        img_med5 = cv2.medianBlur(img, 5)
        img_med7 = cv2.medianBlur(img, 7)
        img_med9 = cv2.medianBlur(img, 9)
        img_med11 = cv2.medianBlur(img, 11)

        compare_filt1 = np.hstack((img, img_med3, img_med5))
        compare_filt2 = np.hstack((img_med7, img_med9, img_med11))
        compare_filt_med = np.vstack((compare_filt1, compare_filt2))
        # cv2.imwrite('compare_filt_med.png', compare_filt_med)

In [5]: ##### Problem 2.2 #####
        # Gaussian filtering (kernel size equals 12*sigma + 1)
        img2_gaus1 = cv2.GaussianBlur(img, (13,13), 1)
        img2_gaus3 = cv2.GaussianBlur(img, (37,37), 3)
        img2_gaus5 = cv2.GaussianBlur(img, (61,61), 5)
        img2_gaus7 = cv2.GaussianBlur(img, (85,85), 7)
        img2_gaus9 = cv2.GaussianBlur(img, (109,109), 9)

        compare_filt1 = np.hstack((img, img2_gaus1, img2_gaus3))
        compare_filt2 = np.hstack((img2_gaus5, img2_gaus7, img2_gaus9))
        compare_filt_gaus = np.vstack((compare_filt1, compare_filt2))
        # cv2.imwrite('compare_filt_gaus_2.png', compare_filt_gaus)

In [6]: # compare images using absolute difference in maximum intesity value
        import pandas as pd

        diff1 = np.abs(np.max(img_gaus1) - np.max(img2_gaus1))
        diff3 = np.abs(np.max(img_gaus3) - np.max(img2_gaus3))
        diff5 = np.abs(np.max(img_gaus5) - np.max(img2_gaus5))
        diff7 = np.abs(np.max(img_gaus7) - np.max(img2_gaus7))
        diff9 = np.abs(np.max(img_gaus9) - np.max(img2_gaus9))

        table = pd.DataFrame({'$\sigma$': ['1', '3', '5', '7', '9'],
                             '|Difference|': [diff1, diff3, diff5, diff7, diff9]})
        table.set_index('$\sigma$', inplace = True)
```

```
In [7]: ##### Problem 3.1 #####
        img = cv2.imread('/Users/piper/Piper Documents/Biomedical Imaging/Assignments/Assignme

        # smooth images via Gaussian filtering
        img_gaus1 = cv2.GaussianBlur(img, (7,7), 1)
        img_gaus3 = cv2.GaussianBlur(img, (19,19), 3)
        img_gaus5 = cv2.GaussianBlur(img, (31,31), 5)
        img_gaus7 = cv2.GaussianBlur(img, (43,43), 7)

        # derivative of Gaussian Filter (Sobel)
        img_sobel1x = cv2.Sobel(img_gaus1, cv2.CV_64F, 1, 0, ksize=3)
        img_sobel1y = cv2.Sobel(img_gaus1, cv2.CV_64F, 0, 1, ksize=3)
        img_sobel3x = cv2.Sobel(img_gaus3, cv2.CV_64F, 1, 0, ksize=3)
        img_sobel3y = cv2.Sobel(img_gaus3, cv2.CV_64F, 0, 1, ksize=3)
        img_sobel5x = cv2.Sobel(img_gaus5, cv2.CV_64F, 1, 0, ksize=3)
        img_sobel5y = cv2.Sobel(img_gaus5, cv2.CV_64F, 0, 1, ksize=3)
        img_sobel7x = cv2.Sobel(img_gaus7, cv2.CV_64F, 1, 0, ksize=3)
        img_sobel7y = cv2.Sobel(img_gaus7, cv2.CV_64F, 0, 1, ksize=3)

        # calculate gradient magnitude for each image
        grad_mag1 = ((img_sobel1x)**2 + (img_sobel1y)**2)**(1/2)
        grad_mag3 = ((img_sobel3x)**2 + (img_sobel3y)**2)**(1/2)
        grad_mag5 = ((img_sobel5x)**2 + (img_sobel5y)**2)**(1/2)
        grad_mag7 = ((img_sobel7x)**2 + (img_sobel7y)**2)**(1/2)

        # generate thresholds (= 20% of max gradient magnitude)
        thresh1 = (np.max(grad_mag1))*0.2
        thresh3 = (np.max(grad_mag3))*0.2
        thresh5 = (np.max(grad_mag5))*0.2
        thresh7 = (np.max(grad_mag7))*0.2

        # threshold the gradient magnitude images to get binary image
        retval, img_thresh1 = cv2.threshold(grad_mag1, thresh1, 255, cv2.THRESH_BINARY)
        retval, img_thresh3 = cv2.threshold(grad_mag3, thresh3, 255, cv2.THRESH_BINARY)
        retval, img_thresh5 = cv2.threshold(grad_mag5, thresh5, 255, cv2.THRESH_BINARY)
        retval, img_thresh7 = cv2.threshold(grad_mag7, thresh7, 255, cv2.THRESH_BINARY)

        # plot x-derivative images for all sigmas
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (10,10))
        ax1.imshow(img_sobel1x, cmap = 'gray')
        ax1.set_xticks([]), ax1.set_yticks([])
        ax1.set_title('$\sigma$ = 1')
        ax2.imshow(img_sobel3x, cmap = 'gray')
        ax2.set_xticks([]), ax2.set_yticks([])
        ax2.set_title('$\sigma$ = 3')
        ax3.imshow(img_sobel5x, cmap = 'gray')
        ax3.set_xticks([]), ax3.set_yticks([])
        ax3.set_title('$\sigma$ = 5')
        ax4.imshow(img_sobel7x, cmap = 'gray')
        ax4.set_xticks([]), ax4.set_yticks([])
        ax4.set_title('$\sigma$ = 7')
        plt.show()
```



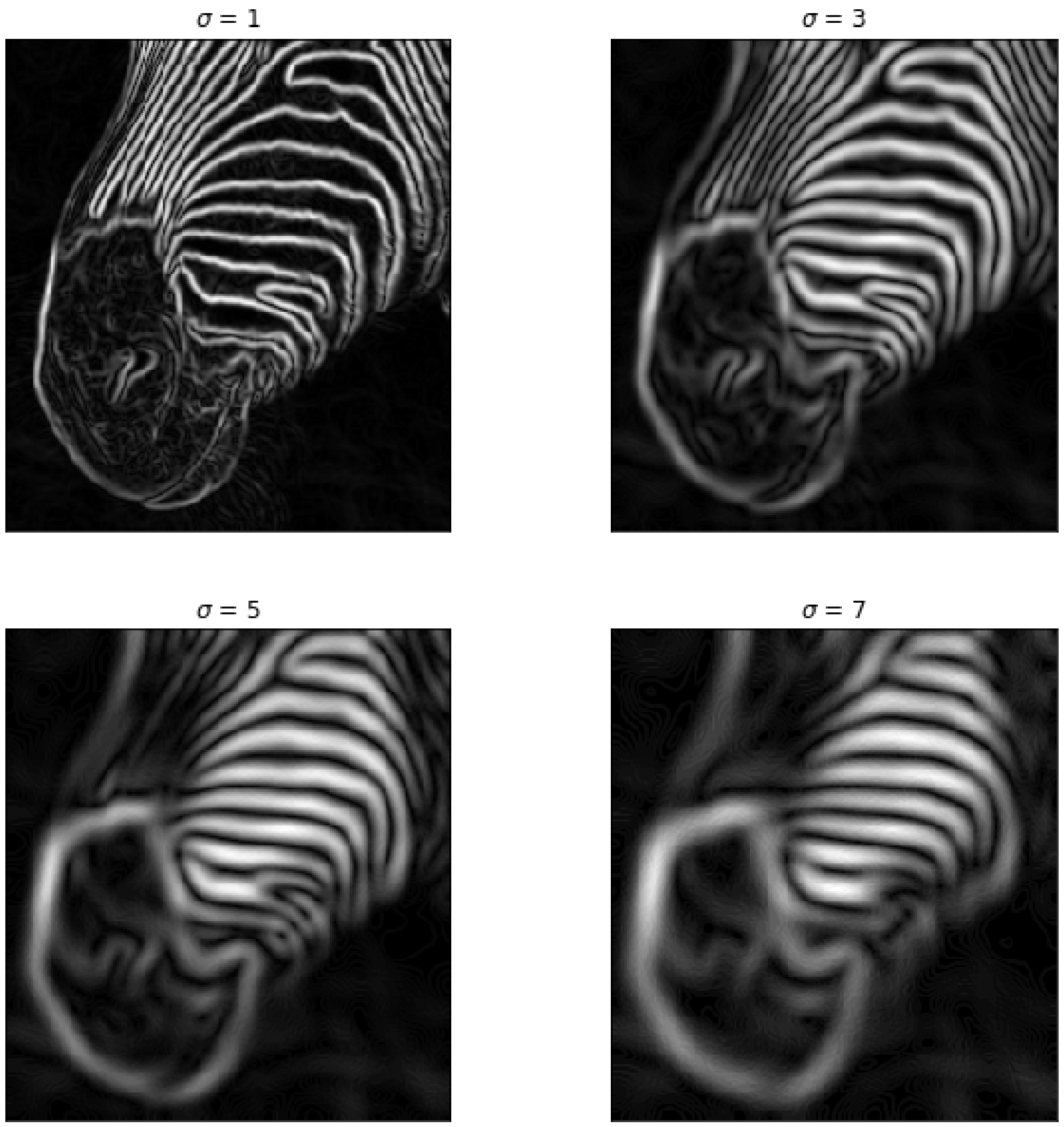
```
In [8]: # plot y-derivative images for all sigmas
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (10,10))
        ax1.imshow(img_sobel1y, cmap = 'gray')
        ax1.set_xticks([]), ax1.set_yticks([])
        ax1.set_title('$\sigma$ = 1')
        ax2.imshow(img_sobel3y, cmap = 'gray')
```

```
ax2.set_xticks([]), ax2.set_yticks([])
ax2.set_title('$\sigma$ = 3')
ax3.imshow(img_sobel5y, cmap = 'gray')
ax3.set_xticks([]), ax3.set_yticks([])
ax3.set_title('$\sigma$ = 5')
ax4.imshow(img_sobel7y, cmap = 'gray')
ax4.set_xticks([]), ax4.set_yticks([])
ax4.set_title('$\sigma$ = 7')
plt.show()
```

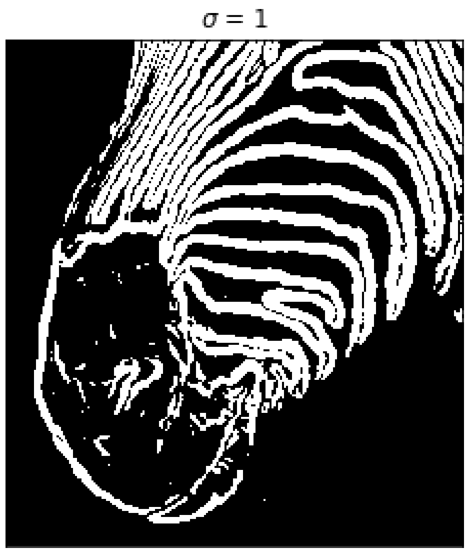


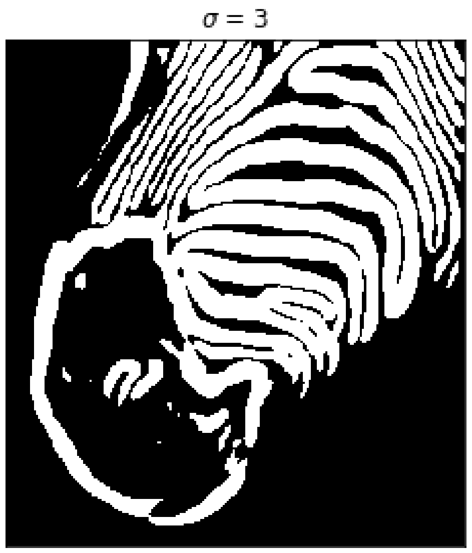
```
In [9]: # plot gradient magnitude images for all sigmas
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (10,10))
```

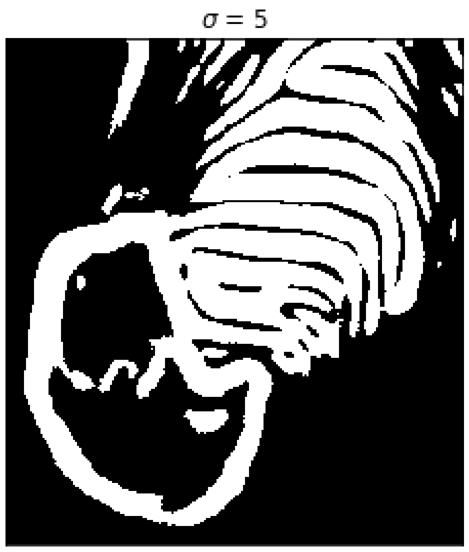
```
ax1.imshow(grad_mag1, cmap = 'gray')
ax1.set_xticks([]), ax1.set_yticks([])
ax1.set_title('$\sigma$ = 1')
ax2.imshow(grad_mag3, cmap = 'gray')
ax2.set_xticks([]), ax2.set_yticks([])
ax2.set_title('$\sigma$ = 3')
ax3.imshow(grad_mag5, cmap = 'gray')
ax3.set_xticks([]), ax3.set_yticks([])
ax3.set_title('$\sigma$ = 5')
ax4.imshow(grad_mag7, cmap = 'gray')
ax4.set_xticks([]), ax4.set_yticks([])
ax4.set_title('$\sigma$ = 7')
plt.show()
```

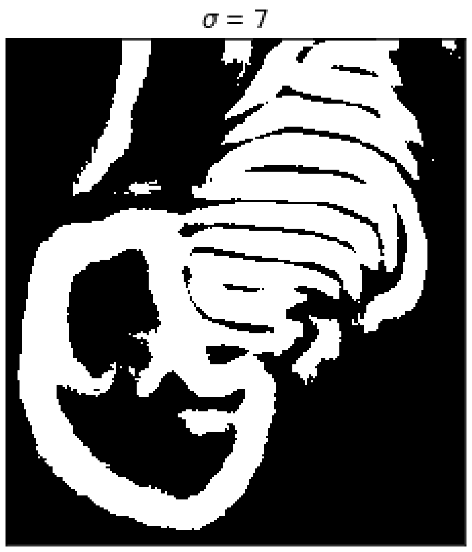


```
In [10]: # plot binary images for all sigmas
         fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (10,10))
         ax1.imshow(img_thresh1, cmap = 'gray')
         ax1.set_xticks([]), ax1.set_yticks([])
         ax1.set_title('$\sigma$ = 1')
         ax2.imshow(img_thresh3, cmap = 'gray')
         ax2.set_xticks([]), ax2.set_yticks([])
         ax2.set_title('$\sigma$ = 3')
         ax3.imshow(img_thresh5, cmap = 'gray')
         ax3.set_xticks([]), ax3.set_yticks([])
         ax3.set_title('$\sigma$ = 5')
         ax4.imshow(img_thresh7, cmap = 'gray')
         ax4.set_xticks([]), ax4.set_yticks([])
         ax4.set_title('$\sigma$ = 7')
         plt.show()
```



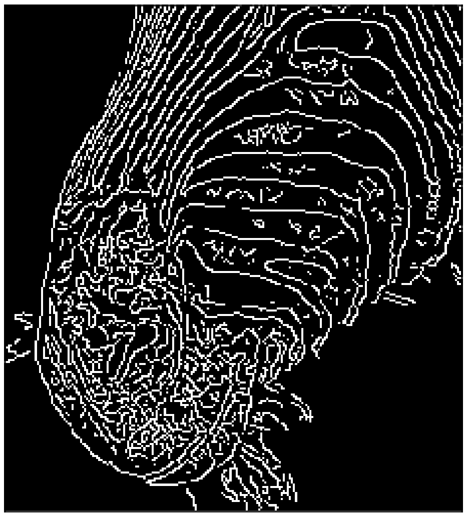


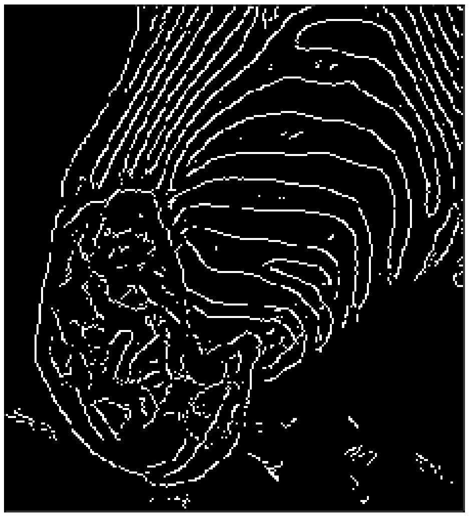


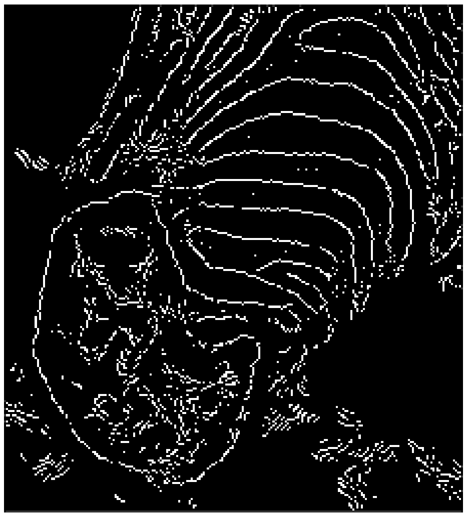


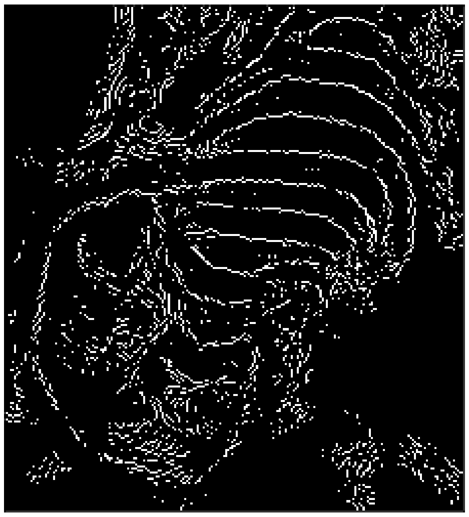
```
In [11]: ##### Problem 3.2 #####
         # find thresholds (high and low)
         thresh1_high = (np.max(grad_mag1))*0.1
         thresh3_high = (np.max(grad_mag3))*0.1
         thresh5_high = (np.max(grad_mag5))*0.1
         thresh7_high = (np.max(grad_mag7))*0.1

         thresh1_low = (np.max(grad_mag1))*0.04
         thresh3_low = (np.max(grad_mag3))*0.04
         thresh5_low = (np.max(grad_mag5))*0.04
         thresh7_low = (np.max(grad_mag7))*0.04
```

```
# Canny edge detection
img_canny1 = cv2.Canny(img_gaus1, thresh1_low, thresh1_high)
img_canny3 = cv2.Canny(img_gaus3, thresh3_low, thresh3_high)
img_canny5 = cv2.Canny(img_gaus5, thresh5_low, thresh5_high)
img_canny7 = cv2.Canny(img_gaus7, thresh7_low, thresh7_high)

# plot canny edge detection images for all sigmas
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize = (10,10))
ax1.imshow(img_canny1, cmap = 'gray')
ax1.set_xticks([]), ax1.set_yticks([])
ax1.set_title('$\sigma$ = 1')
ax2.imshow(img_canny3, cmap = 'gray')
ax2.set_xticks([]), ax2.set_yticks([])
ax2.set_title('$\sigma$ = 3')
ax3.imshow(img_canny5, cmap = 'gray')
ax3.set_xticks([]), ax3.set_yticks([])
ax3.set_title('$\sigma$ = 5')
ax4.imshow(img_canny7, cmap = 'gray')
ax4.set_xticks([]), ax4.set_yticks([])
ax4.set_title('$\sigma$ = 7')
plt.show()
```


$\sigma = 1$

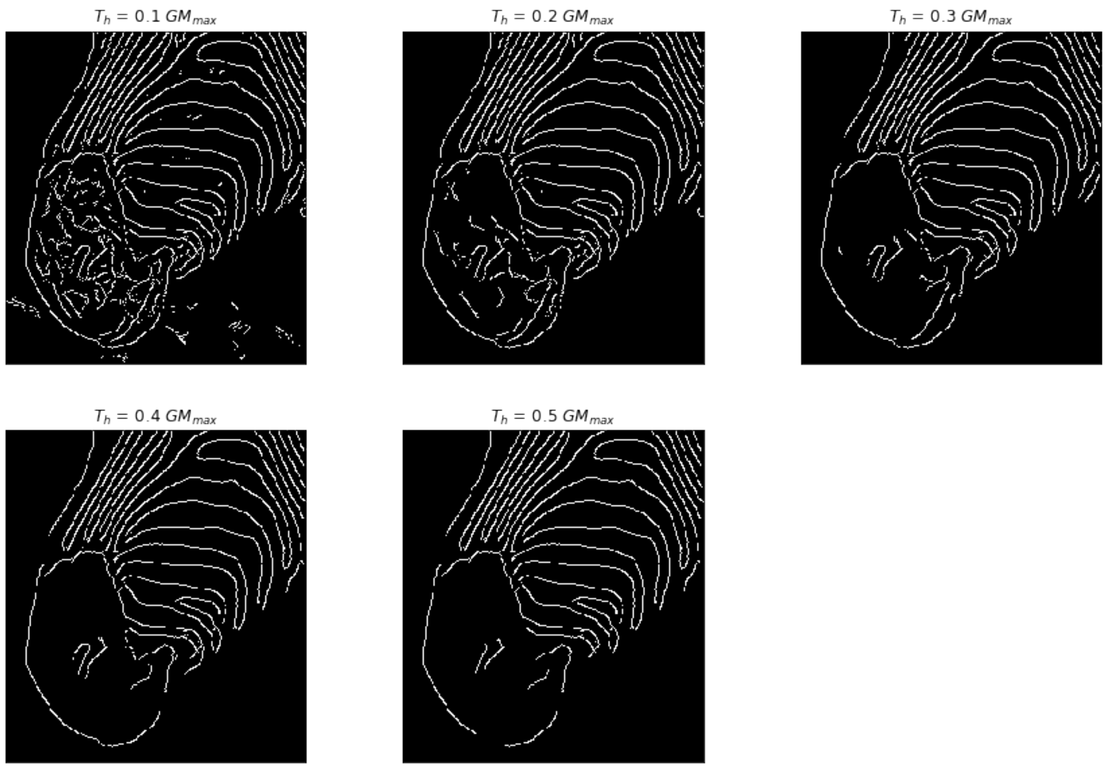

$\sigma = 3$


$\sigma = 5$


$\sigma = 7$

```
In [12]: # constant sigma (=3), varying thresholds
         # find thresholds (high and low)
         thresh10_high = (np.max(grad_mag3))*0.1
         thresh20_high = (np.max(grad_mag3))*0.2
         thresh30_high = (np.max(grad_mag3))*0.3
         thresh40_high = (np.max(grad_mag3))*0.4
         thresh50_high = (np.max(grad_mag3))*0.5

         thresh10_low = thresh10_high*0.4
         thresh20_low = thresh20_high*0.4
         thresh30_low = thresh30_high*0.4
```

```
thresh40_low = thresh40_high*0.4
thresh50_low = thresh50_high*0.4

# Canny edge detection
img_canny10 = cv2.Canny(img_gaus3, thresh10_low, thresh10_high)
img_canny20 = cv2.Canny(img_gaus3, thresh20_low, thresh20_high)
img_canny30 = cv2.Canny(img_gaus3, thresh30_low, thresh30_high)
img_canny40 = cv2.Canny(img_gaus3, thresh40_low, thresh40_high)
img_canny50 = cv2.Canny(img_gaus3, thresh50_low, thresh50_high)

# plot canny edge detection images for all thresholds
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(2, 3, figsize = (15,10))
ax1.imshow(img_canny10, cmap = 'gray')
ax1.set_xticks([]), ax1.set_yticks([])
ax1.set_title('$T_h$ = 0.1 $GM_{max}$')
ax2.imshow(img_canny20, cmap = 'gray')
ax2.set_xticks([]), ax2.set_yticks([])
ax2.set_title('$T_h$ = 0.2 $GM_{max}$')
ax3.imshow(img_canny30, cmap = 'gray')
ax3.set_xticks([]), ax3.set_yticks([])
ax3.set_title('$T_h$ = 0.3 $GM_{max}$')
ax4.imshow(img_canny40, cmap = 'gray')
ax4.set_xticks([]), ax4.set_yticks([])
ax4.set_title('$T_h$ = 0.4 $GM_{max}$')
ax5.imshow(img_canny50, cmap = 'gray')
ax5.set_xticks([]), ax5.set_yticks([])
ax5.set_title('$T_h$ = 0.5 $GM_{max}$')
fig.delaxes(ax6)
plt.show()
```


T_h = 0.1 GM_{max}
T_h = 0.2 GM_{max}
T_h = 0.3 GM_{max}
T_h = 0.4 GM_{max}
T_h = 0.5 GM_{max}