In [4]:

```
from __future__ import print_function
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2
from IPython.display import HTML, YouTubeVideo
import matplotlib.patches as patches
from matplotlib.lines import Line2D
```

Enter your name below and run the cell:

Individual cells can be run with `Ctrl` + `Enter`

In [5]:

```
# Alicia He
```

https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/more-on-regression/v/squared-error-of-regression-line

Note: All Khan Academy content is available for free at khanacademy.org

In [4]:

```
YouTubeVideo('6OvhLPS7rj4', width=560, height=315)
```

Out[4]:

In [5]:

```
YouTubeVideo('mIx2Oj5y9Q8', width=560, height=315)
```

Out[5]:

In [ ]:
```
YouTubeVideo('f6OnoxctvUk', width=560, height=315)
```

In [ ]:
```
YouTubeVideo('u1HhUB3NP8g', width=560, height=315)
```

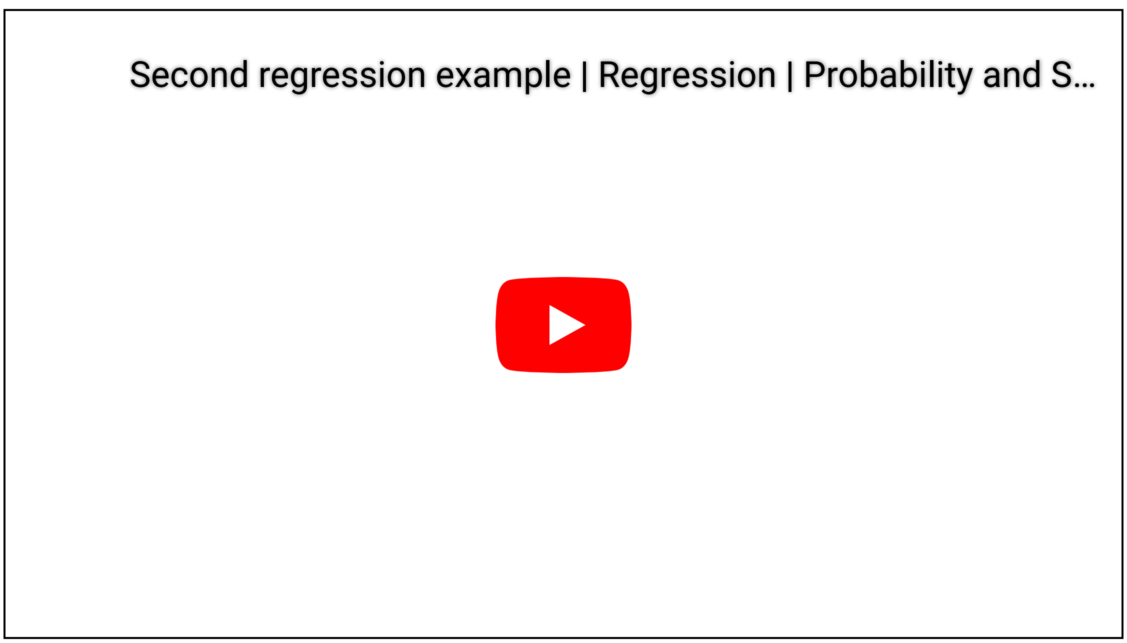
In [ ]:
```
YouTubeVideo('8RSTQl0bQuw', width=560, height=315)
```

In [ ]:
```
YouTubeVideo('GAmzwIkGFgE', width=560, height=315)
```

The last video is optional

In [8]:
```
YouTubeVideo('ww_yT9ckPWw', width=560, height=315)
```

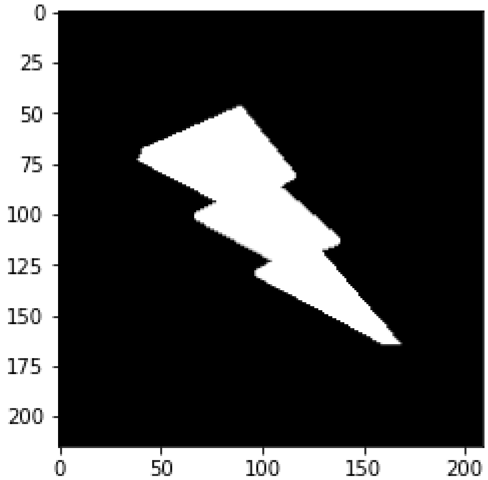
Out[8]:



In [6]:
```
lightningbolt = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GRAYSCALE)
_, lightningbolt = cv2.threshold(lightningbolt,150,255,cv2.THRESH_BINARY)
```

```
print(lightningbolt.shape)
fig,ax = plt.subplots()
ax.imshow(lightningbolt, cmap='gray');
```

```
(215, 209)
```



In [10]:
```
np.argwhere?
```

In [7]:
```
bolt = np.argwhere(lightningbolt)
bolt
```

Out[7]:
```
array([[ 47,  88],
       [ 47,  89],
       [ 47,  90],
       ...,
       [164, 166],
       [164, 167],
       [164, 168]])
```

Linear Regression

$$m = \frac{\bar{x}\bar{y} - \overline{xy}}{(\bar{x})^2 - \overline{x^2}}$$

$$b = \bar{y} - m\bar{x}$$

Question: how can we extract the xs and ys separately from the result of argwhere?

Hint: review numpy slicing by columns and rows

In [ ]:
```
# We can slice the array with [:, :, 1] to get the xs and [:, :, 0] to get the ys.
```

Question: Why would we want to convert x and y points from int values to floats?

In [8]:
```python
# We would want to convert x and y points to floats because it's easier to take the
```

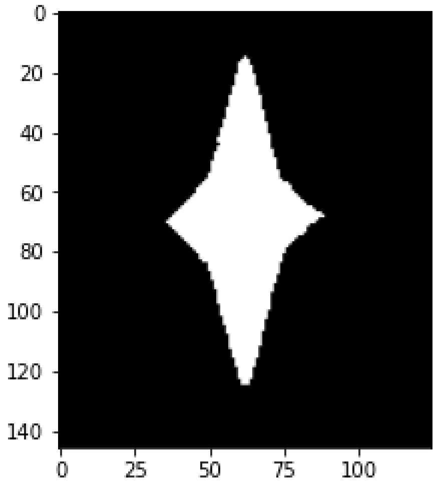
In [17]:
```python
def calculate_regression(points): # input is the result of np.argwhere
    # convert points to float
    points = points.astype(float) # (see astype, https://docs.scipy.org/doc/numpy/r

    xs = points[:, 1]
    ys = points[:, 0]
    x_mean = np.mean(xs)
    y_mean = np.mean(ys)

    xy_mean = np.mean(xs * ys)

    x_squared_mean = np.mean(xs ** 2)

    m = (xy_mean - (x_mean * y_mean))/(x_squared_mean - (x_mean)**2)

    b = y_mean - (m * x_mean)

    return (m,b)
```

The intercept we calculated, b, may be outside of the pixel space of the image, so we must find two points inside of pixel space, (x_1, y_1) and $(x2, y2)$ which will allow us to plot our regression line on the image. It may be best to choose points on the regression line which also occur on the boundaries/extrema of the image.

In [18]:
```python
def find_inliers(m, b, shape):
    intersect1 = -b/m # Finding where the line intersects the top and bottom lines
                      # image (x-coordinate, horizontal lines)
    intersect2 = (shape[0] - b)/m

    x1 = min(max(intersect1, 0), shape[1]) # Assigning x-value based on whether the
                                           # intersection occurs within the bounda
                                           # of the image
    x2 = min(max(intersect2, 0), shape[1])

    y1 = m * x1 + b # Finding y values
    y2 = m * x2 + b

    return [x1, x2, y1, y2]
```

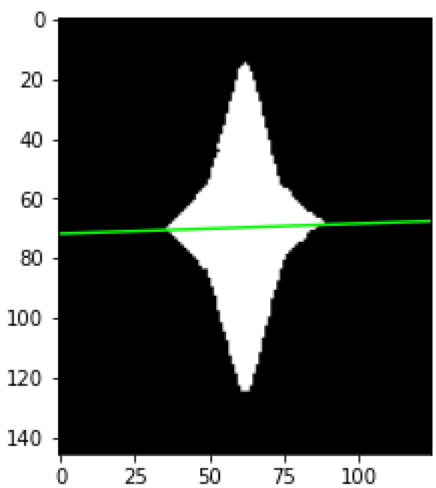
In [19]:
```python
star = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GRAYSCALE)
print(star.shape)

_, star = cv2.threshold(star,125,255,cv2.THRESH_BINARY)
fig,ax = plt.subplots()
ax.imshow(star, cmap='gray');
```

```
(146, 125)
```



In [20]:
```
m,b = calculate_regression(np.argwhere(star))
_ = find_inliers(m,b, star.shape)
```

In [21]:
```
# below is an example of how to draw a random line from (10,25) to (10,55)
# TODO: replace this with the result of find_inliers
# -- pay attention to the directions of the x and y axes
#    in image space, row-column space, and cartesian space
# Look at the help function for Line2D below

fig,ax = plt.subplots()
ax.imshow(star, cmap='gray');
regression = Line2D([_[0],_[1]],[_[2],_[3]], color='lime')
ax.add_line(regression);
```



In [46]:
```
Line2D?
```

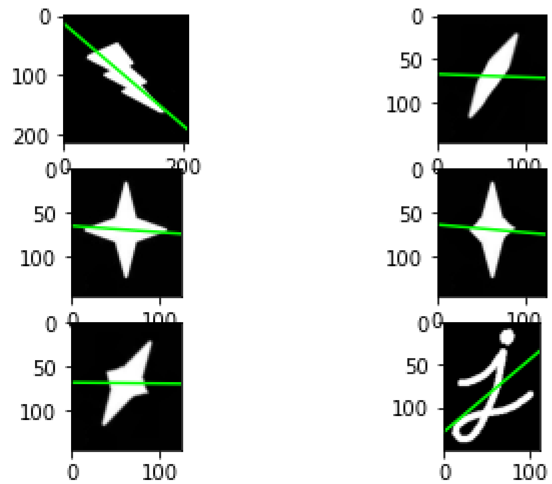
TODO

1. Run your linear regression algorithm on the following images.
2. Plot each of the results.
3. Include each result in your submitted PDF.

In [22]:

```
lightningbolt      = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GRAYSCALE)
blob               = cv2.imread('shapes/blob.png', cv2.IMREAD_GRAYSCALE)
star               = cv2.imread('shapes/star.png', cv2.IMREAD_GRAYSCALE)
squishedstar       = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GRAYSCALE)
squishedturnedstar = cv2.imread('shapes/squishedturnedstar.png', cv2.IMREAD_GRAYSCA
letterj            = cv2.imread('shapes/letterj.png', cv2.IMREAD_GRAYSCALE)

images = [lightningbolt, blob, star, squishedstar, squishedturnedstar, letterj]

# Lightnigbolt
m1,b1 = calculate_regression(np.argwhere(lightningbolt))
_1 = find_inliers(m1,b1, lightningbolt.shape)

fig,ax = plt.subplots(nrows=3, ncols=2)
ax[0, 0].imshow(lightningbolt, cmap='gray');
regression1 = Line2D([_1[0],_1[1]],[_1[2],_1[3]], color='lime')
ax[0, 0].add_line(regression1);

# Blob
m2,b2 = calculate_regression(np.argwhere(blob))
_2 = find_inliers(m2,b2, blob.shape)

ax[0, 1].imshow(blob, cmap='gray')
regression2 = Line2D([_2[0],_2[1]],[_2[2],_2[3]], color='lime')
ax[0, 1].add_line(regression2);

# Star
m3,b3 = calculate_regression(np.argwhere(star))
_3 = find_inliers(m3,b3, star.shape)

ax[1, 0].imshow(star, cmap='gray')
regression3 = Line2D([_3[0],_3[1]],[_3[2],_3[3]], color='lime')
ax[1, 0].add_line(regression3);

# Squished Star
m4,b4 = calculate_regression(np.argwhere(squishedstar))
_4 = find_inliers(m4,b4, squishedstar.shape)

ax[1, 1].imshow(squishedstar, cmap='gray')
regression4 = Line2D([_4[0],_4[1]],[_4[2],_4[3]], color='lime')
ax[1, 1].add_line(regression4);

# Squished Turned Star
m5,b5 = calculate_regression(np.argwhere(squishedturnedstar))
_5 = find_inliers(m5,b5, squishedturnedstar.shape)

ax[2, 0].imshow(squishedturnedstar, cmap='gray')
regression5 = Line2D([_5[0],_5[1]],[_5[2],_5[3]], color='lime')
ax[2, 0].add_line(regression5);

# Letter J
m6,b6 = calculate_regression(np.argwhere(letterj))
_6 = find_inliers(m6,b6, letterj.shape)

ax[2, 1].imshow(letterj, cmap='gray')
```

```
regression6 = Line2D([_6[0],_6[1]],[_6[2],_6[3]], color='lime')
ax[2, 1].add_line(regression6);
```



When you are done:

You should have six images with regression lines plotted on top of them.

1. Double-check that you filled in your name at the top of the notebook!
2. Click `File` -> `Export Notebook As` -> `PDF`
3. Email the PDF to `YOURTEAMNAME@beaver.works`

Stretch goal

Implement a machine learning algorithm!

Random **Sa**mple **C**onsensus, commonly referred to as *RANSAC*, is one of the most widely used machine learning algorithms. In essence, it is a 'guess and check' algorithm. Take a small random sample of your data - two points in this case. Next, define a line through those two points. After doing so, count the number of *inliers*, or points closest to that line (euclidean distance is one way to do this).

https://en.wikipedia.org/wiki/Random_sample_consensus

Implement RANSAC for linear regression, and run it on all of your images.