Caitlyn Jones

Prof. Yangyang Wang

MATH 40A: Intro to Applied Math

23 October 2024

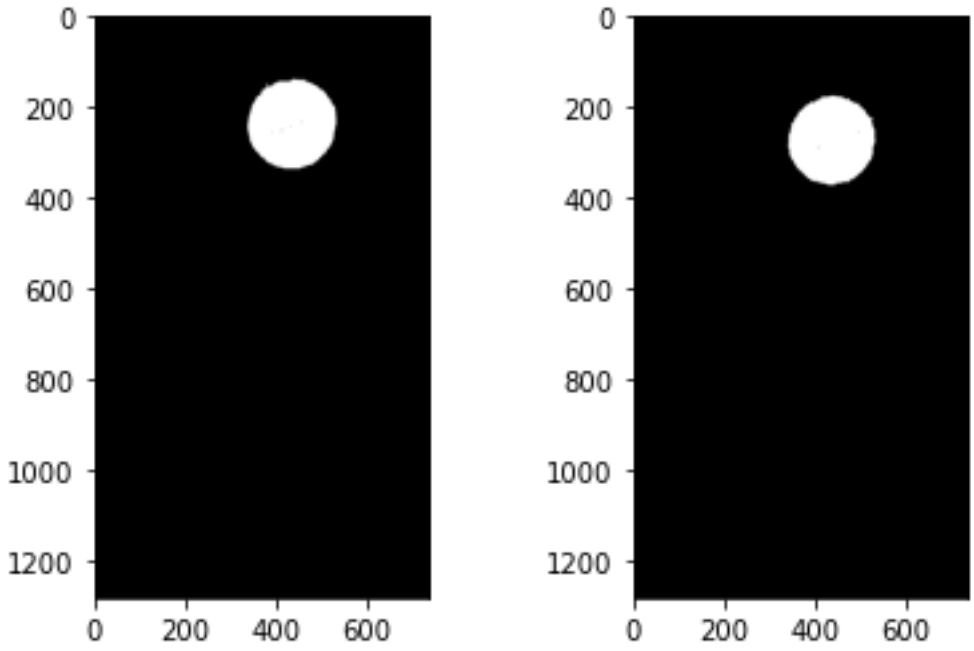
Homework 3

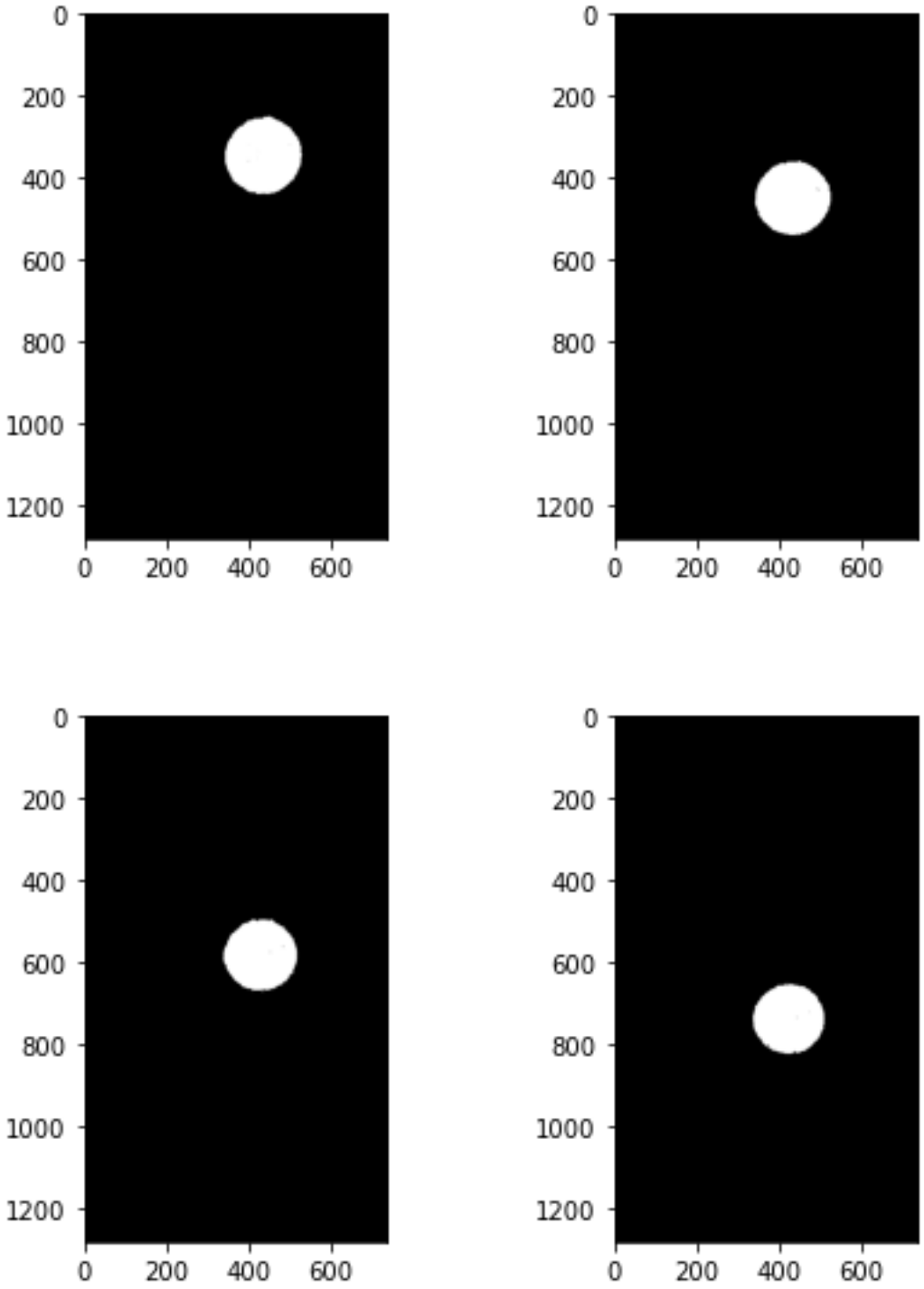
1.

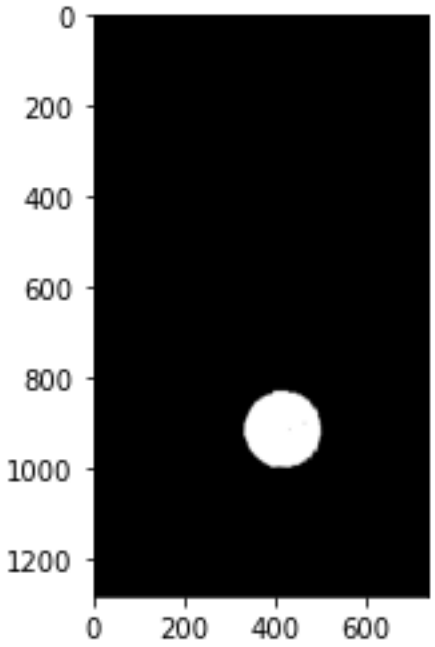
   a. In `process.py`, I created a function `process_image(image)`, where `image` is the original image that will be converted to black and white; the ball will be white, and everything else will be black. To begin, I wrote the first three lines `process.py` into `process_image(image)`. Then, I added a statement to ensure that the shape is in range. Then, I converted the rgb image to hsv and created the upper and lower bounds for the color of the blue hue. Then, I isolated the ball using the range in hue in order to make it white and the background black. I then denoised the image to remove all the extra white space. Lastly, to see the images, I ran `process_image(image)` seven times in order to show the progression of the ball dropping.

```python
def process_image(image):
    a = io.imread(image)
    a = ski.img_as_float64(a)
    a/=255.

    if a.shape[-1] == 4:

       a = a[:, :, :3]

    hsv_image = color.rgb2hsv(a)
    blue_upper = 0.55
    blue_lower = 0.62

    isolate = (hsv_image[:, :, 0] >= blue_upper) & (hsv_image[:, :, 0] <= blue_lower)

    grayscale = np.where(isolate, 1, 0)

    denoised = morphology.remove_small_objects(grayscale.astype(bool), min_size=1000)

    plt.figure()

    plt.imshow(denoised, cmap = "gray")

for k in range(1,8):
    process_image("drop%d.png" %(k))
    plt.pause(0.1)
```

Here are the results.




0
200
400
600
800
1000
1200
0
200
400
600



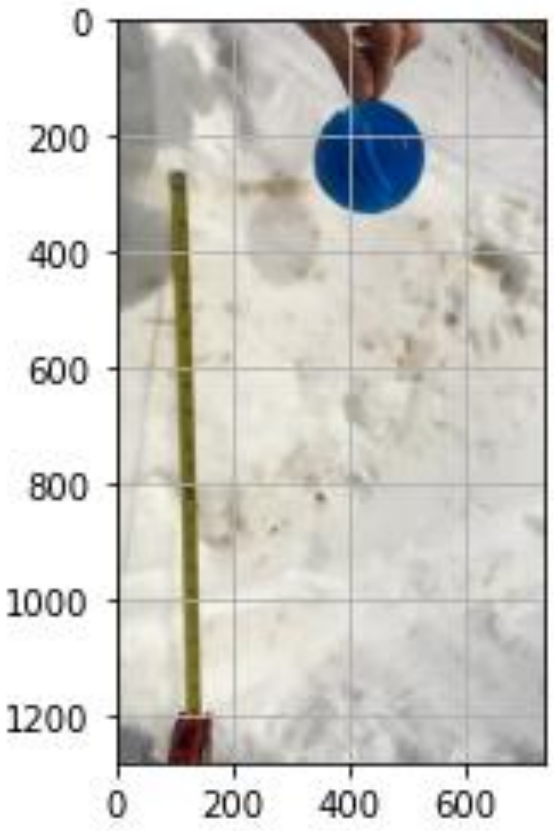
b. To compute the averages, I created a function `position_averages(image)` to do so. I first computed M, which is the sum of the pixels for the position of the ball. From there I was able to compute x and y, which are the averages of the x and y positions for the ball. To run this function, I called it in `process_image(image)`.

```
def position_averages(image):
    ball_position = np.argwhere(image)
    M = np.sum(ball_position)
    x = np.sum(ball_position[:,0])/M
    y = np.sum(ball_position[:,1])/M

    return x,y
```

```
(0.3544137218768674, 0.6455862781231326)
(0.38640907477022074, 0.6135909252297792)
(0.4443900837652669, 0.5556099162347331)
(0.5106570583480101, 0.48934294165198994)
(0.5770911339178142, 0.42290886608218586)
(0.6363181947608686, 0.36368180523913135)
(0.6875883094690138, 0.31241169053098616)
```

c. To examine the measuring tape in the images, I first read the original image into the console, adding a grid for better accuracy.

```
ruler = io.imread("drop1.png")
plt.figure()
plt.grid()
plt.imshow(ruler, cmap = "gray")
```



Then, I determined that one inch is equivalent to about 44.4 pixels.

First, I need to find the velocity.

$$v = \frac{\Delta x}{\Delta t}$$

$$v = \frac{5}{5/24}$$

$v = 24\ in/s$, in meters $v = .6096\ m/s$

Then, I can find the acceleration due to gravity.

$$g = \frac{\Delta v}{\Delta t}$$

$$g = \frac{.6096}{5/24}$$

$$g = 2.92608\ m/s^2$$

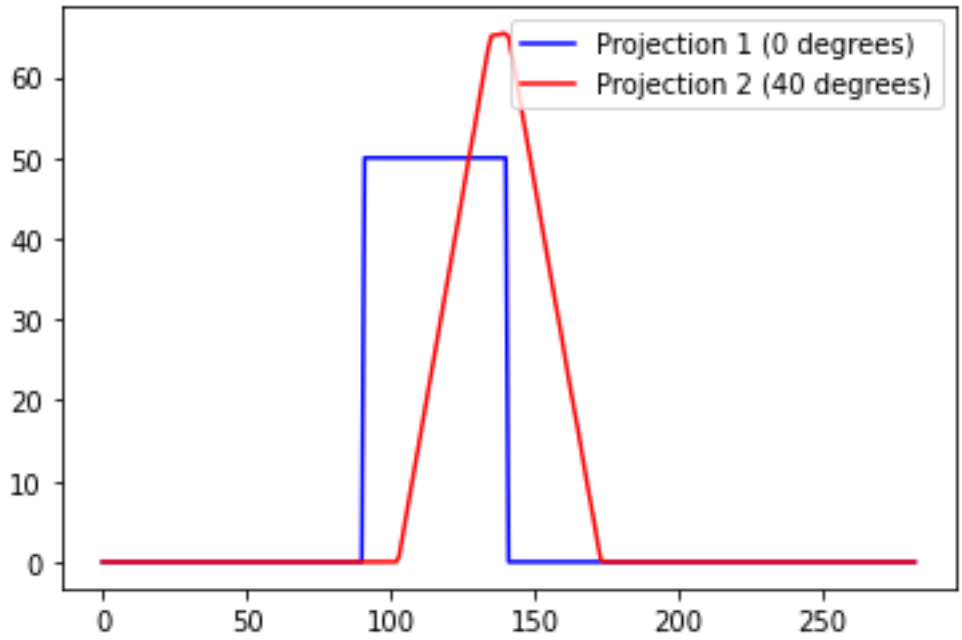
d. The ball does continue to accelerate in the last two images

2.

a. To plot the first and second columns of the Radon transformation, I created a figure with a legend to ensure that both plots are on shown on the figure.

```
plt.figure()
plt.plot(sinogram[:,0], color="blue", label="Projection 1 (0 degrees)")
plt.plot(sinogram[:,1], color="red", label="Projection 2 (40 degrees)")
plt.legend()
plt.show()
```

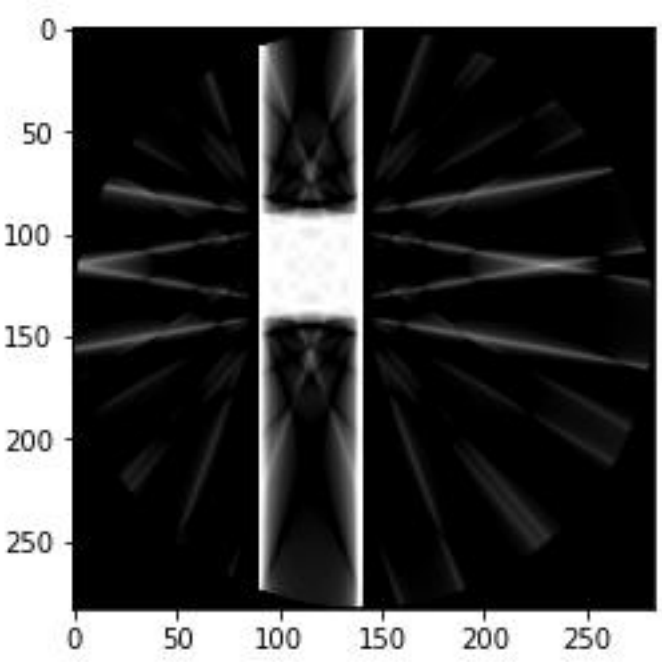
Here is the result.



These projections do make sense; projection one is strictly the original image going through a Radon transformation, which I can see reflected in the figure. Projection two is taken from lines that form 40-degree angles with respect to the horizontal; I can see that reflected in the figure because looking closely at the peak of projection two, there is a straight line, which shows that the original image was a square but was skewed in some way.

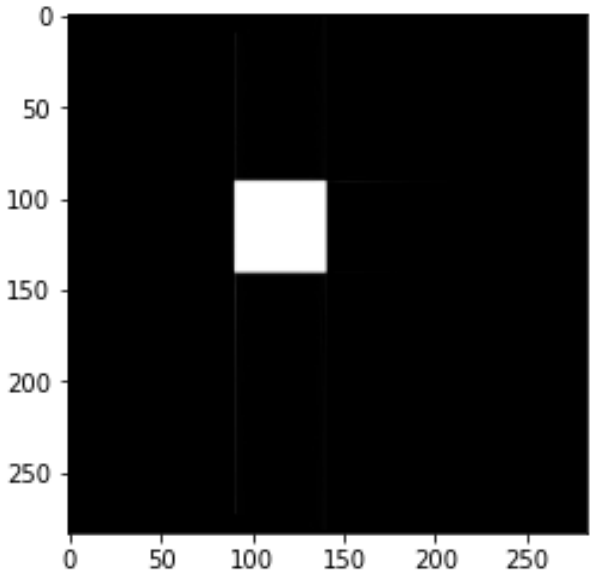
b. I added the given code to the program to produce the following image, which is the inverse transformation of the original image.

```
inv_r=iradon(sinogram ,theta=theta)
plt.imshow(inv_r ,cmap=plt.cm.Greys_r ,vmin=0,vmax=1)
plt.show()
```



The above image depicts the inverse transformation, where you can see the projections of the various angles. Then I changed the angles to produce the following image.

```
theta=np.linspace(0.,360.,360,endpoint=True)
```



The image above is worse than the old image, as it is the same as the original image. The image is worse because it does not show any projections/shadows.

c. To get the transformation below, I used the original image as a white circle in a black image.

```
image_circle = np.zeros((200,200))
center = (100,100)
radius = 25

for i in range(image_circle.shape[0]):
    for j in range(image_circle.shape[1]):
        if np.sqrt((i - center[0])**2 + (j - center[1])**2) < radius:
            image_circle[i, j] = 1.0

plt.title("Original Image with White Circle")
plt.imshow(image_circle, cmap=plt.cm.Greys_r, vmin=0, vmax=1)
plt.show()


# Construct a range of ten angles
theta=np.linspace(0.,360.,360,endpoint=True)

# Carry out the radon tranform
sinogram=radon(image_circle,theta=theta,circle=False)


plt.figure()
plt.plot(sinogram[:,0], color="blue", label="Projection 1 (0 degrees)")
plt.plot(sinogram[:,1], color="red", label="Projection 2 (40 degrees)")
plt.legend()
plt.show()
```

Here are the results.