Caitlyn Jones

Prof. Yangyang Wang

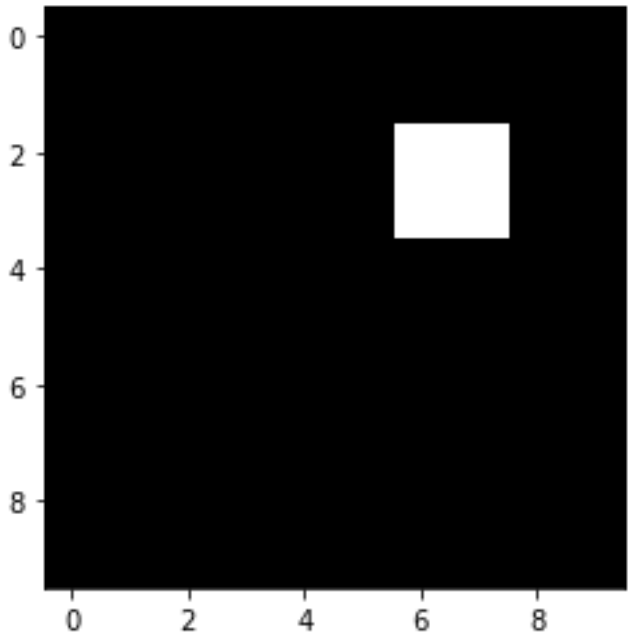
MATH 40A: Intro to Applied Math

18 October 2024

Lab 3

1. This is my sketch on paper based on the code in test_image.py.

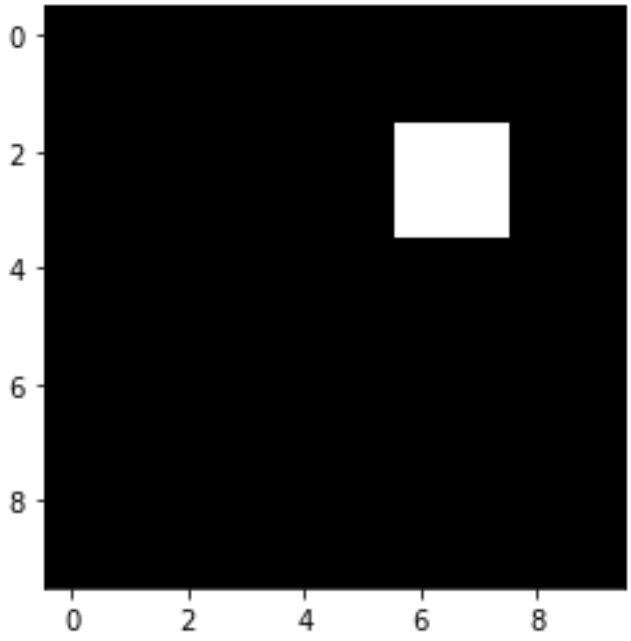
This is the output of running test_image.py.



2. To create function `makesquare()`, I needed to input four variables: the original image, the upper left corner of the square, the size of the square, and the shade of the square. Then, I needed to create the square using the upper left corner as the starting point of the new square with the different shade.

```
def makesquare(image, upper_left, size, shade):
    row, col = upper_left
    image[row:row+size, col:col+size] = shade

    return image

ax = np.zeros((10,10))
ax_square = makesquare(ax,(2,6),2,1.0)
plt.imshow(ax_square, cmap = plt.cm.gray , vmin = 0.0 , vmax = 1.0)
plt.show ()
```

My output image was the same as question 1, as expected.



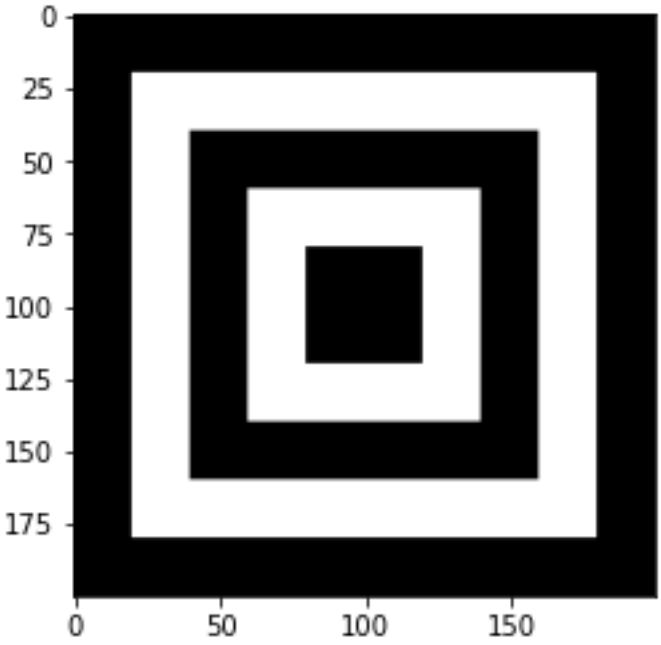
3. To make the leftmost image, I first created the 200 x 200 image with a black background.

```
ax = np.zeros((200,200))
```

Then, I made four alternating white and black squares, up to scale.

```
ax_square = makesquare(ax,(20,20),160,1.0)
ax_square = makesquare(ax,(40,40),120,0.0)
ax_square = makesquare(ax,(60,60),80,1.0)
ax_square = makesquare(ax,(80,80),40,0.0)


plt.imshow(ax_square, cmap = plt.cm.gray , vmin = 0.0 , vmax = 1.0)
plt.show ()
```

Here is the desired output.

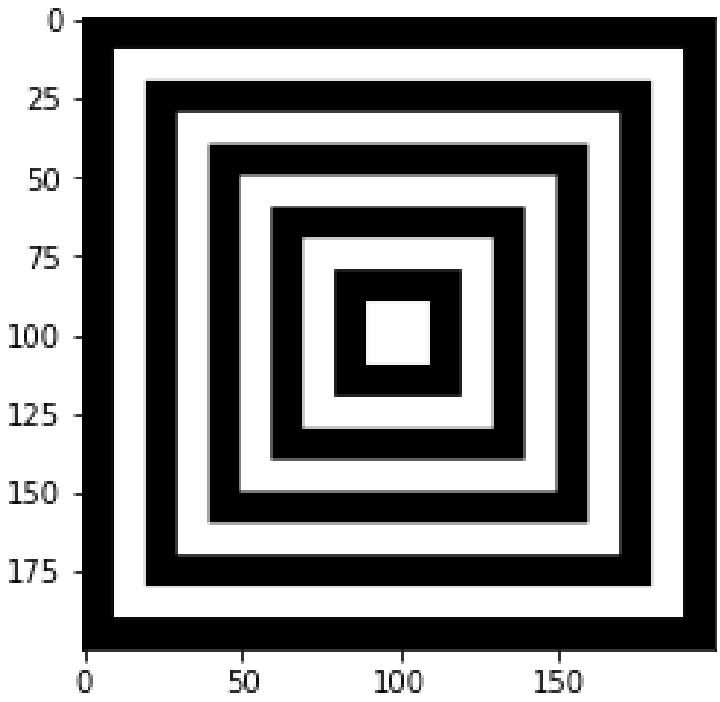


To make the rightmost image, I created a function `makesquare_large(image, x, y)`, where x is tracking the coordinates and y is tracking the size of the square.

```
def makesquare_large(image, x, y):
    for i in range (1,10):
        if (i%2 == 1):
            image = makesquare(image,(x+10, x+10),y-20, 1.0)
        else:
            image = makesquare(image, (x+10, x+10), y-20, 0.0)

        x += 10
        y -= 20

    return image
```

Here is the desired output.

```
ax = np.zeros((200,200))
ax_square = makesquare_large(ax,0,200)

plt.imshow(ax_square, cmap = plt.cm.gray , vmin = 0.0 , vmax = 1.0)
plt.show ()
```



4. The algorithm fails because in images with small horizontal lines, the neighboring pixels will also be affected by the noise. Instead, I did the following.

```
for j in range(1,y-1):
    for i in range(1, x-1):
        l = [a[j-1,i-1],a[j-1,i],a[j-1,i+1],
             a[j,i-1],a[j,i], a[j,i+1],
             a[j+1,i-1],a[j+1,i], a[j+1,i+1]]
        b[j,i] = np.median(l)
```

To get the following denoised image.