CS168 Miniproject 9

Henry Lin, Kaylee Bement

1. A. k/n = 0.21

B.

# Part B

n = 1200

x = load\_text\_file\_as\_array(wonderland\_tree)

np.random.seed(42)

A = np.random.normal(size=(n, n))

def compressive\_sensing(r):

x\_r = cvx.Variable(n)

b = np.dot(A[0:r], x)

objective = cvx.Minimize(cvx.norm(x\_r, 1))

constraints = [b == A[0:r] \* x\_r, x\_r >= 0, x\_r <= 1]

prob = cvx.Problem(objective, constraints)

prob.solve(cvx.ECOS)

return x\_r.value

print("Part B: ", np.allclose(compressive\_sensing(600), x))

C.

r\* = 462 with random seed 42

# Part C

def r\_value\_binary\_search():

possible\_r\_values = [x for x in range(1, 1200)]

valid\_r\_values = []

target = .001

lower = 0

upper = 1199

while lower < upper:

index = lower + (upper - lower) // 2

r = possible\_r\_values[index]

print(r)

x\_r = compressive\_sensing(r)

val = np.linalg.norm(x - x\_r, 1)

print(val)

if val < target:

valid\_r\_values.append(r)

upper = index # see if there's a smaller r

else:

if lower == index:

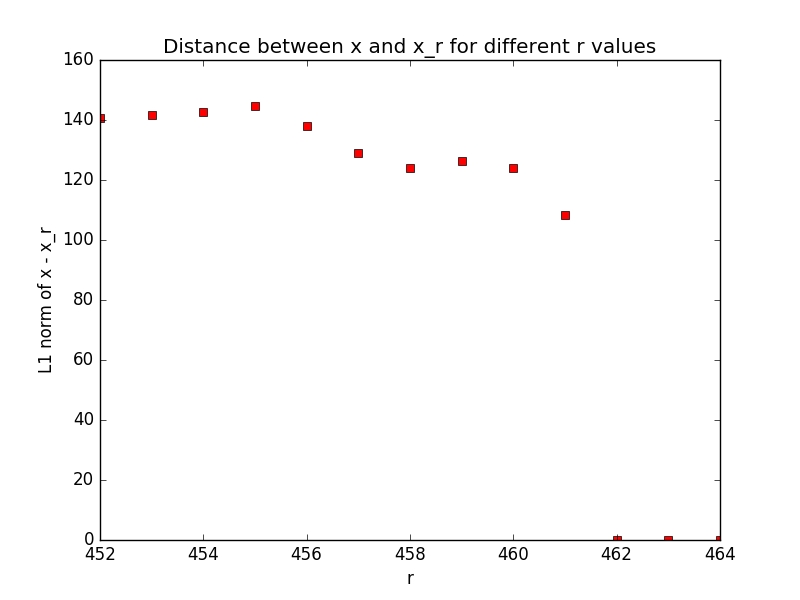
break

lower = index

return min(valid\_r\_values)

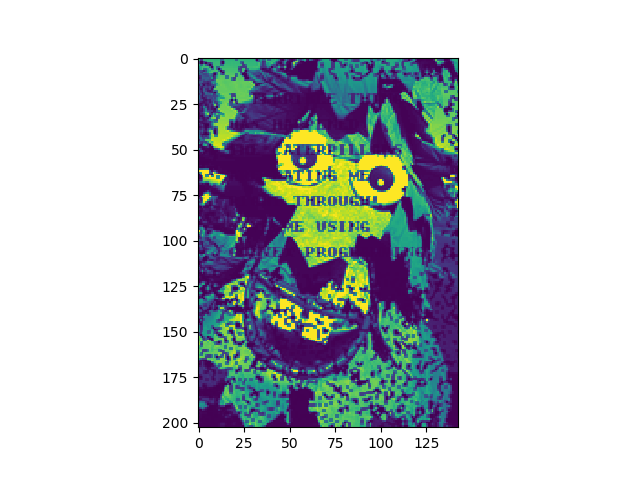
r\_star = r\_value\_binary\_search()

print("Part C: ", r\_star)

D. 

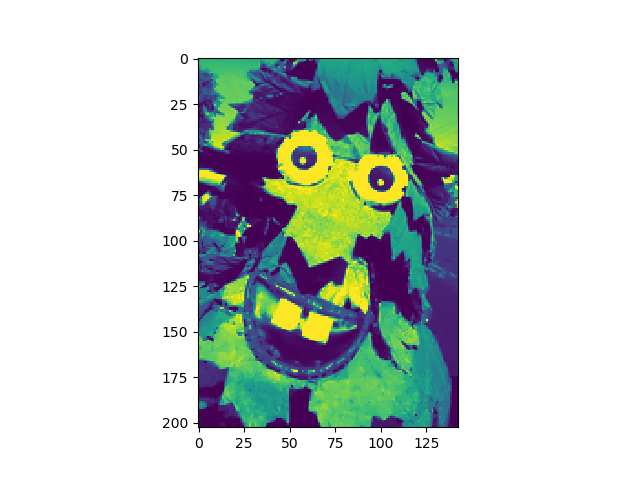
1. A.

B.



The naïve reconstruction method of taking the average of the nearest 4 pixels is a good baseline but it has several problems. If the corrupted image has significant, entire chunks missing, then the algorithm will not be able to accurately determine what is in those chunks because there are no neighboring “good” pixels for the inside part of the chunk. Furthermore, it might run into problems when dealing with small details that are a single or few pixels wide. It will not be able to recover those details well.

C.



We can see that this recovered image is much higher quality than that of the naïve reconstruction. There are no longer many spots of black pixels that resulted from the lack of neighbors, and the image looks much more like the original.

D.