# **Homework 5**

AMATH 482

Christian Diangco

## **Abstract**

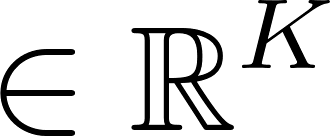
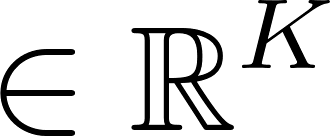
This paper covers the theory and use of discrete cosine transform, image compression, and sparse image recovery.The process is discussed in Sec. 3. Algorithm Implementation and Development, after which results and visualizations are displayed in Sec. 4. Computational Results.

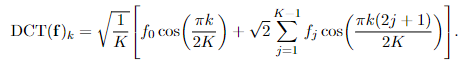
## **Sec. 1. Introduction and Overview**

In this assignment we are tasked with recovering an image from limited observations of its pixels. Below is a portion of Rene Magritte’s “The Son of Man” along with its corrupted variant. Our goal is to recover the original image from the corrupted version.

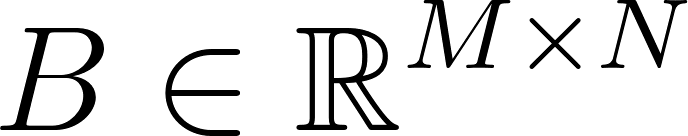
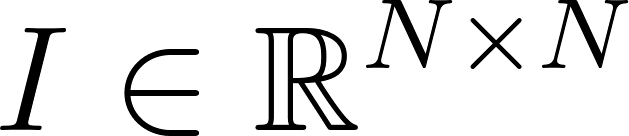
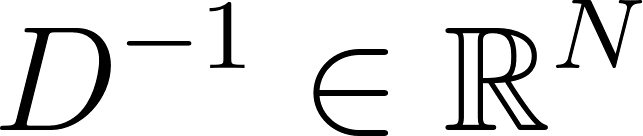
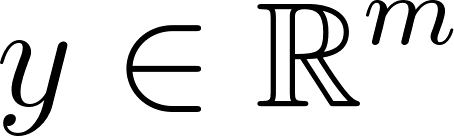
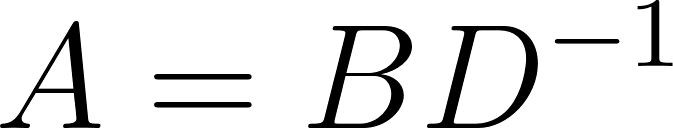


## **Sec. 2. Theoretical Background**

For this assignment we utilize the discrete cosine transform (DCT). Given a discrete signal [****](https://www.codecogs.com/eqnedit.php?latex=%5Cin%20%5Cmathbb%7BR%7D%5EK#0), we define its DCT as DCT(**)** [****](https://www.codecogs.com/eqnedit.php?latex=%5Cin%20%5Cmathbb%7BR%7D%5EK#0)where



This transform is comparable to taking the real part of the FFT of i.e. writing the signal as a sum of cosines. The inverse DCT (iDCT) transform is used to reconstruct the signal from DCT(. The two-dimensional DCT is defined analogously to the two-dimensional FFT by successively applying the one-dimensional DCT to the rows and columns of a 2D image.

In sparse signal and image recovery, it ultimately boils down to a supervised learning/regression problem. Let [](https://www.codecogs.com/eqnedit.php?latex=B%20%5Cin%20%5Cmathbb%7BR%7D%20%5E%20%7BM%20%5Ctimes%20N%7D#0), with M random rows from the identity matrix [](https://www.codecogs.com/eqnedit.php?latex=I%20%5Cin%20%5Cmathbb%7BR%7D%5E%7BN%20%5Ctimes%20N%7D#0) and [](https://www.codecogs.com/eqnedit.php?latex=D%5E%7B-1%7D%20%5Cin%20%5Cmathbb%7BR%7D%5EN#0) be the iDCT matrix. [](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=y%20%5Cin%20%5Cmathbb%7BR%7D%5Em#0) is the result of applying to the vector of a flattened image matrix . We define matrix [](https://www.codecogs.com/eqnedit.php?latex=A%3DBD%5E%7B-1%7D#0) and solve this optimization problem



The minimizer , is the DCT vector of an image that should resemble the original image.

## **Sec. 3. Algorithm Implementation and Development**

I used these functions from the starter code to create the DCT and iDCT matrices:

# construct DCT matrix

def construct\_DCT\_Mat( Nx, Ny ):

# input : Nx number of columns of image

# Ny number of rows of image

# output: D DCT matrix mapping image.flatten() to

# DCT(image).flatten()

Dx = spfft.dct(np.eye(Nx), axis =0, norm='ortho')

Dy = spfft.dct(np.eye(Ny), axis = 0, norm='ortho')

D = np.kron(Dy, Dx)

return D

def construct\_iDCT\_Mat( Nx, Ny ):

# input : Nx number of columns of image

# Ny number of rows of image

# output: iD iDCT matrix mapping DCT(image).flatten() to

# image.flatten()

Dx = spfft.idct(np.eye(Nx), axis =0, norm='ortho')

Dy = spfft.idct(np.eye(Ny), axis = 0, norm = 'ortho')

D = np.kron(Dy, Dx)

return D

I created the following function to threshold the DCT of the image:

# Function for thresholding DCT\_F

def threshold(matrix, p):

DCT\_F\_thresh = copy of matrix

# Threshold at percent 1 - p

thresh = (100 - p)th percentile of matrix

for i in range(len(matrix)):

if abs(matrix[i]) < thresh:

DCT\_F\_thresh[i] = 0

# Compute vec(F) of DCT\_F\_Thresh using DCT\_inv

vec\_F = DCT\_inv \* DCT\_F\_thresh

# Reshape vec(F) to get image

img\_threshold = reshaped version of vec(F) to match image dimensions

return img\_threshold

To recover the sparse images, I defined the following function to solve for the minimizer , which makes use of the CVX library:

def solve\_x\_star(r):

N = 2173

M = round(r \* N)

# Construct a measurement matrix B by randomly selecting M rows of the identity matrix I

B = M random rows of I

# Generate vector of measurements y by applying B to vec(F)

y = B \* vec(F)

# Define matrix A = BD^-1

A = B \* DCT\_inv

x = cvx.Variable(N)

obj = cvx.Minimize(cvx.norm(x, 1))

constraints = [(A @ x) == y]

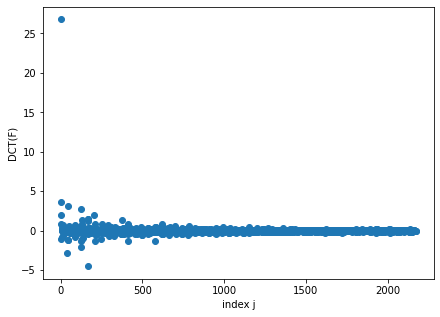
prob = cvx.Problem(obj, constraints)

prob.solve(verbose=True, solver="CVXOPT", max\_iter=1000, reltol=1e-2, featol=1e-2)

x\_star = x.value

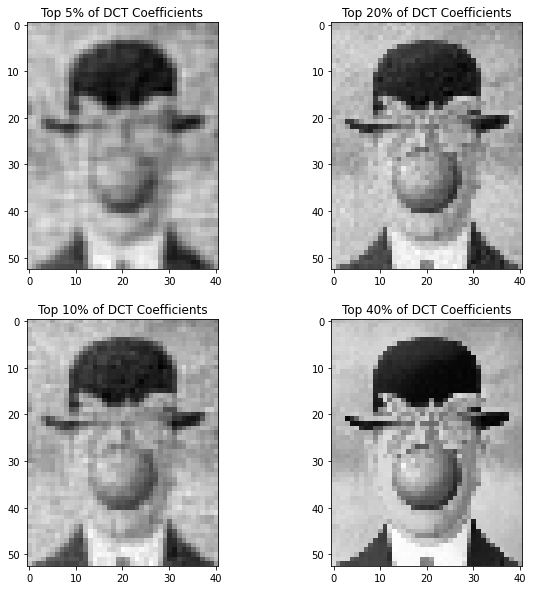
return x\_star

## **Sec. 4. Computational Results**

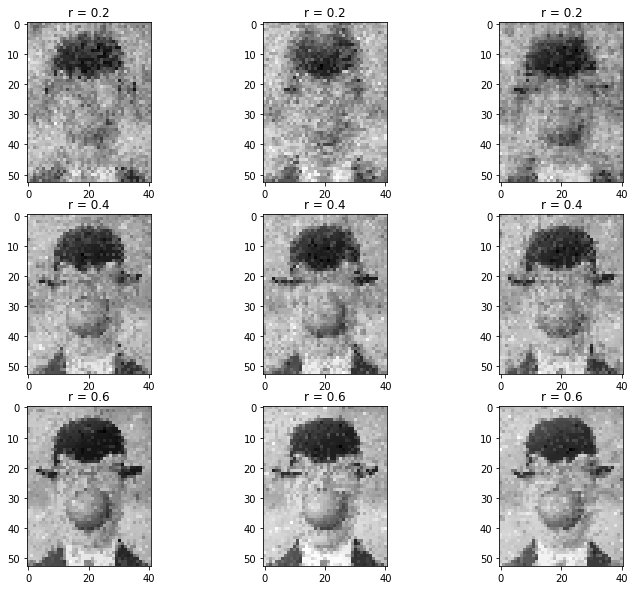


(Figure 1) Plot of DCT(F).

From Figure 1, there is only one large coefficient > 25, with the rest ranging between -5 and 5.



(Figure 2) Reconstructed versions of the image after thresholding its DCT



(Figure 3) Recovered images using different values for . From top to bottom are the recovered images when r = 0.2, r = 0.4, and r = 0.6. Each column represents a separate experiment.

As I expected, image accuracy increases with the value of M. We can see that for r = 0.2, the three images vary quite significantly.



(Figure 4) The reconstructed unknown image.

Nyan cat!

## **Sec. 5. Summary and Conclusions**

We utilized DCT and Python’s CVX package to conduct sparse image recovery. We found that the recovered image accuracy scaled with the value chosen for M.

## **Acknowledgments**

I would like to acknowledge the students in the AMATH 582/482 Discord who allowed me to have fruitful discussion about this assignment.

## **References**

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature

585, 357–362 (2020). <https://doi.org/10.1038/s41586-020-2649-2>

J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in Computing in Science &

Engineering, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Steven Diamond, Stephen Boyd, “A Python-embedded modeling language for convex optimization,” in Journal of Machine Learning Research, vol. 17, no. 83, 2016