[CSED490F] Signal Processing

HW4 - Graph Signal Reconstruction

POSTECH Dept. of CSE

20210716 Dae-hyeon Choi

Abstract

Graph Signal Reconstruction is the process of estimating a signal defined on a graph from a limited set of observed nodes. Here, the graph shows how nodes or vertices are connected by edges, each node represents a data point or variable, and the edge represents the relationship or connection between these variables.

In this report, we will experiment with a process of actually taking only a fraction of the nodes (e.x. 100, 500, 1000) out of a three-dimensional graph with 3400 nodes and restoring them to a complete signal. This report will not cover this topic with deep theorems, and if readers are curious about the detailed principles, they can refer to the following paper:

https://pages.cs.wisc.edu/~wonhwa/publication/eccv2016_wonhwa.pdf

Design

There may be a method of processing 3D data at once (i.e. with one Tensor), but for convenience of implementation, all functions 2343 implemented to be processable for 1D signals. This can be done once for each axis to change into a complete signal in three dimensions be reconstructed.

In particular, some formulas are required in the process of establishing a sampling matrix and restoring g based on it. The g that we want to obtain has the following objective function:

$$g^* = arg \min_{g \in \mathbb{R}^N} ||(Mg - y)||_2^2 + \gamma g^T L g$$

... (1)

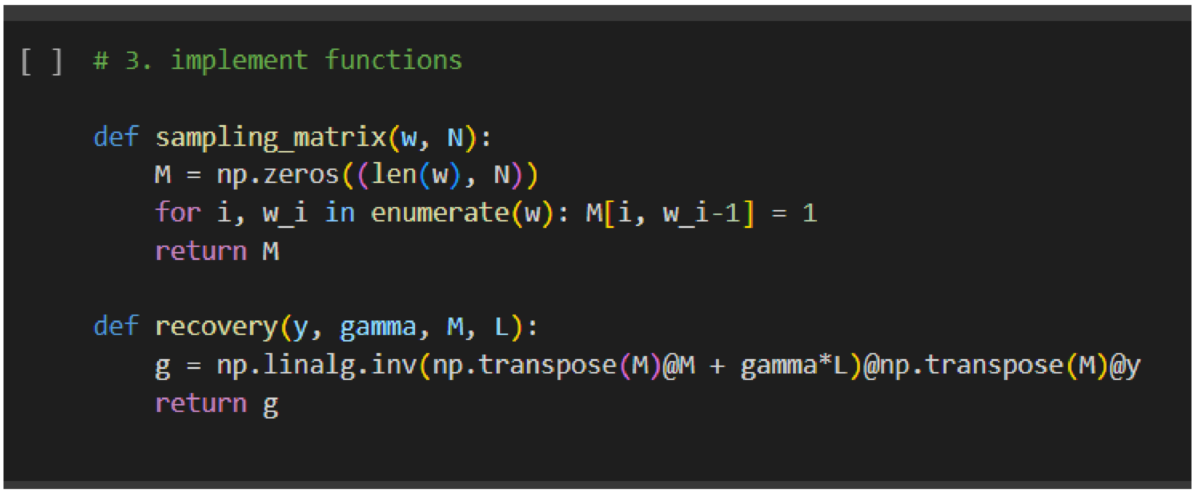
Simply to optimize the first term, Mg = y is a situation, which would not be desirable. Therefore, it prevents overfitting by making some regularization for gamma and graph Laplacian L.

$$g = (M^T M + \gamma L)^{-1} M^T y \qquad \text{... (2)}$$

Since it is a convex problem clearly, it can be seen that the g we want to obtain is as follows by solving it with an equation which is earned by partial derivative of (1).

To link Matlab's matrix data with Python's numpy, we imported some libraries such as 'scipy.io' and used a function 'matload'. 'A.mat' file contains various information. If this is taken as a matload function, it is a data type that combines several structures in which key is the name of the vector and value is the list of the vectors, of which the information we want is w1, w2, w3, y1, y2, and y3. The reason why there are three w and y is that the data we are trying to reconstruct are 3D data. These data will be restored to gx, gy, gz for the x, y, and z axes, respectively, to obtain meaningful shapes in three dimensions.

Implementation

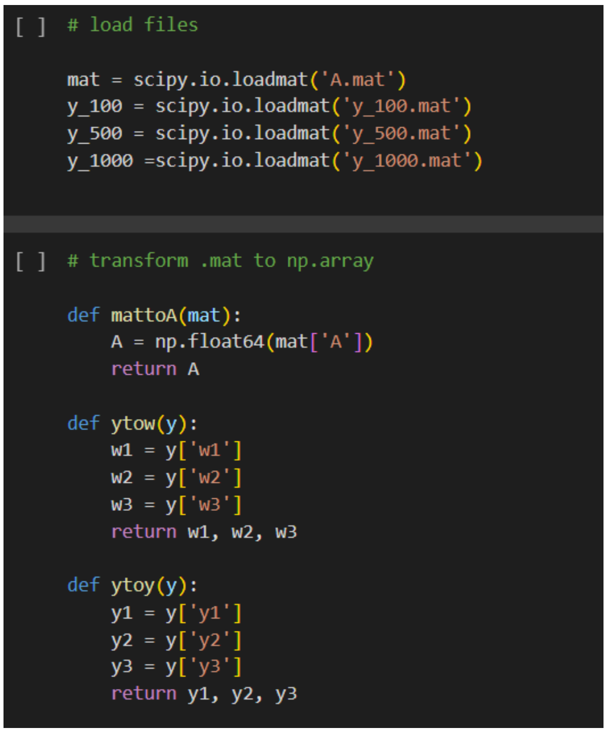


```
[ ]  # 3. implement functions

     def sampling_matrix(w, N):
         M = np.zeros((len(w), N))
         for i, w_i in enumerate(w): M[i, w_i-1] = 1
         return M

     def recovery(y, gamma, M, L):
         g = np.linalg.inv(np.transpose(M)@M + gamma*L)@np.transpose(M)@y
         return g
```

sampling_matrix(w, N) : A function of building a sampling matrix M for a given w. In this case, M is an (mxN) matrix that satisfies Mf = y for y containing sequential values for the original signal f and sampling node by definition.

recovery(y, gamma, M, L): a function that takes y, gamma, M, and L as parameters and returns the restored signal g. At this time, y is a vector with sequential values for the sampled node, gamma is a hyperparameter for regularization, M is a sampling matrix obtained earlier, and L is a graph Laplacian. We can obtain optimum g by calculating the formula (2) with numpy's method.

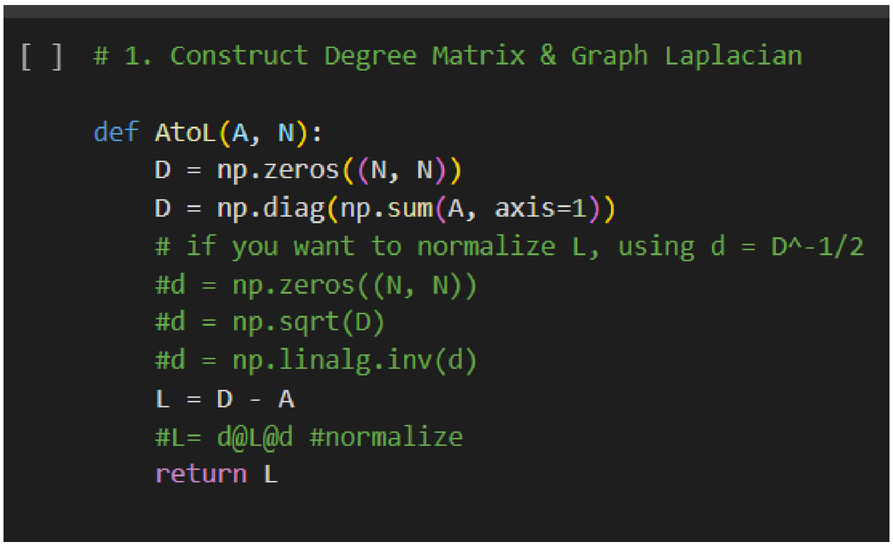
Additional Implement



```
[ ]  # load files

     mat = scipy.io.loadmat('A.mat')
     y_100 = scipy.io.loadmat('y_100.mat')
     y_500 = scipy.io.loadmat('y_500.mat')
     y_1000 =scipy.io.loadmat('y_1000.mat')
```

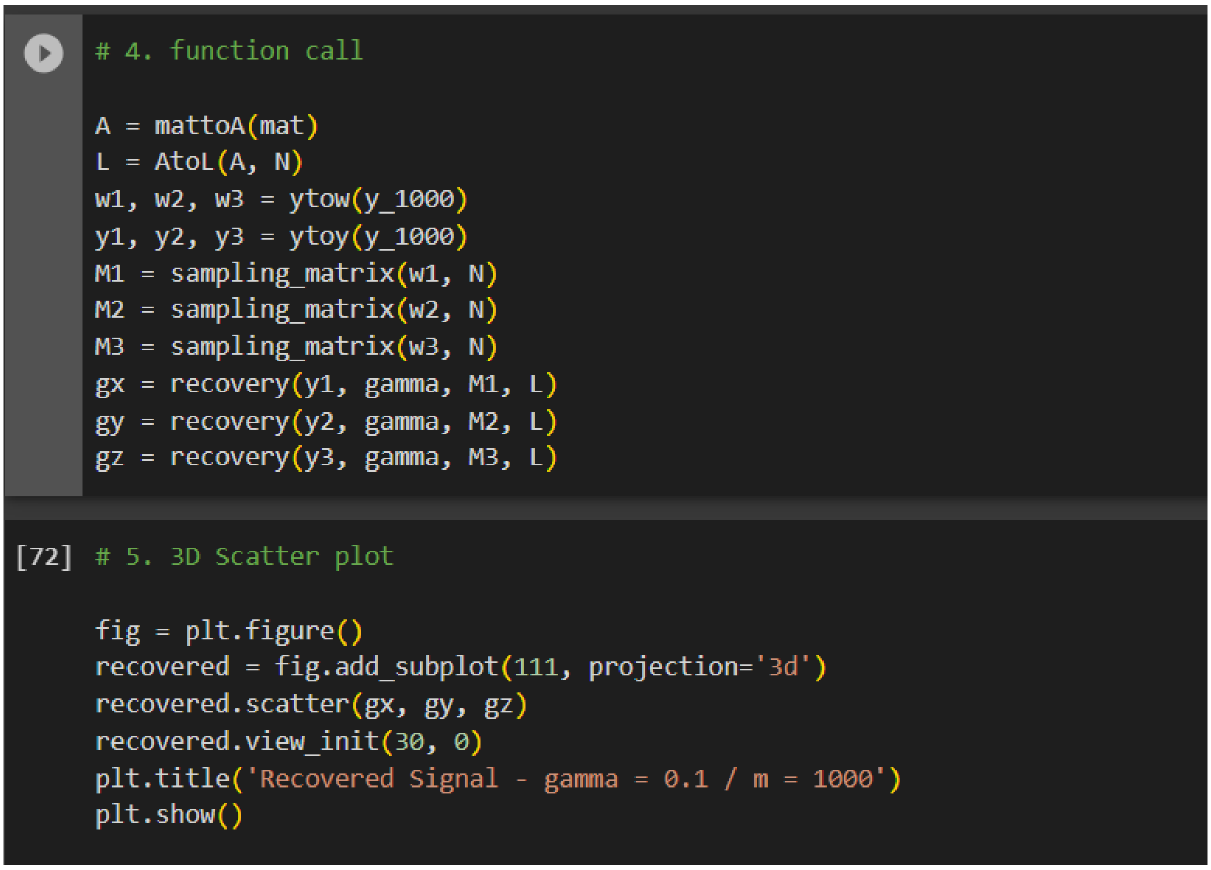
```
[ ]  # transform .mat to np.array

     def mattoA(mat):
         A = np.float64(mat['A'])
         return A

     def ytow(y):
         w1 = y['w1']
         w2 = y['w2']
         w3 = y['w3']
         return w1, w2, w3

     def ytoy(y):
         y1 = y['y1']
         y2 = y['y2']
         y3 = y['y3']
         return y1, y2, y3
```

ytow(y), ytoy(y): It is a function that cuts y and returns a given file y.mat to w1, w2, w3, or y1, y2, y3, respectively.



```
[ ]  # 1. Construct Degree Matrix & Graph Laplacian

     def AtoL(A, N):
         D = np.zeros((N, N))
         D = np.diag(np.sum(A, axis=1))
         # if you want to normalize L, using d = D^-1/2
         #d = np.zeros((N, N))
         #d = np.sqrt(D)
         #d = np.linalg.inv(d)
         L = D - A
         #L= d@L@d #normalize
         return L
```

AtoL(A): It is a function that takes the adjacent matrix A as a parameter and returns the graph Laplacian. In this case, the graph Laplacian is defined as L = D-A, where D is a degree matrix for the adjacent matrix A. Alternatively, it is normalized by multiplying D^-1/2 before and after the obtained L, and we also implemented it and left as an option.



```
# 4. function call

A = mattoA(mat)
L = AtoL(A, N)
w1, w2, w3 = ytow(y_1000)
y1, y2, y3 = ytoy(y_1000)
M1 = sampling_matrix(w1, N)
M2 = sampling_matrix(w2, N)
M3 = sampling_matrix(w3, N)
gx = recovery(y1, gamma, M1, L)
gy = recovery(y2, gamma, M2, L)
gz = recovery(y3, gamma, M3, L)
```

```
[72] # 5. 3D Scatter plot

     fig = plt.figure()
     recovered = fig.add_subplot(111, projection='3d')
     recovered.scatter(gx, gy, gz)
     recovered.view_init(30, 0)
     plt.title('Recovered Signal - gamma = 0.1 / m = 1000')
     plt.show()
```

Function call & 3D Scatter Plotting

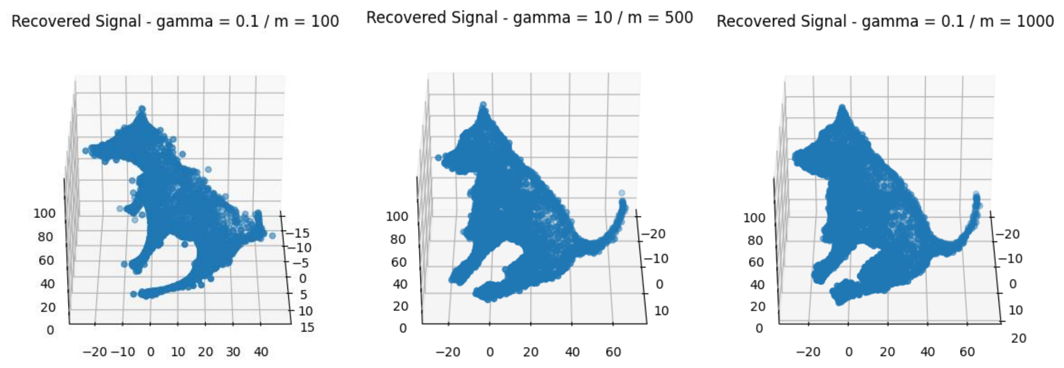
Experiment

Recover the 3-dimensional signal and show a 3D scatter plot for each case, i.e., for 100, 1000, and 1500 nodes. What do you see? Discuss your observations.

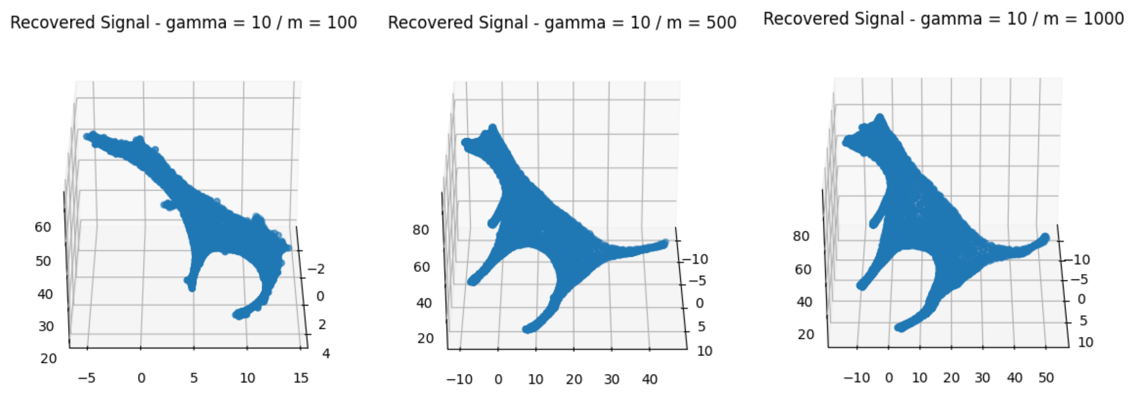
Overall, we can obtain images that are supposed to be a dog, and it can be confirmed that the greater number of sampling nodes, the more clearly restored.

Change γ = 10 and try the same experiment. How do the results change? Discuss your observations.

It is difficult to say that the smaller gamma is better or the larger one is better. However, since gamma determines the intensity of regularizing the size of g, it can be seen that the recovery when gamma = 10 is generally a slim image in which nodes are close to zero.



<plot about gamma = 0.1 / m = 100, 500, 1000>



<plot about gamma = 10 / m = 100, 500, 1000>

Discussion

There were two options for regularizing graph Laplacian and not doing it, which did not perform better when normalizing it.

We implemented the sampling_matrix function by approaching a continuous row at first, and we modified it with a memory-friendly strategy, then the compute time was significantly improved.

In the sampling_matrix function, when y = y_500 was inserted, an out of range error occurred. We checked the statement that a node whose number is 3400 was out of index and found that the range of w was 1-3400, not 0-3399. Since then, the function has been modified to reflect this, so it can be seen that the plot is well done in the desired shape.

When plotting our result, the desired three-dimensional image did not come out at first, but when the angle was set as a view_init function, then the desired image came out.

Reference

1. Adaptive Signal Recovery on Graphs via Harmonic Analysis for Experimental Design in Neuroimaging. W.Kim. et al
2. Construction of Wavelets in Euclidean / non-Euclidean Spaces. W.Kim.