First, I had the following questions:

1. In your sdp code, when you define matrix B, you stacked the babies of matrix B using the following Lines 75-77:

```
75  for i=1:length(b_ind)
76      dB(:,:,i) = [zeros(n_sample,n_sample) ei_s(:,i); ei_s(i,:) 0]; % eq. (8), where dB is a stacked matrix
77  end
78
79  for i=1:length(b_ind)
80      dB(:,:,i) = [zeros(n_sample,n_sample) ei_s(:,b_ind(i)); ei_s(b_ind(i),:) 0]; % eq. (8), where dB is a stacked
81  end
```

which means you ignored the order of the known labels (entries). If I understand correctly, we should use the actual order of the known labels (as Lines 79-81 as shown above) to stack the babies of matrix B, right?

2. The variables of the SDP dual problem are vectors y (dimension N+1) and z (dimension M). Because the signs of the entries of y and z are not known, if we would like to solve all of the variables in Eq. (11) via GDA-based LP at once, we come across the '2^(N+1+M) possible cases' problem, again. Do you think it is reasonable to solve y and z separately somehow?

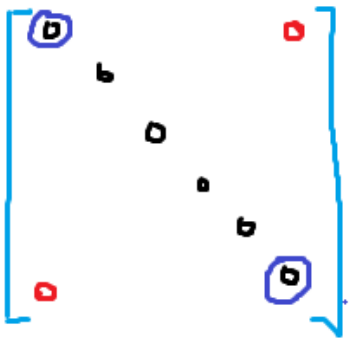
Based on Question 2, I run the following three experiments:

1. RUN_ME_LP_1offdia_2dia.m:
   This code first fixes the initial z's and solves y's without any LP tools; if z's are fixed, then we can just take the minimum values of the inequality:

$$A_{ii} - \sum_{j \neq i} |s_i A_{ij} / s_j| \geq 0,$$

+ P (small)

   Where the minimum values of A_{ii}'s are the y's that we want.
   When we have the updated y's, we fix the signs of y's during the following stages that update y's and z's.
   Now, each time, the code updates one off-diagonal entry (the red circle('s) below) of the targeted PSD matrix and its corresponding two diagonal entries (the two black circles highlighted by blue circles).



$$\min_{\mathbf{y},\mathbf{z}} \ \mathbf{1}_{N\times 1}^{\top}\mathbf{y} + \mathbf{b}^{\top}\mathbf{z} \qquad (11)$$

$$\text{s.t.} \ \sum_{i=1}^{N+1} y_i \mathbf{A}_i + \sum_{i \in \mathcal{F}} z_{o(i)} \mathbf{B}_i - \mathbf{L} \succeq 0$$

In this stage, I set the sign of the off-diagonal entry be positive, run the LP, and then set it to negative, and run the LP again, and then I choose the one that has the smaller objective. However, the resulting z's are always 0's.

```
==============
GDA LP started.
GDA LP main iteration 1 | current obj: 9 | mineig: 0.073229
GDA LP main iteration 2 | current obj: 4.1583 | mineig: 1e-05
=================
GDA LP converged.
GDA LP solutions:

y =

         0.918685026484647
        -0.0734117917658622
         0.890743574752523
        -0.0731948382688598
          0.92262711646062
        -0.0728201888820642
          0.91619081571789
        -0.0736994953982498
         0.876430363029343
        -0.0732252882064667
                    1e-05


z =

     0
     0
     0
     0
     0
```

2. RUN_ME_no_LP1.m:
   This code solves y's and z's separately. First, it does the same thing as RUN_ME_LP_1offdia_2dia.m, where it updates all y's at once based on the linear inequality. Next, it fixes all y's, and updates z's one by one, again, using the linear inequality. This code does not use any LP's to solve y's or z's.
   However, the problem is that: if we update the off-diagonal (z's) one at a time, then we always end up with a negative z('s), since the objective is [2 2 … 2].*[z_1 z_2 … z_M].
   And we can see from the following result that the solutions of z's are close to zero's:

```
==============
GDA no LP started.
GDA no LP main iteration 1 | current obj: 9 | mineig: 0.073229
GDA no LP main iteration 2 | current obj: 8.991 | mineig: 1e-05
GDA no LP main iteration 3 | current obj: 4.3825 | mineig: 1e-05
GDA no LP main iteration 4 | current obj: 4.2297 | mineig: 1e-05
================
GDA no LP converged.
GDA no LP solutions:

y =

         0.926273068402433
        -0.0743385868203551
         0.910163899024232
        -0.0741291523804646
         0.927073481946437
        -0.0733181883111079
         0.926511444033087
        -0.0758630342620816
         0.901750374836481
        -0.0732294037747238
        0.00885634695737906


z =

     -3.87666924434846e-07
     -8.93928800573178e-06
     -1.22468016899351e-07
     -8.52562483125347e-07
     -1.56860061633269e-05
```

3. RUN_ME_no_LP2.m:
   This code is very similar to RUN_ME_no_LP1.m, except that it updates the scalars after all z's are updated.
   It has similar results as RUN_ME_no_LP1.m, However, again, we have the same problem as RUN_ME_no_LP1.m.

We can set up a Wechat/Zoom discussion sometime this or next week whenever you are available.