We have:

\[
G = (V = [n], E) \\
A_G = (A_{ij})_{i,j \leq n} \text{ is adjacency matrix}
\]

We want to embed the graph in the r-dimensional unit sphere to minimize the 'elastic energy':

\[
\Rightarrow \text{ want to find } \sigma_i, \ldots, \sigma_n \text{ that solve:} \\
\max_{\sigma} F(\sigma) = \sum_{i,j=1}^{n} A_{ij} \langle \sigma_i, \sigma_j \rangle - \frac{2}{n} \left\| \sum_{i=1}^{n} \sigma_i \right\|_2^2 \\
\text{s.t. } \sigma_i \in \mathbb{R}^r, \left\| \sigma_i \right\|_2 = 1 \quad \forall \ i \in [n]
\]

\[
F(\sigma) = \sum_{i,j=1}^{n} A_{ij} \langle \sigma_i, \sigma_j \rangle - \frac{2}{n} \left\| \sum_{i=1}^{n} \sigma_i \right\|_2^2 \\
= \sum_{i,j=1}^{n} \left( A_{ij} - \frac{2}{n} \right) \langle \sigma_i, \sigma_j \rangle
\]

For \( k=2 \) the algorithm is:

1. Input: \( G = (V = [n], E) \) \\
2. \( T = \# \text{ of sweeps} \)
3. Initialize \( \sigma_i \) uniformly \( \mathcal{U} \) random on \( S^{r-1} \)
4. For \( t \in \{1, \ldots, nT\} \)
5. \( i(t) \sim \text{Unif}([n]) \)
6. Let \( \sigma_{i(\tau)}^{\text{new}} = \arg\max_{i \in [n]} F(\sigma) \)
7. Set \( \sigma_{i(t)} = \sigma_{i(\tau)}^{\text{new}} \)
8. Regarding \( \sigma \in \mathbb{R}^{n \times n} \), compute its top right singular vector \( v_i = v_i(\sigma) \in \mathbb{R}^n \)
9. Return vector \( \hat{\sigma} = (\hat{\sigma}_1, \ldots, \hat{\sigma}_n) \), where \( \hat{\sigma}_i = \text{sign}(v_i, i) \)
solve step 4 outright

\[ F(\sigma) = \sum_{i,j=1}^{n} (A_{ij} - \frac{2}{n}) \sigma_i \sigma_j \]

\[ = \sum_{i,j=1}^{n} M_{ij} \sigma_i \sigma_j \]

\[ i(t) = k \]

\[ \frac{d}{d\sigma_k} F(\sigma_k) = \frac{d}{d\sigma_k} \sum_{i,j=1}^{n} M_{ij} \sigma_i \sigma_j \]

\[ = \frac{d}{d\sigma_k} \sum_{i,j \neq k} M_{ij} \sigma_i \sigma_j + \frac{d}{d\sigma_k} \sum_{j \neq k} M_{kj} \sigma_k \sigma_j \]

\[ = \sum_{j \neq k} M_{kj} \left( \frac{d}{d\sigma_k} \sigma_j \sigma_j \right) + M_{kk} \frac{d}{d\sigma_k} \sigma_k \sigma_k \]

\[ = \sum_{j \neq k} M_{kj} \sigma_j + M_{kk} 2\sigma_k \]

\[ \frac{d}{d\sigma_k} F(\sigma_k) = \sum_{j \neq k} M_{kj} \sigma_j + M_{kk} 2\sigma_k \]

\[ 0 = 2M_{kk} \sigma_k + \sum_{j \neq k} M_{kj} \sigma_j \]

\[ -2M_{kk} \sigma_k = \sum_{j \neq k} M_{kj} \sigma_j \]

\[ \sigma_k = \frac{-1}{2M_{kk}} \sum_{j \neq k} M_{kj} \sigma_j \]

\[ \sigma_k = \frac{-1}{2M_{kk}} C_d \in \mathbb{R}^{r 	imes 1} \]

\[ C = \sigma \text{ without } \sigma_k \in \mathbb{R}^{r \times (n-1)} \]

\[ d = k^m \text{ col of } M \text{ w/ out } k^m \text{ elem} \in \mathbb{R}^{(n-1) \times 1} \]

but we know that \( \|\sigma_i\|_2 = 1 \) \( \forall i \in [n] \)

so instead of \( \frac{-1}{2M_{kk}} \) factor, divide by norm

\[ \sigma_k = \frac{1}{\|C_d\|_2} C_d \in \mathbb{R}^{r \times 1} \]
This now makes the relaxation-based clustering algorithm for $k=2$:

1. initialize $\sigma \in \mathbb{R}^{r \times n}$ uniformly at random on $S^{r-1}$
2. for $t \in \{1, \ldots, nT\}$
3. draw $k = i(t) \sim \text{Unif}([n])$
4. let $\sigma_{i(t)}^{\text{new}} = \sigma_{i(t)}^{\text{new}} = \arg\max_{\sigma_k} F(\sigma) = \frac{1}{\|Cd\|_2}$
5. set $\sigma_k = \sigma_k^{\text{new}}$
6. compute the top right singular vector of $\sigma^\top v_i \in \mathbb{R}^n$
7. return $\hat{\sigma} = \text{sign}(v_i) \in \mathbb{R}^n$

For implementation:

\[ \sum_{j \neq k} M_{jk} o_j \] can be computed either as:

- $Cd$ $C = \sigma$ without $\sigma_k \in \mathbb{R}^{r \times (n-1)}$
  $d = k^{th}$ col of $M$ without $k^{th}$ elem $\in \mathbb{R}^{(n-1) \times 1}$

- $Cd$ $C = \sigma^\top \in \mathbb{R}^{r \times n}$
  $d = k^{th}$ col of $M$ with $k^{th}$ element $= 0 \in \mathbb{R}^{n \times 1}$

This is computationally faster be don't have to make new matrix $C$ each time just make the vector $d$

3 parameters:

- $\gamma$ forces balanced embedding
  - set $\gamma \geq d$ (avg. degree of $G$)
- dimension $r$
  - $r \in \{2, \ldots, 20\}$
- $T = \# \text{ of sweeps}$ (so $nT = \# \text{ of iterations}$)
  - $T \approx 100$
Part A:

As $a$ increases, the overlap increases as well. This is similar to the centered Ag matrix method from HW2 part B. However, here there is a sharp change at $a=25$, while that was a gradual change.

```plaintext
generate 20 random graphs from stochastic block model
n = 5000
b = 10
a = $\{10, 15, 20, 25, 30\}$

$G = (V, E) \sim G(n, \frac{a}{n}, \frac{b}{n})$ w/vertices $V = [n]$
made by:
attribute to each vertex in $[n]$ a true label $\sigma_i \in \{1, -1\}$
uniformly at random
add edges independently w/ probabilities
$P((i, j) \in E | \sigma_i = \sigma_j) = \frac{a}{n}$ if $\sigma_i = \sigma_j$
$P((i, j) \in E | \sigma_i \neq \sigma_j) = \frac{b}{n}$

apply relaxation based clustering

empirical overlap
$Q(\sigma, \hat{\sigma}) = \frac{1}{K-1} \max_{\pi \in \Sigma_k} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left( \prod_{i=1}^{K} \mathbb{1}_{\pi(\hat{\sigma}_i) = \sigma_i} \right) - \frac{1}{K} \right\}$

$\Pi[k]$ is a permutation of the vertex labels $1, \ldots, K$
$\Sigma_k$ is set of all permutations

report overlap for each $(n, a, b)$ over the 20 graphs

We set lambda equal to the avg. degree of $G$, $T$ equal to 100, and $r$ equal to 2.

Mean overlaps for each condition:
[ 0.01068 0.03348 0.2358 0.9058 0.97662 ]

<table>
<thead>
<tr>
<th>a Value</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Overlap</td>
<td>0.01068</td>
<td>0.03348</td>
<td>0.2358</td>
<td>0.9058</td>
<td>0.97662</td>
</tr>
</tbody>
</table>
```

As $a$ increases, the overlap increases as well. This is similar to the centered Ag matrix method from HW2 part B. However, here there is a sharp change at $a=25$, while that was a gradual change.
Part B:

We applied our algorithm to the polblogs dataset from HW2. We set lambda equal to the avg. degree of G, T equal to 100, and varied r.

We got an overlap of .9, which is quite good considering that overlap is between 0 and 1. However, using K-Means with k=2 in the last homework gave us an even better overlap of 0.926.

Our results did not change when we varied r. This is probably because most of the variance in the data is captured in the first 2 dimensions, as seen when we plotted the top two principle components in HW2 and saw great separation. Therefore, relaxing to a higher dimensional unit sphere offers no benefits. If we had a higher dimensional dataset (e.g. with underlying dimension c), we would probably see worse results for r < c and then the results would be equal for all r >= c.

```
[In [1]: run hw5_partb.py
For r = 2, overlap = 0.900327
For r = 3, overlap = 0.900327
For r = 4, overlap = 0.900327
For r = 5, overlap = 0.900327
For r = 6, overlap = 0.900327
For r = 7, overlap = 0.900327
For r = 8, overlap = 0.900327
For r = 9, overlap = 0.900327
For r = 10, overlap = 0.900327
For r = 11, overlap = 0.900327
For r = 12, overlap = 0.900327
For r = 13, overlap = 0.900327
For r = 14, overlap = 0.900327
For r = 15, overlap = 0.900327
For r = 16, overlap = 0.900327
For r = 17, overlap = 0.900327
For r = 18, overlap = 0.900327
For r = 19, overlap = 0.900327
For r = 20, overlap = 0.900327
```
import numpy as np
import itertools
import matplotlib.pyplot as plt
from scipy import linalg

#### CODE FROM HW2 ####

def StochasticBlockModel(a, b, n):
    # true label sigma {0,1} uniformly randomly assigned
    sigma = np.random.randint(2, size=n)
    # make probability threshold P
    s = 2*sigma-1  #want sigma {-1,1}
    s = np.reshape(s, (n, 1))  #make it a column vector
    P = (a+b)/2.0/n + (a-b)/2.0/n * np.dot(s, s.T)
    # add edges independently with probability P
    matrix_rand = np.reshape(np.random.rand(n*n), (n, n))
    rand_edges = (matrix_rand <= P).astype(int)
    rand_edges_triu = np.triu(rand_edges, 1)
    Ag = rand_edges_triu + rand_edges_triu.T  #Ag is symmetric
    return Ag, sigma

def Overlap(sigma_true, sigma_hat, k):
    max_match = -1  #set to impossible value at start
    # iterate through permutations as in HW1, since only identify cluster, not spec. label
    for pi in itertools.permutations(range(k)):
        sigma_hat_permute = [pi[j] for j in sigma_hat]
        equal = (sigma_hat_permute == sigma_true)
        match = np.mean(equal)
        if match > max_match:
            max_match = match
    Q = (1.0*k/(k-1))*((max_match - 1.0/k)
    return Q

#### END OF CODE FROM HW2 ####

def sign_01(v):
    v_signed = np.zeros(len(v))
    for i in range(len(v)):
        if v[i] > 0:
            v_signed[i] = 1
    return v_signed

def RelaxationClustering(Ag, lambda_value, r, T):
    n = Ag.shape[0]
    M = Ag - (lambda_value/n)*np.ones((n, n))
    # 1. initialize sigma uniformly at random on unit sphere of dimension r
    npoints = n
    X = np.random.uniform(0, 1, (npoints, r))
    denom = np.sqrt(np.sum(np.square(X), axis=1))
    tile = np.tile(denom, (r, 1))
    sigma = np.divide(X, tile.T)
    sigma = sigma.T
    # 2. for t in range(n*T):
    # 3.
    k = np.random.randint(n)
    # 4.
    d = M[:, k]
    d[k] = 0
    vec = np.dot(sigma, d)
    sigma_k_new = vec/np.linalg.norm(vec)
    # 5.
    sigma[:, k] = sigma_k_new
    # then we have a rxn matrix sigma from which we calculate the n length vector sigma_hat
    # 6.
    [U, s, Vh] = linalg.svd(sigma)
    v1 = Vh[0, :]
    # 7.
    sigma_hat = sign_01(v1)
    return sigma_hat
PART A: RUN ON STOCHASTIC BLOCK MODEL

```python
k = 2
n = 5000
b = 10
a_values = np.array([10, 15, 20, 25, 30])
num_iter = 20
OverlapMatrix = np.zeros((len(a_values), num_iter))
for i in range(len(a_values)):
    a = a_values[i]
    for j in range(num_iter):
        # generate random graph from stochastic block model
        Ag, sigma_true = StochasticBlockModel(a, b, n)

        # choice of 3 parameters:
        T = 100
        lambda_value = np.mean(np.sum(Ag, axis=0))  # lambda >= avg degree of G
        r = 2

        # apply relaxation based clustering
        sigma_hat = RelaxationClustering(Ag, lambda_value, r, T)

        # calculate empirical overlap
        Q = Overlap(sigma_true, sigma_hat.astype(int), k)
        OverlapMatrix[i, j] = Q
        print('Done with a = %d, iteration %d' % (a, j))
        print('For a = %d, Overlap = %f' % (a, np.mean(OverlapMatrix[i, :])))

mean_overlaps = np.mean(OverlapMatrix, axis=1)
print('Mean overlaps for each condition: %f' % mean_overlaps)
```

PART B: RUN ON POLBLOGS DATA

```python
# load the data
import networkx as nx  # from piazza
G = nx.read_gml('polblogs.gml', label='id')
Ag = nx.adjacency_matrix(G).toarray()
Ag = np.fmin(Ag + Ag.T, 1)
sigma_true = np.array([G.node[key]['value'] for key in G.node])
# remove isolated nodes
idx = (np.sum(Ag, 1) != 0)
Ag = Ag[np.ix_(idx, idx)]
sigma_true = sigma_true[idx]

# apply relaxation based clustering
k = 2
T = 100
lambda_value = np.mean(np.sum(Ag, axis=0))  # lambda >= avg degree of G
for r in range(2, 21):
    sigma_hat = RelaxationClustering(Ag, lambda_value, r, T)
    Q = Overlap(sigma_true, sigma_hat.astype(int), k)
    print('For r = %d, overlap = %f' % (r, Q))
```