MDS6106 Final Project Report

Convex Clustering

LI Chenliang	221041006
JIN Xuyang	221041010
LI Yuxiao	221041017
QUAN Beichen	221041039
SUN Wang	221041045

Contents

1 Introduction

We are interested in the general unsupervised clustering problems by investigations on different optimization models and try to utilize minimization methodologies that were introduced in the lecture to solve convex clustering problems for large-scale unsupervised learning.

General form of this kind of optimization problem is:

$$\min_{X \in \mathbb{R}^{d\times n}} \frac{1}{2}\sum_{i=1}^{n} ||x_i - a_i||^2 + \lambda \sum_{i=1}^{n}\sum_{j=i+1}^{n} ||x_i - x_j||_l \tag{1}$$

where a_i‘s are the data points. In total we have n data inputs with dimensions being d. l denotes the norm form we are using; for example, $l = 2$ refers to the normal Euclidean norm.

After solving (1) and obtaining the optimal solution $X = (x_1, x_2, ..., x_n)$, we assign a_i and a_j to the same cluster i.f.f. $x_i = x_j$. In other words, x_i acts as a centroid (center of the cluster) for the observation a_i. In practice, instead of requiring $x_i = x_j$, we can assign a_i and a_j to the same cluster if $||x_i - x_j|| \leqslant \varepsilon$ for a given tolerance $\varepsilon > 0$.

Diving into the details, we first randomly generated four datasets for test and study. Since the second term in (1) can be in place of different forms for specific fields of problems, for the second part the optimization problem becomes:

$$\min_{X \in \mathbb{R}^{d\times n}} \frac{1}{2}\sum_{i=1}^{n} ||x_i - a_i||^2 + \lambda \sum_{i=1}^{n}\sum_{j=i+1}^{n} \varphi_{\text{hub}}(x_i - x_j) \tag{2}$$

Huber norm has the following properties:

$$\varphi_{\text{hub}}(y) = \begin{cases} \frac{1}{2\delta}||y||^2 & \text{if } ||y|| \leqslant \delta \\ ||y|| - \frac{\delta}{2} & \text{if } ||y|| > \delta \end{cases} \quad \rightarrow \quad \nabla\varphi_{\text{hub}}(y) = \frac{y}{\max\{\delta, ||y||\}}$$

For the third part, the form changes again:

$$\min_{X \in \mathbb{R}^{d\times n}} \frac{1}{2}\sum_{i=1}^{n} ||x_i - a_i||^2 + \lambda \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{ij}||x_i - x_j|| \tag{3}$$

where

$$w_{ij} = \begin{cases} \exp(-\vartheta||a_i - a_j||^2) & \text{if } (i,j) \in \boldsymbol{\mathcal{E}} \\ 0 & \text{otherwise} \end{cases}$$

and $\boldsymbol{\mathcal{E}} = \bigcup_{i=1}^{n}\{(i,j) : a_j \text{ is among } a_i\text{‘s } k\text{-nearest neighbors}, \ i < j \leqslant n\}$, $\vartheta > 0$ is given.

We applied AGM and Newton-CG for the parts above. Necessary changes and trials for selection in terms of parameters are conducted for a decent outcome.

For the last part, we altered our optimization strategy with other methods such as ADAM, SGD and other types of norms. Both of the generated data and real data are tested within several methods. Detailed results can be found in the rest of this report.

2 Data Preparation

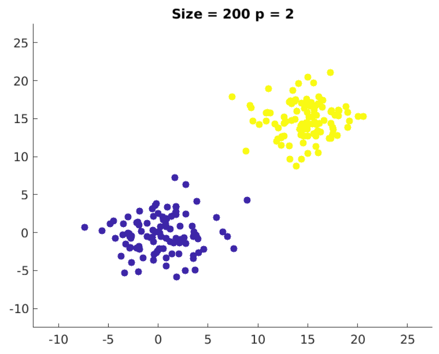
Firstly, as requested, we generated three data point clouds under the methodology given:

$$a_j = c_i + \begin{pmatrix} \varepsilon_{j,1} \\ \varepsilon_{j,2} \end{pmatrix}; \quad \varepsilon_{j,1}, \varepsilon_{j,2} \sim N(0, \sigma_i^2), \quad i = 1, 2, \ldots, p$$

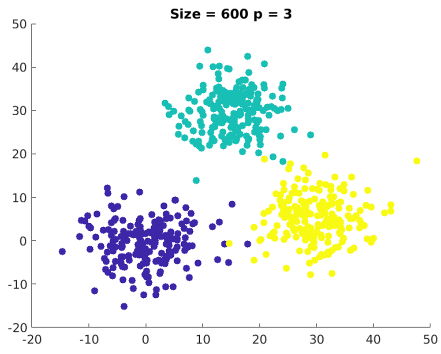
and generate n data points $a_1, \ldots, a_n \in \mathbb{R}^2$ with p centroids. For simplicity we divide the data points equally.

dataset	d	n	classes	reference points	std
1	2	200	2	(0,0) (15,15)	[2.5, 2.5]
2	2	600	3	(0,0) (15,30) (30,5)	[5, 5, 5]
3	2	400	4	(0,0) (10,20) (20,0) (30,20)	[5, 5, 5, 5]

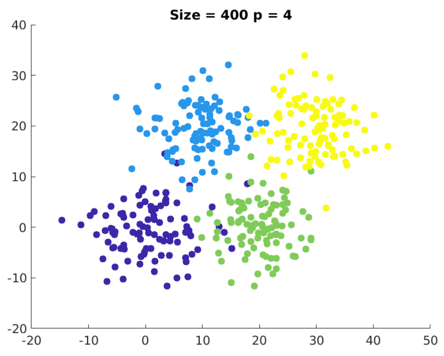
Table 1: Dataset Information



(a) Dataset 1



(b) Dataset 2



(c) Dataset 3

Figure 1: Dataset Visualization

3 Huber-type Clustering and Weighted Models

We implemented AGM and Newton CG in optimizing the target function, and we tried different lambdas (in total 11 values, from 0.05 to 0.1 with step being 0.005) to generate different outcomes of classification with 3 synthetic datasets. The scales of these 3 datasets are 200, 600, 400, with 2, 3, 4 centroids respectively. We will compare the outcomes with different lambdas and convergence in different methods.

To be specific and clear, the methods we used are showing in the title of the graph, namely:

1. "AGMbeta1" stands for the AGM method with *beta* involving t, using the Huber Norm;
2. "AGMbeta2" stands for the AGM method with *beta* involving L & μ only, using Huber Norm;
3. "NewtonCG" stands for the Newton method using CG under Huber Norm;
4. "AGMbeta1Weighted" stands for the similar setup with "AGMbeta1" using weighted model instead of Huber Norm;
5. "GM" stands for gradient method under Huber Norm, this is only for comparison.

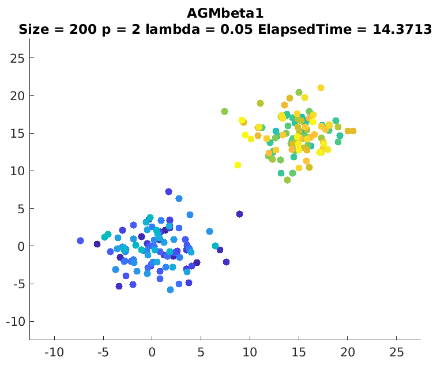
3.1 Outcomes of classification

We check the data set one by one and list respective methods with different values of λ.

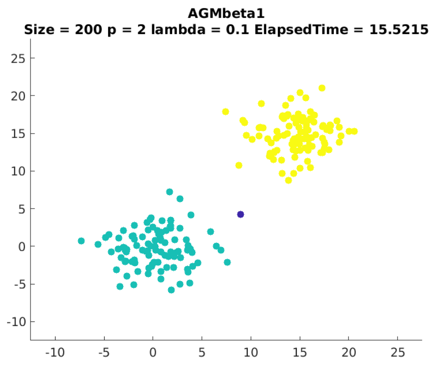
3.1.1 Dataset 1: $n = 200,\ p = 2$

3.1.1.1 Method: AGMbeta1

Here we present the result for $\lambda = 0.05$ and $\lambda = 0.1$. During our study, we found out that only when λ achieves 0.1 can the clustering result behave according to our expectations.



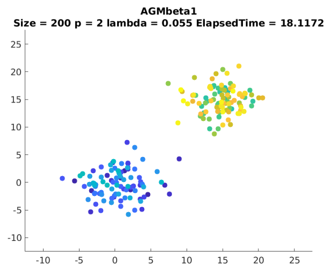
(a) $\lambda = 0.05$



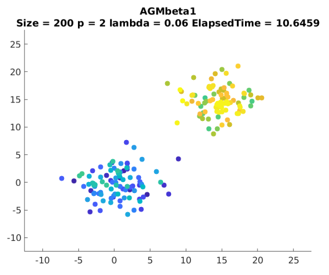
(b) $\lambda = 0.1$

Figure 2: Clustering result for dataset 1 using AGMbeta1 (representative)

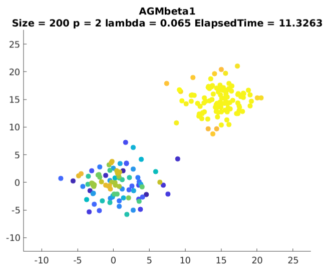
Detailed results for the λ values in between:



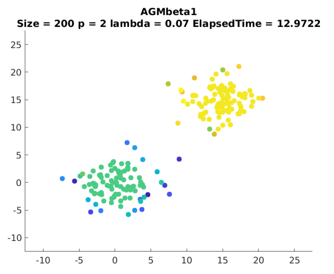
(a) $\lambda = 0.055$



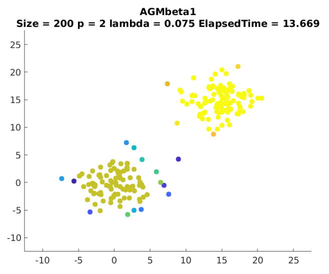
(b) $\lambda = 0.06$



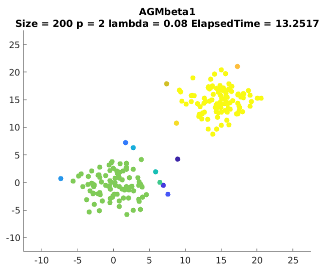
(c) $\lambda = 0.065$



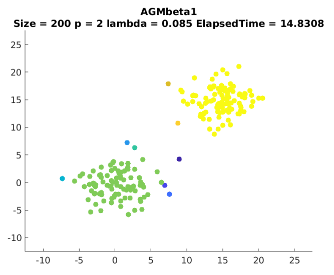
(d) $\lambda = 0.07$



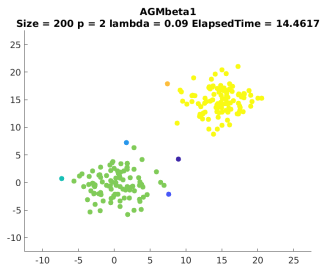
(e) $\lambda = 0.075$



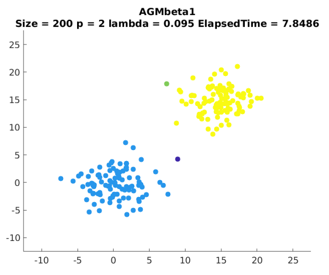
(f) $\lambda = 0.08$



(g) $\lambda = 0.085$



(h) $\lambda = 0.09$

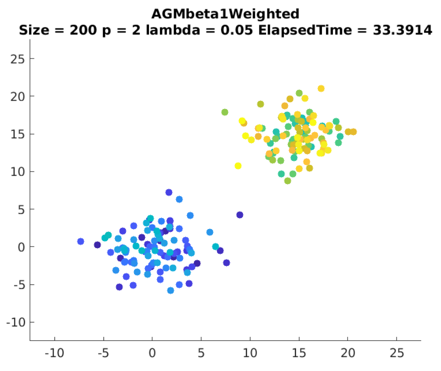


(i) $\lambda = 0.095$

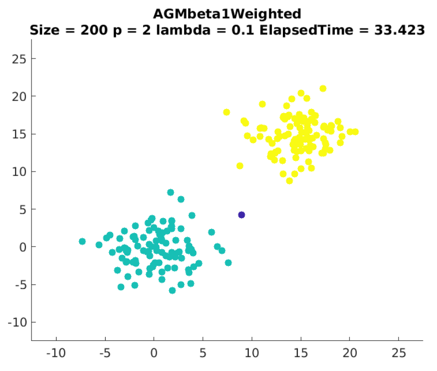
Figure 3: Clustering results for dataset 1 using AGMbeta1 (details)

3.1.1.2 Method: AGMbeta1Weighted

Here we present the result for $\lambda = 0.05$ and $\lambda = 0.1$. Similar to the result in the previous parts, 0.1 is the best result we can achieve.



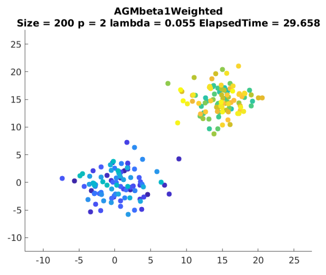
(a) $\lambda = 0.05$



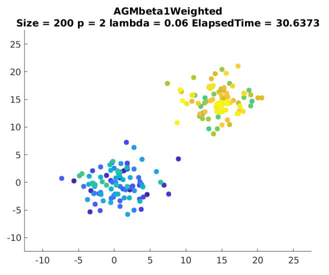
(b) $\lambda = 0.1$

Figure 4: Clustering result for dataset 1 using AGMbeta1Weighted (representative)

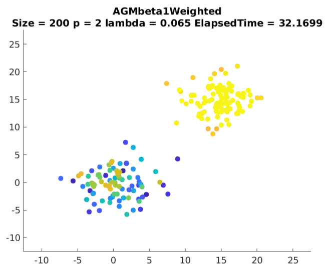
Detailed results for the λ values in between:



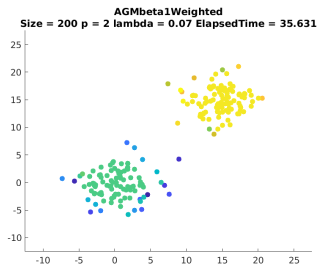
(a) $\lambda = 0.055$



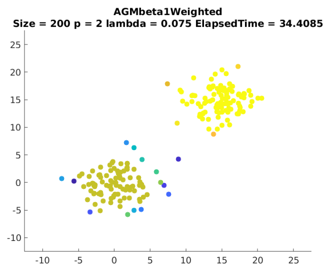
(b) $\lambda = 0.06$



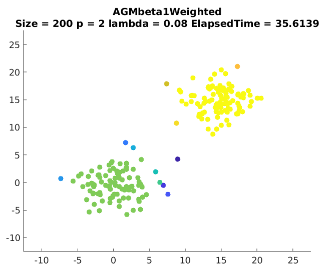
(c) $\lambda = 0.065$



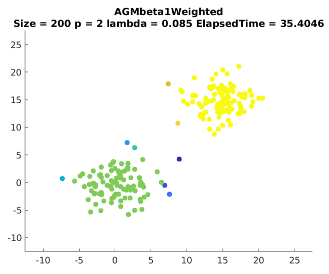
(d) $\lambda = 0.07$



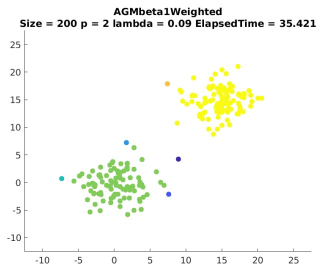
(e) $\lambda = 0.075$



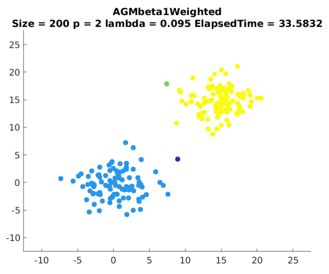
(f) $\lambda = 0.08$



(g) $\lambda = 0.085$



(h) $\lambda = 0.09$

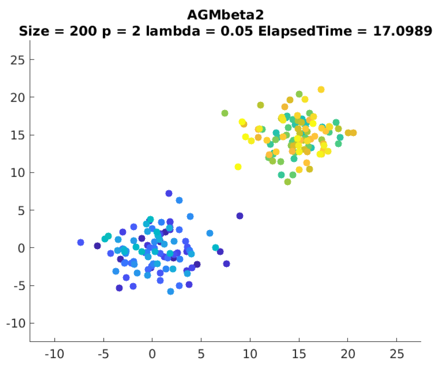


(i) $\lambda = 0.095$

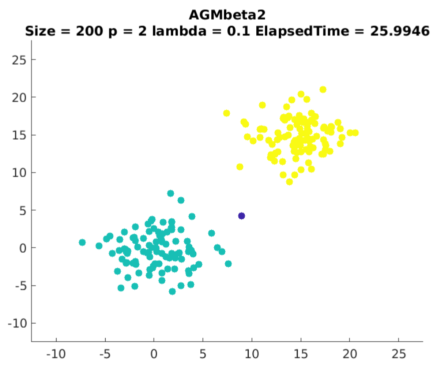
Figure 5: Clustering results for dataset 1 using AGMbeta1Weighted (details)

3.1.1.3 Method: AGMbeta2

Here we present the result for $\lambda = 0.05$ and $\lambda = 0.1$. Similar to the result in the previous two parts,$\lambda = 0.1$ is the best result we can achieve.



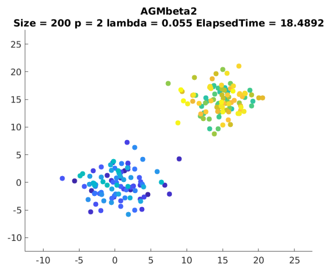
(a) $\lambda = 0.05$



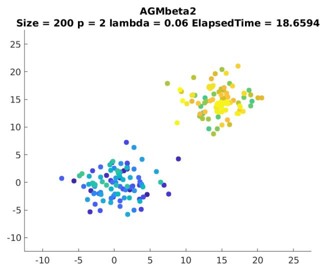
(b) $\lambda = 0.1$

Figure 6: Clustering result for dataset 1 using AGMbeta1Weighted (representative)

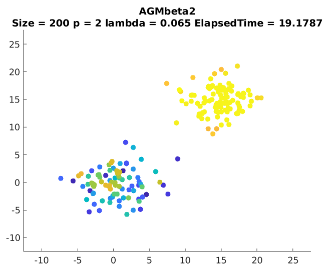
Detailed results for the λ values in between:



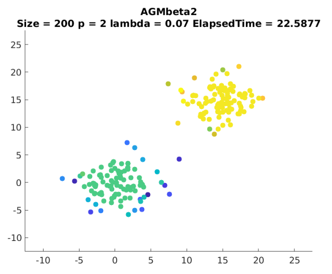
(a) $\lambda = 0.055$



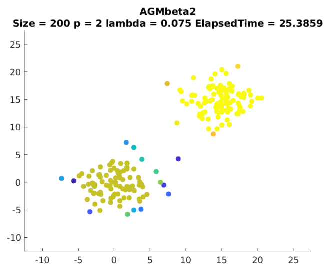
(b) $\lambda = 0.06$



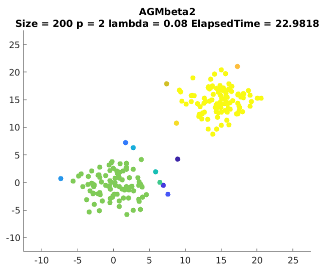
(c) $\lambda = 0.065$



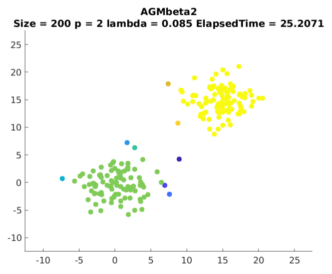
(d) $\lambda = 0.07$



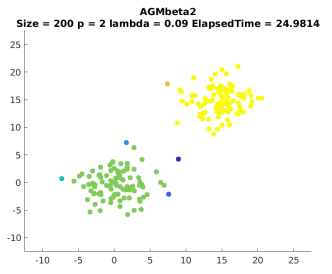
(e) $\lambda = 0.075$



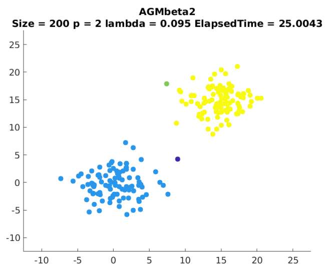
(f) $\lambda = 0.08$



(g) $\lambda = 0.085$



(h) $\lambda = 0.09$

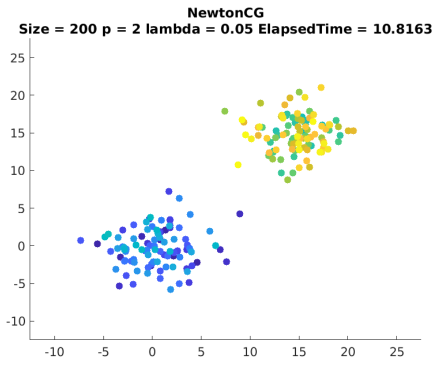


(i) $\lambda = 0.095$

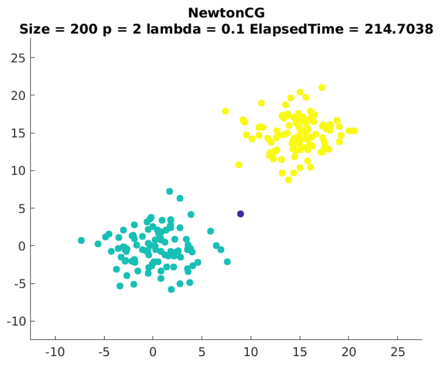
Figure 7: Clustering results for dataset 1 using AGMbeta2 (details)

3.1.1.4 Method: NewtonCG

Here we present the result for $\lambda = 0.05$ and $\lambda = 0.1$. Similar to the result in the previous three parts,when $\lambda = 0.1$, we can achieve the best results.



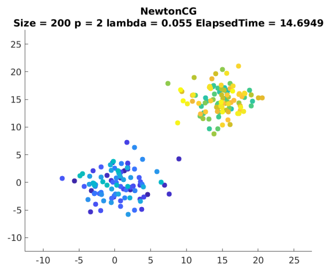
(a) $\lambda = 0.05$



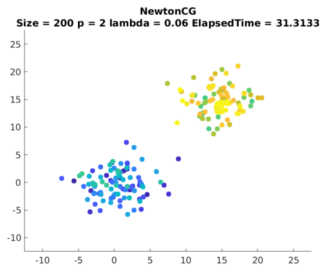
(b) $\lambda = 0.1$

Figure 8: Clustering result for dataset 1 using NewtonCG (representative)

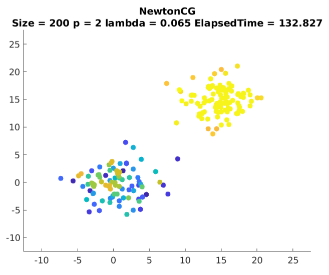
Detailed results for the λ values in between:



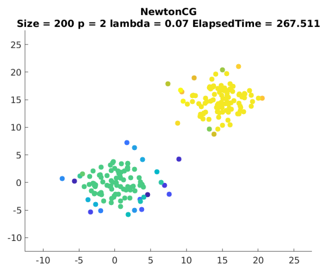
(a) $\lambda = 0.055$



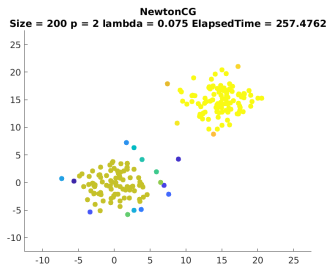
(b) $\lambda = 0.06$



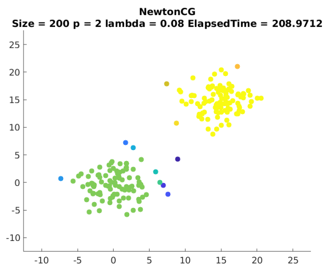
(c) $\lambda = 0.065$



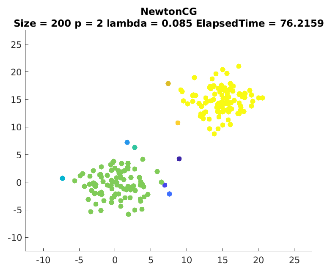
(d) $\lambda = 0.07$



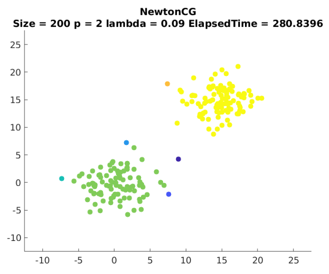
(e) $\lambda = 0.075$



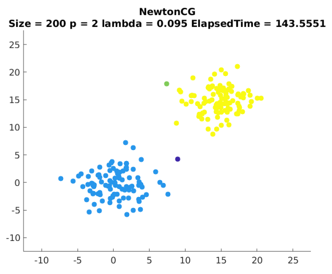
(f) $\lambda = 0.08$



(g) $\lambda = 0.085$



(h) $\lambda = 0.09$



(i) $\lambda = 0.095$

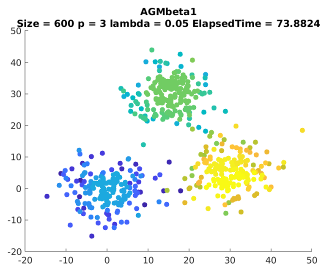
Figure 9: Clustering results for dataset 1 using NewtonCG (details)

3.1.1.5 Conclusion of observations

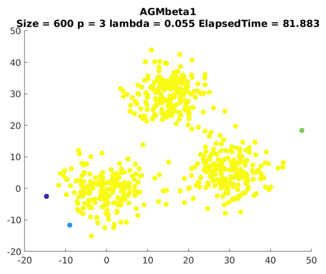
From the above results, we can see that, for dataset 1 with 200 observations, a relative large value of λ is good for clustering, and under this case, the best result of achieved at $\lambda = 0.1$.

3.1.2 Dataset 2: n = 600, p = 3

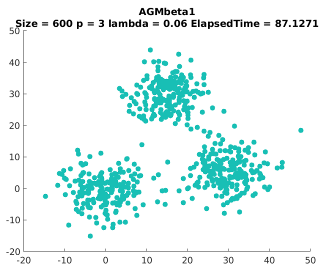
3.1.2.1 Method: AGMbeta1



(a) $\lambda = 0.05$



(b) $\lambda = 0.055$

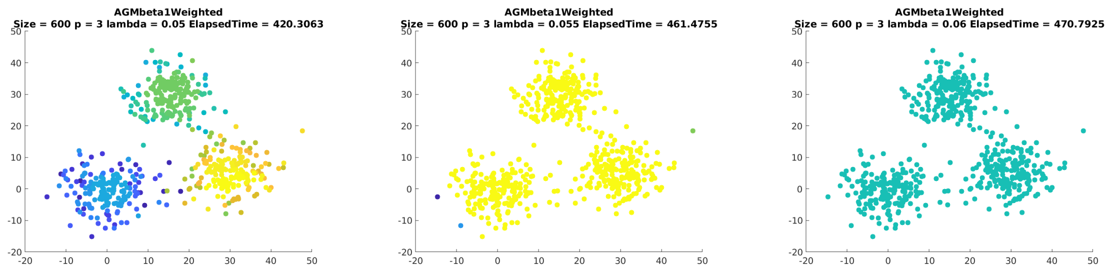


(c) $\lambda = 0.06$

Figure 10: Clustering result for dataset 2 using AGMbeta1 (representative)

Different from the result in the dataset 1, this time λ behaves more sensitively. As you can see in the plots on last page, for $\lambda = 0.05$ we can see there is clustering behavior approximately but more than 3 groups, while after λ jumps to value larger than 0.055, they combined into one group. The other results are quite alike the situation for $\lambda = 0.06$ and we omit to show them here as a result.

3.1.2.2 Method: AGMbeta1Weighted

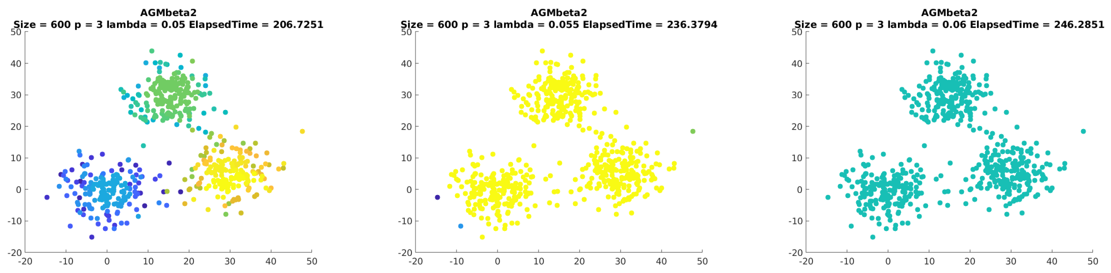


(a) $\lambda = 0.05$ (b) $\lambda = 0.055$ (c) $\lambda = 0.06$

Figure 11: Clustering result for dataset 2 using AGMbeta1Weighted (representative)

It is easy to see that the result is quite alike what we have got in the previous part.

3.1.2.3 Method: AGMbeta2

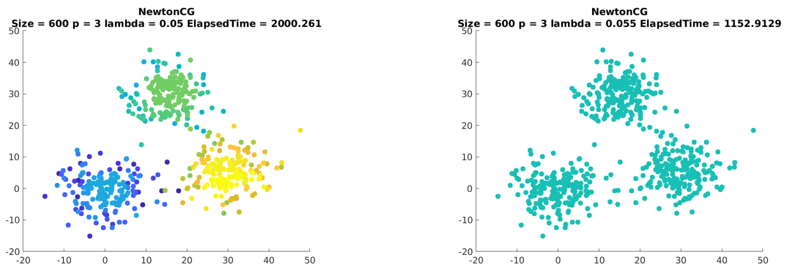


(a) $\lambda = 0.05$ (b) $\lambda = 0.055$ (c) $\lambda = 0.06$

Figure 12: Clustering result for dataset 2 using AGMbeta2 (representative)

Similarly, the result is quite alike the previous two parts.

3.1.2.4 Method: NewtonCG



(a) $\lambda = 0.05$ (b) $\lambda = 0.055$

Figure 13: Clustering result for dataset 2 using NewtonCG (representative)

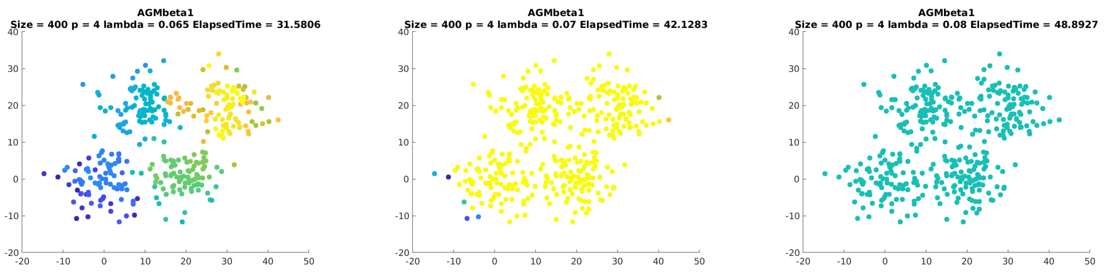
This time the clustering change is even more sensitive as the situations for $\lambda \geqslant 0.55$ behave alike already.

3.1.2.5 Conclusion of observations

From the above results, we can see that, for dataset 2 with 600 observations, $\lambda = 0.055$ is already too large for clustering, indicating that the sensitivity increased significantly due to the nature of data set.

3.1.3 Dataset 3: n = 400, p = 4

3.1.3.1 Method: AGMbeta1

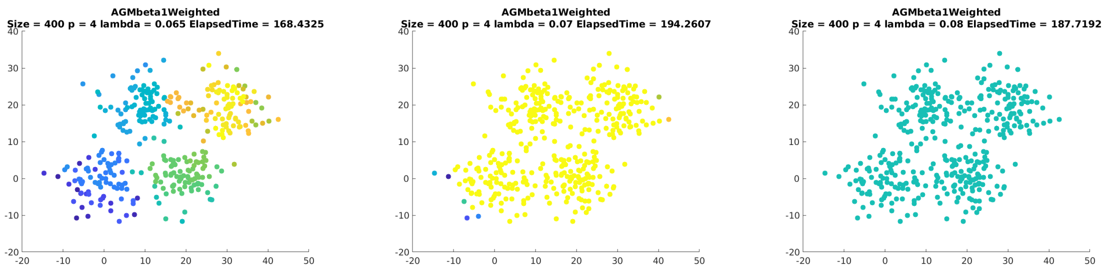


(a) $\lambda = 0.065$ (b) $\lambda = 0.07$ (c) $\lambda = 0.08$

Figure 14: Clustering result for dataset 3 using AGMbeta1 (representative)

Different from the result in the dataset 1 and 2, this time λ behaves sensitively on a different level. As you can see in the plots on last page, for $\lambda = 0.065$ we can see there is clustering behavior approximately but more than 4 groups, while after λ jumps to value larger than 0.07, they combined into one group.

3.1.3.2 Method: AGMbeta1Weighted

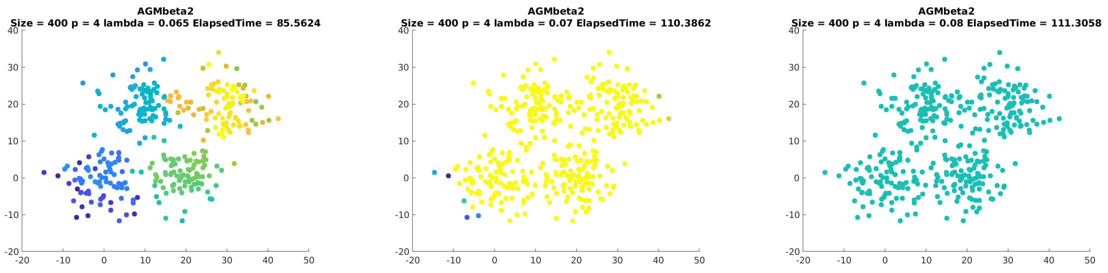


(a) $\lambda = 0.065$ (b) $\lambda = 0.07$ (c) $\lambda = 0.08$

Figure 15: Clustering result for dataset 3 using AGMbeta1Weighted (representative)

It is easy to see that the result is quite alike what we have got in the previous part.

3.1.3.3 Method: AGMbeta2



(a) $\lambda = 0.065$ (b) $\lambda = 0.07$ (c) $\lambda = 0.08$

Figure 16: Clustering result for dataset 3 using AGMbeta2 (representative)

Similarly, the result is quite alike the previous two parts.

3.1.3.4 Method: NewtonCG

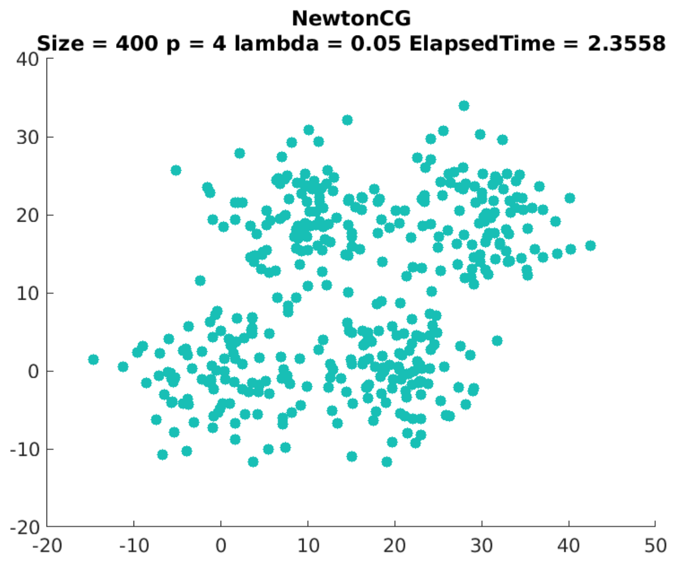


Figure 17: Clustering result for dataset 3 using NewtonCG (λ from $0.05 \sim 0.1$)

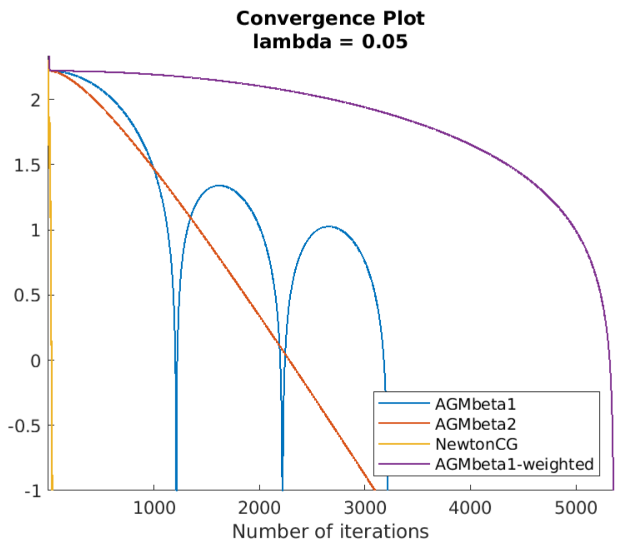
This time the clustering is ugly as the NewtonCG method didn‘t gives us the proper result and just cluster all the data into one group.

3.1.3.5 Conclusion of observations

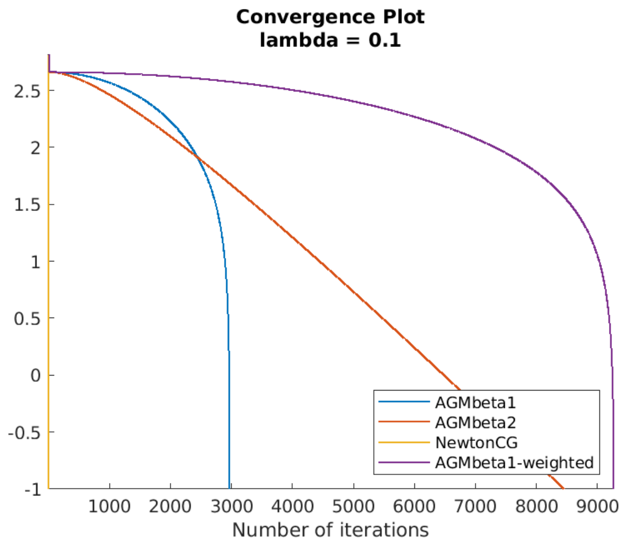
From the above results, we can see that, for dataset 2 with 600 observations, $\lambda = 0.07$ is already too large for clustering. Moreover, NewtonCG method is not applicable as it has no power of clustering at all. This might be due to the choice of starting point.

3.2 Convergence

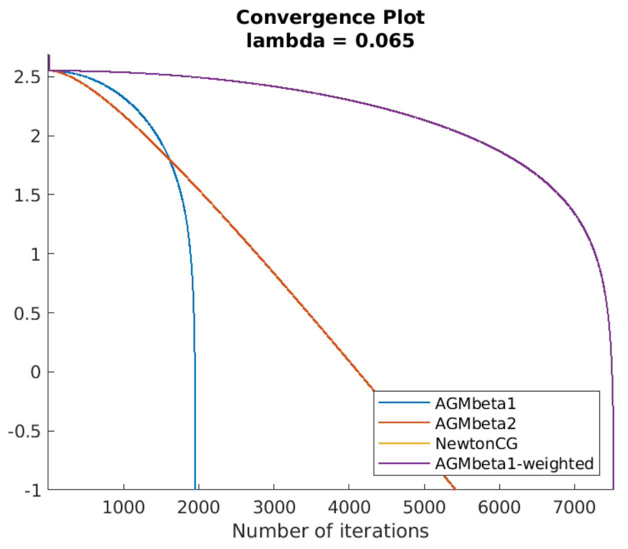
Figure 18 shows convergence of different kinds of methods in 3 datasets. Newton CG has the highest convergence speed. AGM beta 1and AGM beta 2 converges after similar steps, but the former has some fluctuations. AGM-weighted converges at lowest speed, but steadily.



(a) Dataset 1



(b) Dataset 2



(c) Dataset 3

Figure 18: Convergence results

4 Performance, Extensions, and Stochastic Optimization

4.1 BFGS and L-BFGS

4.1.1 Implementation

In this section, we try to use BFGS and L-BFGS approximate optimization algorithm to have a approximation about $(\nabla^2 f(x))^{-1}$ is calculated as follows:

$$H_{k+1} = H_k + \frac{(s^k - H_k y^k)(s^k)^T + s^k(s^k H_k y^k)^T}{(s^k)^T y^k} - \frac{(s^k - H_k y^k).T y^k}{((s^k)^T y^k)^2} \cdot s^k (s^k)^T \tag{4}$$

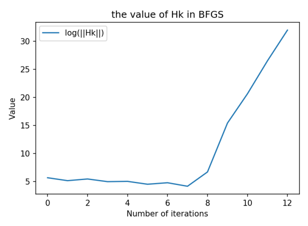
where $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

we also use two-loop algorithm to calculate direction($\nabla f(x_k) H_k$) to save the storage space (known as L-BFGS) but both of them encounter overflow problem.

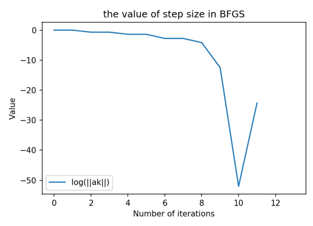
4.1.2 Overflow Problem

After implementation, we have test BFGS and L-BFGS in cluster question,and both encounter overflow question, this is because H_k increased rapidly with the increment of k, then the second norm value of H_k approaches positive infinity, which will lead to a overflow question of python, or extremely small step size almost 0 due to the limit of floating point precision of the computer. so we can not use BFGS or L-BFGS singly to get a x_* with required tolerance.

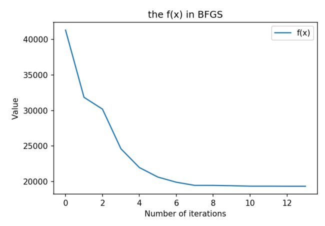
You can see the change of step size and the second norm value of H_k below. BFGS or L-BFGS can approach the convergence point to some extent, and it is very fast until the problem happened.



(a) the trend of $||H_k||$



(b) the trend of step size a



(c) the trend of f(x)

4.1.3 Solution Strategy

After discussion, we may have two solution strategies to solve this problem, the first one is to change a computer with higher floating point precision.and the secondly is to change the direction or update strategy before the overview problem happening. (for example, use gradient descent function, but it will slow down the convergence time and have higher iteration times.)

4.2 Extensions

We tried other optimization implementations and different kinds of penalty functions. Firstly, we applied Adam and SGD into the synthetic datasets and compare them with AGM and Newton-CG. The convergence outcome is shown in Figure 19. Since Adam and SGD are both gradient based, their convergence is much slow than that of AGM or Newton-CG. In addition, Adam converges faster than SGD in the beginning, and it slows down as the number of iterations increases.

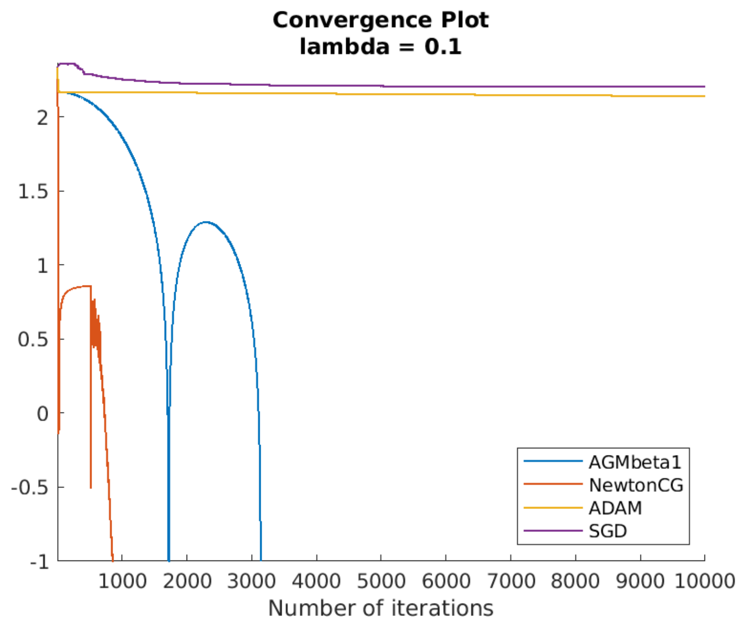


Figure 19

We applied L-1 norm, L-2 norm, L-infinity norm and nonconvex log-function in the penalty function. Since l1 norm and maximum norm is not twice differentiable, we only implement them in AGM. The convergence outcome is shown in Figure 20. Nonconvex log-function has faster convergence in the beginning but ends up with similar number of iterations with huber function. The norm of gradients of l1 norm and maximum norm decreases steadily in the first 1000 steps but fluctuates after then. This is because the gradient of them is a constant when x is far from x^*. x takes large steps before entering the local zone.

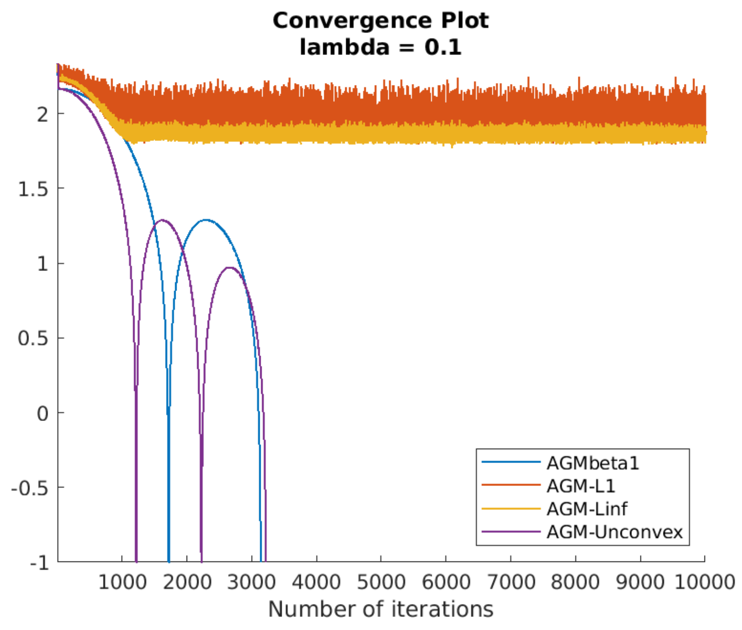


Figure 20

We applied AGM and Newton CG in two real datasets, wine and vowel. We evaluate the performance of classification with Adjusted Rand Index (ARI). Rand Index (RI) computes a similarity measure between two clusters by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusters. RI is calculated as follows:

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

To guarantee that the score of random classification should be close to 0 as an evaluation index, ARI is put forward, which has a higher degree of differentiation. RI is calculated as follows:

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

ARI ranges from -1 to 1. Higher ARI means higher similarity.

To guarantee good performance of classification, different lambdas are tried and we only use the one with the best performance. The outcome is shown in Table 2. In the dataset of wine, AGM and Newton-CG has similar performance of classification. In the dataset of vowel, the ARI of AGM is 0.118, while the ARI of Newton-CG is around 0 though several lambdas are tried.

ARI	AGM	Newton-CG
wine	0.339	0.338
vowel	0.118	NA

Table 2

Appendix

Calculation for μ in β_2

$\nabla f = (I_{n\times n} + Q\Lambda Q^\top)X - a$, let $P = I_{n\times n} + Q\Lambda Q^\top$, we have $\nabla^2 f = K = P \otimes I_{d\times d}$.

We need to calculate the minimum eigenvalue of K.

1. Proof for K & P sharing identical eigenvalues

Denote the eigenvalues of P as $\lambda_1, \lambda_2, \ldots, \lambda_n$ and I as $\mu_1, \mu_2, \ldots, \mu_d$.
Clearly $\mu_j = 1,\ \forall j = 1, 2, \ldots, d$.

Moreover, the set of eigenvalues for K is $\{\lambda_i \mu_j \mid i = 1, 2, \ldots, n\ ,\ j = 1, 2, \ldots, d\}$, which is equivalent to $\{\lambda_i \mid i = 1, 2, \ldots, n\}$ since all the u_j's are 1.

2. Proof for $Q\Lambda Q^\top$ to be positive semidefinite

$$\forall\ x \in \mathbb{R}^n, x^\top Q\Lambda Q^\top x := y^\top \Lambda y \qquad (y = Q^\top x)$$

(i) Since $\text{rank}(Q) < n$, $Q^\top x = 0$ has non-zero solutions, i.e. if $Q^\top x = 0 \ \Rightarrow\ y^\top \Lambda y = 0$.
(ii) If $Q^\top x \neq 0$, we have $y^\top \Lambda y > 0$ due to the reason that Λ is positive definite.

As a result, $x^\top Q\Lambda Q^\top x = y^\top \Lambda y \geqslant 0 \ \Rightarrow\ Q\Lambda Q^\top$ is positive semidefinite.

Conclusion:
(1)Based on part 2, $Q\Lambda Q^\top$ has a minimum eigenvalue being 0; thus for P, it is 1.
(2)Based on part 1, we have K has its minimum eigenvalue being 1 also.

Gradient and Hessian of Huber Norm

Let $x_i \in \mathbb{R}^d, i = 1, 2, \ldots, n$, and $y_i \in \mathbb{R}^d, i = 1, 2, \ldots, \frac{n(n-1)}{2}$.

$$\text{Then } X = \begin{pmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_n^\top \end{pmatrix}_{n\times d}, \text{ and } Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\frac{n(n-1)}{2}} \end{pmatrix} = \begin{pmatrix} x_1^\top - x_2^\top \\ x_1^\top - x_3^\top \\ \vdots \\ x_{n-1}^\top - x_n^\top \end{pmatrix}_{\frac{n(n-1)}{2}\times d} = (Q_n)^\top X.$$

$$\text{where } Q_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}_{2\times 1}, Q_3 = \begin{pmatrix} \mathbf{1}^\top & \mathbf{0}^\top \\ -I_2 & Q_2^\top \end{pmatrix}_{3\times 3}, \ldots, Q_n = \begin{pmatrix} \mathbf{1}^\top & \mathbf{0}^\top \\ -I_{n-1} & Q_{n-1}^\top \end{pmatrix}_{n\times \frac{n(n-1)}{2}}.$$

Let $g(Y) = \mathbf{1}^\top \varphi_{hub}(Y) = \sum_{i=1}^{n(n-1)/2} \varphi_{hub}(y_i)$, where $\nabla \varphi_{hub}(y_i) = \frac{y_i}{\max\{\|y_i\|, \delta\}}$.

$$\frac{\partial g(Y)}{\partial X} = \mathrm{vec}(\nabla \varphi_{hub}(Y)) \cdot \frac{\mathrm{d}Y}{\mathrm{d}X} \qquad \text{where } \frac{\mathrm{d}Y}{\mathrm{d}X} = \begin{pmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} & \dots & \frac{\mathrm{d}y_1}{\mathrm{d}x_n} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} & \dots & \frac{\mathrm{d}y_2}{\mathrm{d}x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\mathrm{d}y_{n(n-1)/2}}{\mathrm{d}x_1} & \frac{\mathrm{d}y_{n(n-1)/2}}{\mathrm{d}x_2} & \dots & \frac{\mathrm{d}y_{n(n-1)/2}}{\mathrm{d}x_n} \end{pmatrix}_{\frac{n(n-1)}{2}d \times n \times d}$$

$$= \left(\nabla \varphi_{hub}(y_1)^\top, \nabla \varphi_{hub}(y_2)^\top, \dots, \nabla \varphi_{hub}(y_{n(n-1)/2})^\top\right)_{1 \times \frac{n(n-1)}{2}d} \cdot \frac{\mathrm{d}Y}{\mathrm{d}X}$$

$$= \begin{pmatrix} \sum_{i=1}^{n(n-1)/2} \nabla \varphi_{hub}(y_i)^\top \cdot \frac{\partial y_i}{\partial x_1} \\ \sum_{i=1}^{n(n-1)/2} \nabla \varphi_{hub}(y_i)^\top \cdot \frac{\partial y_i}{\partial x_2} \\ \vdots \\ \sum_{i=1}^{n(n-1)/2} \nabla \varphi_{hub}(y_i)^\top \cdot \frac{\partial y_i}{\partial x_n} \end{pmatrix}_{n \times d}$$

$$= Q_n \Lambda Q_n^\top X$$

where $\Lambda = \mathrm{diag}\left(\frac{1}{\max\{\|y_i\|, \delta\}}\right)_{\frac{n(n-1)}{2}}$.

$$\frac{\partial^2 g(Y)}{\partial X^2} := K = \begin{pmatrix} \mathrm{diag}(q_{11}) & \mathrm{diag}(q_{12}) & \dots & \mathrm{diag}(q_{1n}) \\ \mathrm{diag}(q_{21}) & \mathrm{diag}(q_{22}) & \dots & \mathrm{diag}(q_{2n}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{diag}(q_{n1}) & \mathrm{diag}(q_{n2}) & \dots & \mathrm{diag}(q_{nn}) \end{pmatrix}_{nd \times nd} \qquad \text{for} \quad q_{ij} \in Q_n \Lambda Q_n^\top$$

In other form :

$$\frac{\partial^2 g(Y)}{\partial X^2} = (Q_n \Lambda Q_n^\top) \otimes \mathbf{I}$$

Gradient of Huber-type L-1 Norm and L-infinity Norm

L-1 Norm

$$\nabla \varphi_{hub}(y) = \begin{cases} \frac{\|y\|_1}{\delta} \operatorname{sign}(y) & \text{if} \|y\|_1 \le \delta \\ \operatorname{sign}(y) & \text{if} \|y\|_1 > \delta \end{cases}$$

L-infinity Norm

$$\nabla \varphi_{hub}(y) = \begin{cases} \frac{\|y\|_\infty}{\delta} \mathbf{e}_i & \text{if} \|y\|_\infty \le \delta \\ \mathbf{e}_i & \text{if} \|y\|_\infty > \delta \end{cases} \qquad \text{where } y_i = \max(y)$$

Matlab Code

```
function [Q] = generate_Q(N)
    Q = sparse(N,N*(N-1)/2);
    csum = [0,cumsum(flip(1:N-1))];
    for i = 1:N-1
        start = csum(i)+1;
        stop = csum(i+1);
        Q(i+1:end,start:stop) = -speye(N-i);
        Q(i,start:stop) = 1;
    end
end
```

```
function [iter,ng,x] = AGM_beta1(d,Q,a,L,x0,lambda,delta,tol,print_output
    )
    x = x0;
    xp = x;
    t = 1;
    tp = 1;
    iter = 0;
    ng = [];
    while 1
        iter = iter + 1;
        beta = (tp-1)/t;
        y = x + beta*(x-xp);
        xp = x;
        h = Q'*y;
        normh = vecnorm(h,2,2);
        i = normh <= delta;
        f = (1/2)*sum(vecnorm(y - a,2,2).^2) + lambda*sum((1/(2*delta)).*
            normh(i).^2) + lambda*sum(normh(~i)-delta/2);
        tmp(i,:) = h(i,:)./delta;
        if size(h(~i,:),1) ~= 0
            tmp(~i,:) = h(~i,:)./normh(~i);
        end
        g = y - a + lambda.*(Q*tmp);
        x = y - g./L;
        tp = t;
        t = (1/2)*(1+sqrt(1+4*tp^2));
        normg = norm(g,'fro');
        ng = [ng;normg];
        if print_output == true
            fprintf('Iter = %d\tf = %f\tg = %f\n',iter,f,normg);
        end
        if normg < tol
            break;
        end
    end
end
```

```
function [iter,ng,x] = AGM_beta2(d,Q,a,L,x0,lambda,delta,tol,mu,
    print_output)
    x = x0;
    xp = x;
    iter = 0;
    ng = [];
    while 1
        iter = iter + 1;
        beta = (sqrt(L)-sqrt(mu))/(sqrt(L)+sqrt(mu));
        y = x + beta*(x-xp);
        xp = x;
        h = Q'*y;
        normh = vecnorm(h,2,2);
        i = normh <= delta;
        f = (1/2)*sum(vecnorm(y - a,2,2).^2) + lambda*sum((1/(2*delta)).*
            normh(i).^2) + lambda*sum(normh(~i)-delta/2);
        tmp(i,:) = h(i,:)./delta;
        if size(h(~i,:),1) ~= 0
            tmp(~i,:) = h(~i,:)./normh(~i);
        end
        g = y - a + lambda.*(Q*tmp);
        x = y - g./L;
        normg = norm(g,'fro');
        ng = [ng;normg];
        if print_output == true
            fprintf('Iter = %d\t f = %f\t g = %f\n',iter,f,normg);
        end
        if normg < tol
            break;
        end
    end
end
```

```
function [iter,ng,x] = AGM_weighted(d,Q,a,L,x0,lambda,delta,tol,w,
    print_output)
    x = x0;
    xp = x;
    t = 1;
    tp = 1;
    iter = 0;
    ng = [];
    while 1
        iter = iter + 1;
        beta = (tp-1)/t;
        y = x + beta*(x-xp);
        xp = x;
        h = Q'*y;
        normh = vecnorm(h,2,2);
        i = normh <= delta;
        f = (1/2)*sum(vecnorm(y - a,2,2).^2) + lambda*sum(w(i).*((1/(2*
            delta)).*normh(i).^2)) + lambda*sum(w(~i).*(normh(~i)-delta/2)
            );
        tmp(i,:) = h(i,:)./delta;
        if size(h(~i,:),1) ~= 0
            tmp(~i,:) = h(~i,:)./normh(~i);
        end
        g = y - a + lambda.*(Q*tmp);
        x = y - g./L;
        tp = t;
        t = (1/2)*(1+sqrt(1+4*tp^2));
        normg = norm(g,'fro');
        ng = [ng;normg];
        if print_output == true
            fprintf('Iter = %d\tf = %f\tg = %f\n',iter,f,normg);
        end
        if normg < tol
            break;
        end
    end
end
```

```
function [iter,ng,x] = newton_cg(d,Q,a,x0,lambda,delta,tol,options,
    print_output)
    x = x0;
    iter = 0;
    ng = [];
    while 1
        iter = iter + 1;
        h = Q'*x;
        normh = vecnorm(h,2,2);
        i = normh <= delta;
        f = (1/2)*sum(vecnorm(x - a,2,2).^2) + lambda*sum((1/(2*delta)).*
           normh(i).^2) + lambda*sum(normh(~i)-delta/2);
        tmp(i,:) = h(i,:)./delta;
        if size(h(~i,:),1) ~= 0
            tmp(~i,:) = h(~i,:)./normh(~i);
        end
        [P,K] = generate_K(h,Q,delta,lambda);
        g = P*x - a;
        normg = norm(g,'fro');
        cg_tol = min(1,normg^(0.1))*normg;
        v = sparse(size(x,1)*d,1);
        r = reshape(g.',size(x,1)*d,[]);
        p = -r;
        t = -g;
        dlt = 0;
        for j = 1:options.maxit
            tmp1 = K*p;
            tmp2 = p'*tmp1;
            if tmp2 <= 0
                if j == 1
                    t = -g;
                else
                    t = reshape(v',d,size(x,1))';
                end
                break;
            end
            normr = norm(r);
            dlt = normr^2 / tmp2;
            v = v + dlt.*p;
            r = r + dlt.*tmp1;
            normr_n = norm(r);
            if normr_n <= cg_tol
                t = reshape(v',d,size(x,1))';
                break;
            end
            beta = normr_n^2 / normr^2;
            p = -r + beta.*p;
            if j == options.maxit
                t = reshape(v',d,size(x,1))';
            end
        end
        alpha = options.s;
        xn = x + alpha.*t;
        h = Q'*xn;
        normh = vecnorm(h,2,2);
        i = normh <= delta;
        fn = (1/2)*sum(vecnorm(xn - a,2,2).^2) + lambda*sum((1/(2*delta)).*normh(i).^2) + lambda*sum(normh(~i)-delta/2);
        while fn > f - options.gamma * alpha * reshape(g.',size(x,1)*d,[])' * reshape(t.',size(x,1)*d,[])
            alpha = options.sigma * alpha;
            xn = x + alpha.*t;
            h = Q'*xn;
            normh = vecnorm(h,2,2);
            i = normh <= delta;
            fn = (1/2)*sum(vecnorm(xn - a,2,2).^2) + lambda*sum((1/(2*delta)).*normh(i).^2) + lambda*sum(normh(~i)-delta/2);
        end
        x = xn;
        g = P*xn - a;
        normg = norm(g,'fro');
        ng = [ng;normg];
        if print_output == true
            fprintf('Iter = %d\tf = %f\tg = %d\tj = %d\n',iter,fn,normg,j);
        end
        if normg <= tol
            break;
        end
    end
end
```