A METHOD OF ACCELERATING K-MEANS BY DIRECTED PERTURBATION OF THE CODEVECTORS

By

SUMAKWEL MURALLA

Bachelor of Science in Civil Engineering

University of the Philippines

Diliman, Quezon City

Philippines

1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2006

A METHOD OF ACCELERATING

K-MEANS BY DIRECTED PERTURBATION

OF THE CODEVECTORS

Thesis Approved:

Douglas R. Heisterkamp
Thesis Adviser

John P. Chandler

H. K. Dai

A. Gordon Emslie
Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to express my heartfelt appreciation to my committee members for their guidance in the preparation of this thesis. A million thanks go to my adviser and chairman of the committee, Dr. Douglas R. Heisterkamp. His expertise, insight and friendship in guiding me through the whole process were invaluable. Doug, I will always owe you a debt of gratitude for your help and concern. My appreciation goes also to my committee members, Dr. John P. Chandler and Dr. H. K. Dai, for their comments and time they have spent in the review of this document.

To my friends here in Stillwater: the Lucases, especially Ed and mother Aida, the Tongcos, the Lims, the Tubanas, Arnold, and the Filipino community here in Stillwater, thank you for your friendship. To my church family: our pastor and his wife, our brethren in the church, thank you all for your prayers and love.

To Papa and Mama, my mother-in-law and father-in-law, my brother and sisters, thank you for your love and prayers. Most of all, special thanks go to my wife Rose and my son, Joseph. Thanks for being patient with me. You are my inspiration and comfort. And finally, thanks to God Almighty for His blessings, patience, and infinite love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. Introduction

1.1 Background

Cluster analysis (data clustering) is an important research domain in data mining. It has lots of applications in the fields of machine learning, pattern recognition, image analysis and bioinformatics. This area of study is relevant in business, biology, geography, medicine, and web archive, to name a few, and has become a very hot research topic today [1, 3]. Clustering algorithms are used to discover structures or patterns in data in order to gain new knowledge and insight from a database. Clustering algorithms group data instances into subsets or clusters that have similar features by using proximity according to some predefined distance measure, usually using Euclidean metric. This, consequently, builds "concept hierarchies" [25] that are useful in gaining new knowledge from the database.

1.2 Clustering Classifications and Definitions

Data clustering are usually classified into two groups namely *hierarchical clustering* and *partitional* clustering. In hierarchical clustering "the data are not partitioned into a particular number of classes or clusters at a single step. Instead the classification consists of a series of partitions which may run from a single cluster containing all individuals, to *n* clusters each containing a single individual. Hierarchical clustering techniques may be subdivided into *agglomerative* methods which proceed by a series of successive fusions of the *n* individuals into groups, and *divisive* methods, which separate the *n* individuals

successively into finer groupings" [19, p.55]. Hierarchical clustering usually does not specify *a priori* the number of clusters. "Prototype-based partitional clustering algorithms can be divided into two classes: *crisp(or hard)* clustering where each data point belongs to only one cluster, and *fuzzy* clustering where every data point belongs to every cluster to a certain degree. Fuzzy clustering algorithms can deal with overlapping cluster boundaries. Partitional algorithms are dynamic, and points can move from one cluster to another [24, 2]."

Clustering is usually defined as follows: Given a set of N points in a feature space consisting of d dimensions, find subsets (or clusters) of interesting points. "Techniques of cluster analysis seek to separate a set of data into its constituent groups or clusters. Ideal data for such an analysis would yield clusters so obvious that they could be picked out, at least in small scale cases, without the need for complicated mathematical techniques and without a precise definition of the term 'cluster'. In practice, however, things are rarely so straightforward, so there has been a great proliferation of clustering techniques over the last three decades or so" [19, p.10]. "The second half of the twentieth century has seen a dramatic increase in the number of numerical classification techniques available. This growth has largely paralleled the development of high-speed computers, such machines being needed to undertake the large amounts of arithmetic generally involved. As well as an increase in the variety of numerical classification methods, a similar expansion has taken place in the areas of their applications. Nowadays such techniques are used in fields as disparate as archaeology and psychiatry, and market research and astronomy" [19, p.4].

"A number of names have been applied to these methods depending largely on the area of application. *Numerical taxonomy* is generally used in biology. In psychology the term *Q analysis* is sometimes employed. In the artificial intelligence literature *unsupervised pattern recognition* is common. In other areas *clumping* and *grouping* have also been used occasionally. Nowadays however, the most common generic term is *cluster analysis*…" [19, p.4]

1.3 Motivations for Clustering

What are the motivations for clustering? MacKay [20] lists reasons for the usefulness of clustering. First, a good clustering has useful predictive power. Man for instance creates internal model of objects he had already encountered so that when he encounters another object that belongs to a group he already knew he can predict more or less the behavior of the new object based on the characteristics of the class that are already stored in his memory. Second, clustering is a great help in communicating because it is a form of *lossy* compression. We don't have to describe all the attributes of an object, say a tree, in order to describe it. We simply label a tree as a tree instead of saying "that twenty feet tall thing with leaves, trunk and roots" every time we see a tree. Third, failure of cluster model to predict will highlight objects that need our special attention. For example, when our internal model predicts that green things (e.g. plants) don't move and we encounter a green thing (e.g. a snake) that moved, then things that failed our prediction should deserve our full attention. Fourth reason for clustering is its usefulness as model of learning in neural systems. As an example, the K-Means algorithm is a model of a type of learning called *competitive learning algorithm* in neural systems.

1.4 K-Means Algorithm

K-Means clustering algorithm is a simple but very powerful technique of partitioning data sets. "The term k-means is attributed to MacQueen, whose first implementation involved only a single pass through the data" [21]. In MacQueen's method, as the data sample is presented, the cluster to which the data belongs is determined and updated. This method is also called Online K-Means [22]. In the 1980s, S.P. Lloyd introduced a variant called *Lloyd's algorithm* [7] [8], also called Batch K-Means [22]. In Lloyd's method, the whole data should be present from the beginning. Arrival of new data is not considered. Originally, Lloyd's version dealt with scalar quantities only. Later, it was expanded to include multi-dimensional data and called the *generalized Lloyd's algorithm (GLA)* [3, 10].

K-Means or Lloyd's algorithm [23] is probably the most popular clustering method among the algorithms that are based on minimizing a formal objective function. Throughout this paper, whenever K-Means is mentioned, it is shorthand for the standard K-Means or the generalized Lloyd's algorithm.

Given a set of N data points, d-dimensional real space, $\boldsymbol{R}^d$, and a positive integer K, the task is to determine those K points in $\boldsymbol{R}^d$ called *cluster centers* (also known as *codebook vectors, codevectors* or *codewords)* such that the mean squared distance of each data point to its nearest center is minimized. This algorithm partitions N data points into K disjoint subsets $\boldsymbol{S}_j$ containing $\boldsymbol{N}_j$ data points so as to minimize the sum-of-squares criterion

$$J = \sum_{j=1}^{K} \sum_{n \in S_j} \left| x_n - \mu_j \right|^2$$

where $\boldsymbol{x}_n$ is a vector representing the nth data point and $\boldsymbol{\mu}_j$ is the geometric centroid of the data points in $\boldsymbol{S}_j$.

In general, the basic data for clustering is a matrix X, where the rows represent the objects under investigation and the columns represent the features or descriptions of the objects; that is,

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2p} \\ \vdots & & & & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \cdots & x_{np} \end{bmatrix}.$$

The algorithm generally does not achieve a global minimum of $\boldsymbol{J}$ over the assignments. In fact, since the algorithm uses discrete assignment rather than a set of continuous parameters, the "minimum" it reaches cannot even be properly called a local minimum. The solution for getting the best and the most efficient clustering using K-Means is known to be an NP-hard problem. The number of distinct partitions of n individuals into g non-empty groups is given by the formula [19, p.94]:

$$N(n, g) = \frac{1}{g!} \sum_{i=0}^{g} (-1)^{g-1} \binom{g}{i} i^n$$

Even with our present computational capabilities the numbers involved are so astounding that it is not practical to do complete enumeration of each of the possible partitions.

Everett [19, p.93] lists some numerical examples to illustrate the magnitude of the problem.

$N(15, 3) = 2,375,101$

$N(20, 4) = 45,232,115,901$

$N(25, 8) = 690,223,721,118,368,580$

$N(20, 4) = 10^{68}$, where $N(n, g)$ is the number of distinct partitions of n data points into g non-empty clusters.

The tendency of K-Means algorithm to easily get trapped in a local minimum is well documented. Despite its limitations, the algorithm is used fairly frequently because it is relatively easy to implement [4].

1.5 New K-Means Algorithm Using Perturbed Codevectors

In this study, a different way of computing the codevectors in order to optimize the performance of K-Means is discussed. K-Means algorithm uses the previous codevectors as the seeds for computing the next cluster centroids. These new cluster centroids, in turn, are going to be the new codevectors that will be used as the new input for the next iteration. In this paper, a new method is introduced in which the computed centroids will not be used as the new seeds for computing the next cluster centroids. Instead, some perturbations are added to the centroids and these perturbed codevectors are used as the seeds to compute the next codevectors. This cycle continues until the codevectors do not change their positions. The perturbations are added in the direction of the momentum of

the codevectors. The direction of the momentum is approximated by the line joining the current and previous codevectors. This is a way, it is hoped, of accelerating the convergence of the codevectors to their final locations. An objective of this study is to measure the optimum amounts of perturbations that are needed to get maximum improvement in performance. This would decrease overall iterations needed to reach convergence.

II. Literature Review

2.1 K-means as a Vector Quantizer

J. MacQueen introduced the K-Means algorithm in 1967 and it has evolved into one of the most popular clustering algorithm used with scientific and industrial applications. The Batched version was developed by S.P. Lloyd (originally, it dealt with scalar data only). It was later expanded into a general form called *generalized Lloyd's algorithm (GLA)* to handle multi-dimensional data [7, 10].

In this paper, the term K-Means refers to generalized Lloyd's algorithm. K-means was originally developed for *vector quantization (VQ)* purposes. VQ is a lossy data compression method based on the process of mapping a large set of vectors into a smaller set of vectors [5]. VQ is one of the Kohonen networks, developed by a Finnish academician Prof. Teuvo Kohonen, one of the most prolific and pre-eminent researchers in the field of neurocomputing. As a quantizer, VQ must satisfy the two necessary criteria, namely, the nearest-neighbor condition and the centroid condition for codevectors.

K-Means also is closely related to the Linde-Buzo-Gray *(LBG)* algorithm which is another implementation of vector quantization. Teuvo Kohonen [6] gives a good theoretical discussion on the mathematical foundation of *VQ*. T. Kohonen describes *VQ* as a classical signal-approximation that forms a quantized approximation to the

distribution of the input vectors $x \in \boldsymbol{R}^d$, by the use of so-called *codebook vectors* $m_i \in \boldsymbol{R}^d$, $i = 1,2,\ldots,k$. When the "*codebook*" is chosen, approximating x means finding the *codebook vector* m_c closest to x (in the input space) by using some distance metric, usually by Euclidean metric:

$$||x - m_c|| = \min\{||x - m_i||\}, \text{ or}$$

$$c = \arg\min\{||x - m_i||\}$$

A kind of optimal selection of the m_i minimizes the mean expected square of the quantization error, oftentimes called the distortion measure and is defined by the following:

$$E = \int ||x - m_c||^2 \, p(x)dx,$$

where the integral is evaluated throughout the metric x space, dx is a shorthand notation for the d-dimensional volume differential of the integration space, and $p(x)$ is the probability density function of x. For general $p(x)$, there is no close-form solution to m_i, hence, one must resort to iterative approximation schemes. A mathematically rigorous discussion by Teuvo Kohonen on the derivation of the VQ algorithm is found in pages 59-62 of [6].

K-means is an iterative implementation of VQ. Given a set of data points and initial cluster centers (codevectors):

1. Allocate data point to their nearest codevectors. Any tie-breaking mechanism can be adopted for points located at boundaries between codevectors. This is equivalent to forming Voronoi cells. This also satisfies the *nearest neighbor condition* of a VQ (Please refer to Figure1).

The red stars in Figure 1 are the codevectors that represent the data points. These codevectors belong to a set called the *codebook* that satisfy VQ optimality requirement of nearest-neighbor condition as well as the centroid requirement.

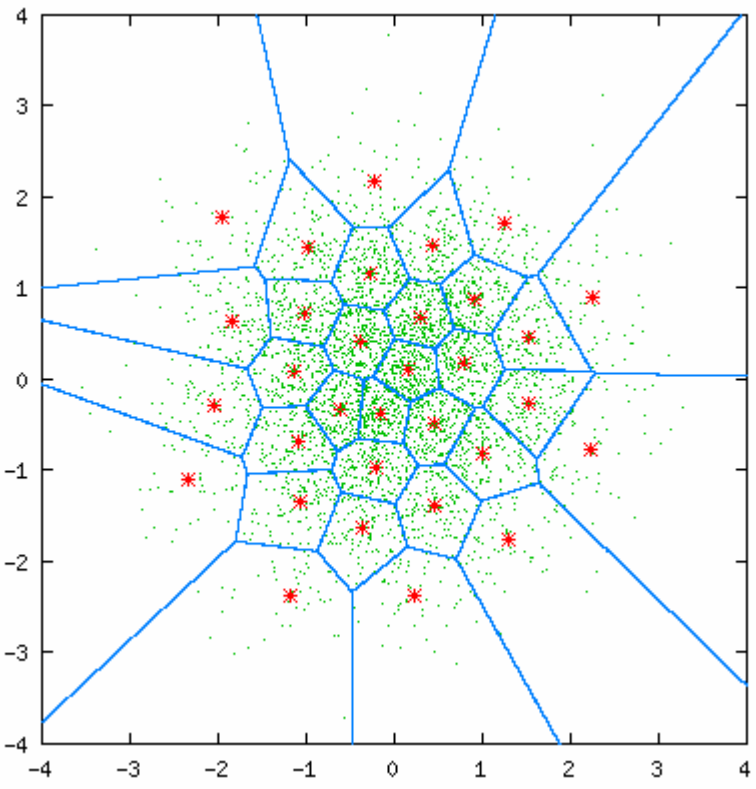

Figure 1. 2 -dimensional Voronoi cell after nearest neighbor computation (From http://www.data-compression.com/vq.html)

2. Compute centroids of the data points in each Voronoi cell. These centroids are going to be the new codevector. This satisfies the *centroid condition* for the codevectors of a VQ.
3. Repeat 1 and 2 until there is no more change of membership among the data points. When the process converges the codevectors of the Voronoi cells that are generated and satisfy both the nearest neighbor condition and the centroid condition of codevectors requirements.

The final centroids are going to be the vector quantizer for the data points. In other words, we use these centroids to represent the distribution of the data points.

2.2 General Methods of Making K-Means Efficient

Many ways have been proposed to make K-Means algorithm more efficient. Choosing good initial centers (codevectors) is one important process in improving efficiency as well as in the avoidance of potential problems during iteration of this algorithm. There are three basic problems that may arise when the initial cluster centers are poorly chosen. These are the problem of dead centers (these are centers with no members or data points), local minima and center redundancy. Dead centers are usually located between two active centers or outside the range of the data points. A way to avoid this problem is by selecting the initial centers randomly and to set the range of the random values within the range of the data sets. The problem of poor local minima may be avoided by using such algorithms as simulated annealing, stochastic gradient descent, genetic algorithms, etc. These, however, may entail more involved computation. The problem of center redundancy arises when there is too much cluster centers so this may be addressed by

making sure that there's no overcrowding of the centers so that there is no tendency for these centers to be very close or in the same position with each other.

2.3 Use of K-d tree Data Structure

Pelleg and Moore [11] presented a way of improving the efficiency of K-Means by using a data structure called kd-tree. In their paper, they propose storing the data points information in kd-tree database for the purpose of minimizing the K-Means nearest-neighbor query on these data points. They made use of the fact that kd-tree nodes can store large number of points. So instead of updating the centroids point by point, they used the concept of updating in bulk by using the information already stored in the kd-tree nodes. Thus, using the statistics stored in the nodes they were able to reduce the number of arithmetic operations needed to update the cluster centroids. This algorithm is good to use in cases where databases are large. Kanungo, et al made a more detailed analysis of this kd-tree-based algorithm and presented a data-sensitive analysis as the separation between clusters increases. They proved that as the separation increases, the algorithm runs more efficiently. They called their method *The Filtering Algorithm* [8].

2.4 One Iteration Method

Bradley, Fayyad and Reina [12] proposed an algorithm that requires only one pass on the entire data set. Their work is based on identifying three regions: regions that are compressible, regions that must be maintained in memory and regions that are discardable. Their work focuses on the problem of clustering large databases under the confines of limited buffer memory. These databases are too large for loading in the RAM.

The algorithm is based on the idea of storing only the important portions of the database while summarizing those that are of least importance. The process works as follows:

1. Get a sample point from the database to be put in the buffer (RAM).
2. Update model based on the recent sample.
3. Then, decide if the singleton data is to be retained in the buffer (data point is to be used all the time), be discarded or be reduced and summarized into a more efficient representation.
4. Finally, check if stopping criteria has been satisfied. If not satisfied, go back to step1.

Even though the algorithm requires only one pass through data set, the overhead necessary unfortunately, makes this algorithm slower than the standard K-Means. Its real benefit though, is its usefulness in working with large databases on a limited RAM. Fredrik Farnstrom, et al [13] made further refinements on Bradley et al's algorithm by simplifying some of its process.

2.5 A Method That Works in O(c) Time

A fast scaling-up method proposed by Hulten and Domingos [14] works not just in linear time with respect to the total data points but in sublinear time (constant time). This is done by limiting the quantity of data that are used at each step. This algorithm uses sampling methods based on *Hoeffding inequality* and other statistical bounds. During each iteration, sample size is increased in such a way as to maintain the loss bound from the multi-pass K-Means [3].

2.6 Better Codebook Through Pertubation of the Codevectors

2.6.1 Method of Simulated Annealing

Another problem being addressed for improving K-Means is in the design of better codebook (codevectors). K-Means is a *descent algorithm* which means that its performance (in terms of distortion error decreases) improves every time iteration is performed. The problem with a descent algorithm is that it easily gets trapped in a local minimum. Kirkpatrick et al. [15] introduced the concept called *Simulated Annealing (SA)* to remedy the problem of local minimum. The analogy comes from the fields of metallurgy and materials science in which a metal is slowly heated and then slowly cooled so that the system at any time is approximately at thermodynamic equilibrium. If cooling is done in a fast manner (quenched), the system will form defects because it freezes out in metastable states or it is in local minimum energy state. In simulated annealing, the energy of the system is defined by its distortion function. Since the K-Means is a descent algorithm (i.e. its distortion function is monotonic) it can easily get trapped in a local minimum at the end iterations. This is similar to the way a hot metal behaves when it is quenched. During the iterations, the K-Means system is being "quenched" or "cooled" very fast thus, trapping the distortion function in a local minimum. Simulated annealing (SA) remedies this situation by making sure that the system "cooling" behaves in a non-monotonic way. This is done by perturbing the state of the system at every iteration so that the distortion function (or energy E) will not decrease in a monotonic way [16]. In the case of the SA, the perturbation is accepted when the perturbation results in a net decrease of energy ($\Delta E < 0$). When the $\Delta E > 0$, perturbation is accepted with the probability $\exp(-\Delta E/T)$, where T is the temperature or

variance of the noise [17]. In SA, perturbation is introduced by corrupting the input training data by some noise before the nearest neighbor (NN) repartitioning is done. This is called *encoder perturbation*. Another perturbation is done when the codevectors are reconstructed to satisfy centroid condition and this is called *decoder perturbation*. Due to its complexity SA require more computational time when compared to K-means.

2.6.2 Method of Stochastic Relaxation

A more general form of the *Simulated Annealing (SA)* algorithm was proposed by Zeger, et al [16, 17, 18] and it is called *Stochastic Relaxation Scheme (SR)*. SR seeks to provide improvements in terms of computation efficiency as well as in ease of implementation. The main difference between SR and SA methods is that in the case of SR, codevector perturbations are accepted unconditionally, unlike that of the SA where perturbation is accepted only with probability $\exp(-\Delta E/T)$ whenever $\Delta E > 0$. Another way of saying this is that the condition imposed by SA in accepting codevector perturbations is "relaxed", hence, the name Stochastic Relaxation Scheme. As in SA, since SR is the general form, encoder (before nearest neighbor computation) and decoder (at centroid computation) perturbations are applied to the codevectors when using SR method. SR is an attempt to reduce the combinatorial complexity of the SA but still provide a close to globally optimal solution.

III. Directed Perturbation of the Codevectors

3.1 Background

This paper borrows from some of the concepts of codevector perturbation (alteration of the codevectors parameters) as espoused in Simulated Annealing (SA) and Stochastic Relaxation (SR) techniques. A more directed way of adding perturbation to the codevectors will be presented. Recall from 2.6.1 and 2.6.2 that in the case SA and SR, encoder perturbations as well as decoder perturbations are done by adding random noise on the parameters. The purpose of this noise alteration is to add energy to the system in such a way as to avoid the monotonic descent of the codevectors as they converge so they will not be trapped in local minima.

Here is the summary of the SA and SR rules. For the SA algorithm, the following rules apply:

1. Accept proposed perturbation conditionally.
2. Simultaneously either perturb all encoder or all decoder parameters.
3. Do a repartitioning and centroid computation.

The SR algorithm on the other hand has the following rules:

1. Accept proposed perturbation unconditionally.
2. Simultaneously either perturb all encoder or all decoder parameters.
3. Do a repartitioning and centroid computation.

Rule 3 is just the usual standard or "greedy" K-Means rule. The proposed new algorithm is a special case of the SR algorithm because it satisfies all three of its general rules. The difference is in the way it perturbs the parameters. In the proposed method, perturbations are added to the codevectors not in any random direction and amount but in a more directed way towards the general direction where the codevectors are going. Just like in case of the standard K-Means algorithm, the convergence using the new method also proceeds in a monotonically decreasing manner. Figure 2 shows the various positions, represented by green circles, of codevector m^i through all of its n iterations. It starts as an initial seed m^i_0 at the start of iteration and ends as the final codevector m^i_n on the n^{th} iteration. We connect all these codevector (circles) positions at various iterations t by a curve C.

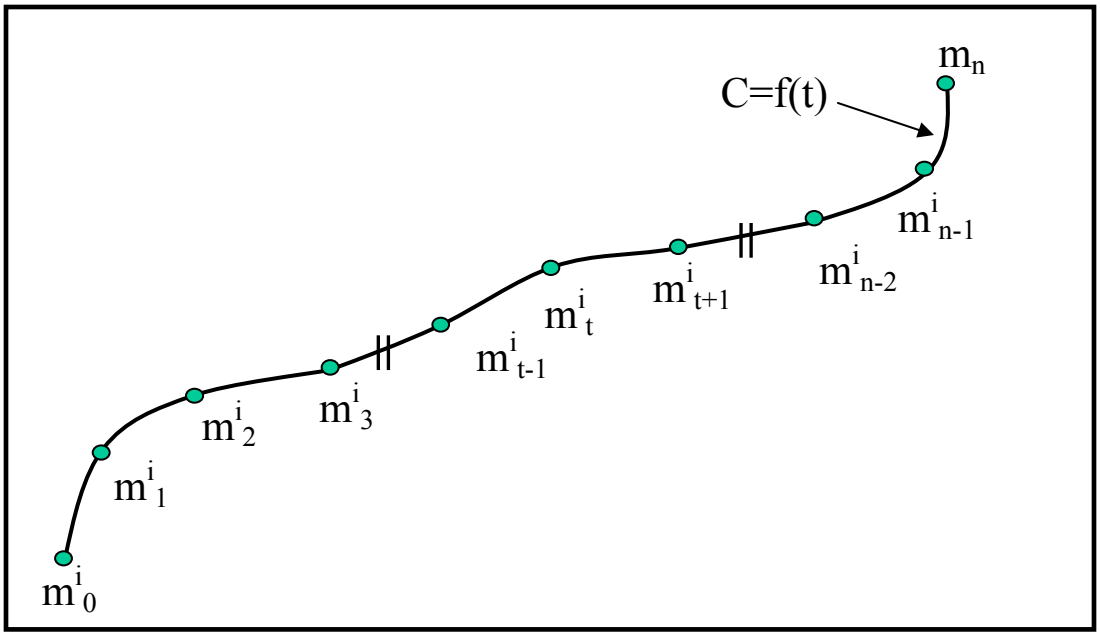

Figure 2. Various states of a codevector through its iterations.

Suppose m^i_{t-1} is the i^{th} *codevector* location in the vector space after iteration t-1 and m^i_t is the i^{th} codevector location after iteration t. (Please refer to Figure 3). Then C passes through m^i_{t-1} and m^i_t as the codevector descends monotonically towards convergence.

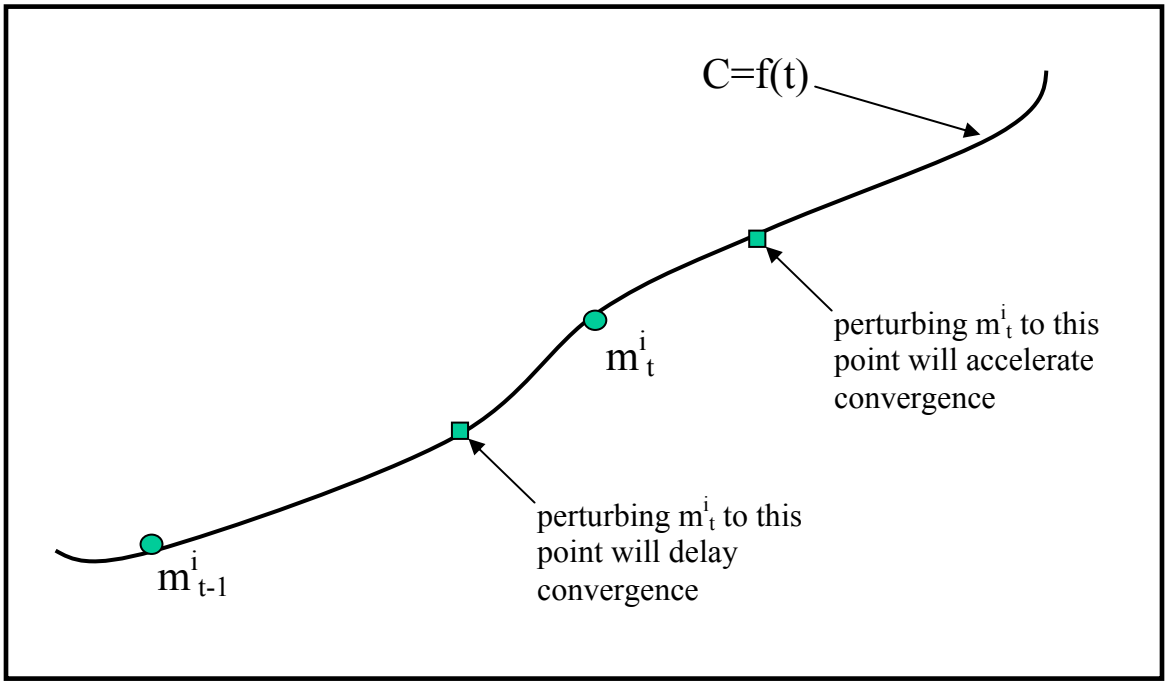

Figure 3. Possible locations after codevector perturbation.

Now, suppose perturbation is introduced to m^i_t such that this codevector will cause to be located somewhere on the curve C between m^i_{t-1} and m^i_t . By doing this, it is hypothesized that convergence of m^i to its final location may be delayed because we are decreasing its momentum thereby increasing the total number iterations needed for convergence.

On the other hand, if we apply perturbation such that m^i moves beyond m^i_t (towards the right) on the curve C, then it is hypothesized that by applying just the right amount of

perturbation we may be accelerating the convergence of m^i to its equilibrium state by giving it just a little more energy, thereby decreasing the total number of iterations needed to reach a stable state. We need to know where to locate m^i after perturbing it on the curve C so that we can use this perturbed value as the seed value for the next iteration which we hope, will result in faster convergence.

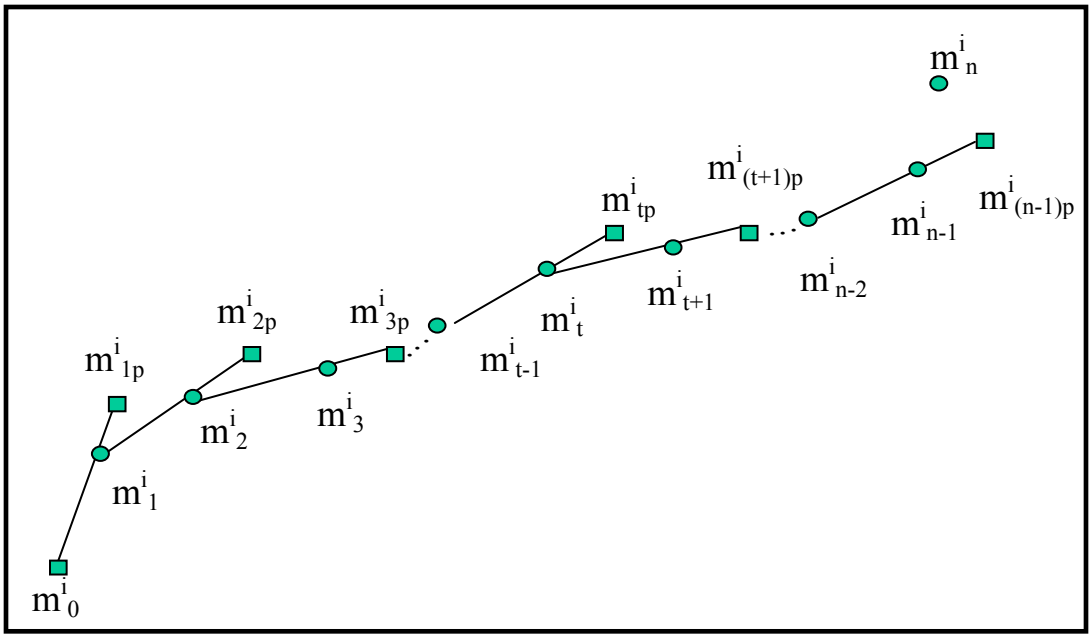

Figure 4. Computed centroids (circles) perturbed to new locations (squares).

For simplicity, curve C can be replaced by creating a straight line passing through codevectors m^i_{t-1} and m^i_t . Then, locate the perturbed value of m^i_t on this line and call it m^i_{tp} .Then, use m^i_{tp} as the seed for the next iteration (i.e. at iteration t+1). Thus, the line connecting the previous codevector with the current codevector value is used to approximate the general direction of the codevector momentum as the system descends to local minimum. It is suspected that the perturbed codevector must be located on the line beyond the current codevector if improvement in iteration is to be realized. The task

therefore is to find the right amount of perturbation that should be added to the codevector in order to enhance performance of K-Means in terms of reduction in the total number of iterations at the end of each run. Figure 4 shows the locations of the computed centroids (circles) and the locations of perturbed centroids (squares) throughout all iterations.

The objective of SA and SR algorithms is mainly to find global minimum solutions. Due to complex overhead in the computation of the SA and SR algorithms, these schemes are not really able to address the problem of efficiency in terms of the speed of iteration. Unlike in the case of SA and SR, the purpose of the new method is not to arrive at a global minimum solution but to a local minimum only (as in standard K-Means) but with the distinct advantage reduction of the total iterations at the end of run.

3.2 Experimental Methodology

3.2.1 Preliminaries

Let m^i_t and m^i_{t-1} be the i_{th} codevector locations after iterations t-1 and t, respectively (Figure 5), furthermore, let

D = the distance between points m^i_t and m^i_{t-1},

ΔD = the change in D after perturbing codevector m^i_t,

m^i_p = the new point after perturbing m^i_t, and

$$\alpha = (D + \Delta D) / D. \qquad (1)$$

It is clear from Figure 5 that m^i_p is computed according to this equation

$$m^i_p = m^i_{t-1} + \alpha * D. \qquad (2)$$

When $\Delta D=0$ (i.e., no perturbation), then $\alpha = 1.0$, and it follows from equation 2 that $m^i_p = m^i_{t-1} + D = m^i_t$. This reduces the algorithm to the standard K-Means at scale $\alpha = 1.0$, hence it is clear that standard K-Means is just a special case of the Perturbed K-Means.

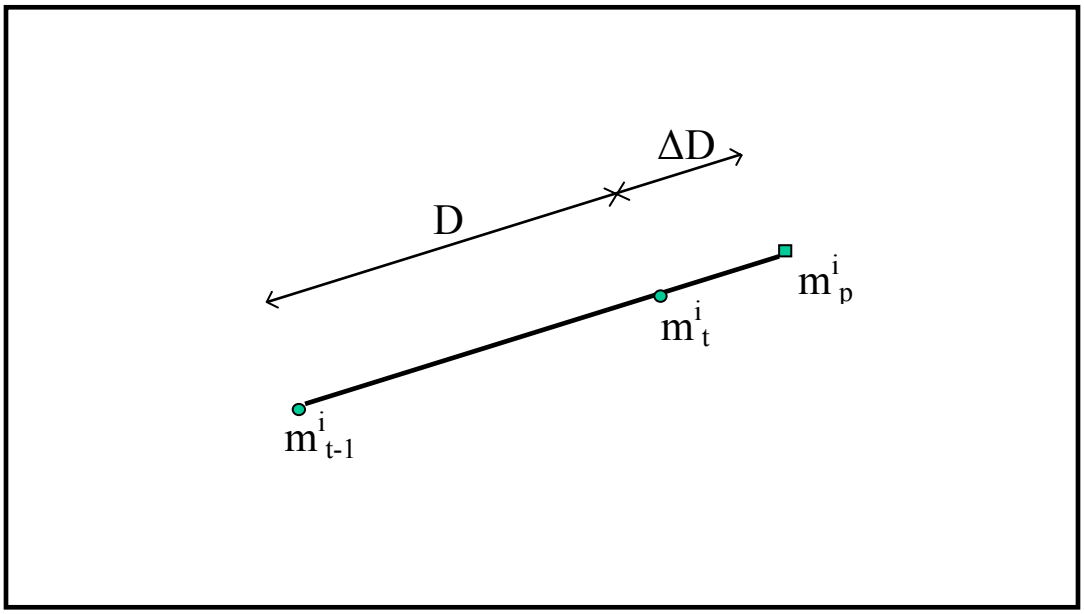

Figure 5. Scale

The goal in this study is to find different values of the scale α such that efficiency is achieved in terms of reduction in total iteration for every run of the algorithm. We want to test the following:

1. if $0 \leq \alpha < 1.0$, then perturbation will increase the iterations needed towards convergence, thereby delaying it, and
2. if $1.0 < \alpha \leq 2.0$, then the perturbation will decrease iterations needed towards convergence, at some values of α in this range, effectively accelerating convergence.

3. Furthermore, we want to show that the final locations of the perturbed codevectors are equivalent in quality to the final locations of the unperturbed (standard K-Means) codevectors.

3.2.2 The Perturbed K-Means Program

A program named KM.java has been designed using the Perturbed K-Means algorithm that will test the correctness of the hypotheses. KM.java will read its input from a file as well as write its output on a different file. This program has the following input and output properties:

Program Input:

1. Input File Name – This is a text file containing a set of multivariate data points arranged in rows and columns.
2. Integer K – This is an integer representing the predetermined number of clusters centers or classes the Perturbed algorithm will partition the data points into.
4. Output File Name – This is the file where we want our output to go to.

Program Output:

1. Final codevectors.
2. Total number of iterations after complete run.

The source code is found on page – of the Appendices.

3.2.3 Description of Data

The data that were used as inputs to the program are combinations of real world data as well as synthetically-generated data coming from various online data libraries. These data contain points whose features are described by numerical descriptions only. In other

words, phonetic alphabets and other non-numerical symbols are excluded in the description of these data points. These multivariate data are arranged in rows and columns. Columns are delimited by space. The m^{th} row represents the m^{th} datapoint and the n^{th} column of the m^{th} row is the dimension or features of the m^{th} point. The data were processed "as is". No weights were added to any of its dimension. The program was run on data with the following ranges dimensionality:

Dimensions: 2- 60.

3.2.4 Initial Cluster Seeds

The integer K values that were tested have these values,

K : {2, 4, 8, 16, 32}.

We wanted our initial codebook or codevectors to be located randomly around the datapoints so that they are more or less good representatives of the data points. This was easily achieved by using pseudo-random number generators from standard library of our programming language choice.

3.2.5 Scales

The values of the scale α that were tested are the following:

α : {0.6, 0.8, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 2.0}.

For each data set the same values of randomly generated initial cluster seeds were used to run all the α values. This ensured uniformity of criteria when comparing the behavior of Perturbed K-Means through various values of the scale.

IV. Tests and Results

4.1 The Experiment

The program KM.java was run using the following data sets as our input namely: 2d.txt, 5Tec.txt, 14Housing.txt, 60Synt.txt. These data were taken from various free databases that are available online. We wanted to test the consistency of the improvement in performance across different dimensions so data that have wide range of dimensionality were represented. In this case, data with dimensions 2, 5, 14 and 60, respectively, were tested. For each data set and for a particular value of K, the Perturbed program KM.java made 40 complete runs producing 40 complete iterations results. For every run it computed K random seeds for the initial codevectors. These initial codevectors were used to compute the total iterations for complete runs for each particular scale value α. 40 total iterations values (not necessarily unique) were generated for each value of α. To get better indication of the iterations' spread and skewness, the iterations were graphed in boxplot graphs. For each particular input data used and K value, the total iterations are graphed against different α values. From these graphs it can easily be seen if there are improvements in performance as the different scale values are compared. These results are summarized in Figures A-B, pages 41-50 of the Appendices.

4.2 Other Metrics

In order to compare the quality of the perturbed points (codevectors) that we are getting to that of the unperturbed points the within class scatter between the two, the class

within-class scatter resulting from K-Means and with that resulting from Perturbed K-means, were compared. Recall that it is the *within-class scatter metric*s that are being minimized when the K-Means algorithm is run. We want to know if the result of running the Perturbed K-Means is essentially the same as that of the Unperturbed K-Means by comparing if the within-class metrics of the two algorithms converge to more or less the same values at the completion of their respective runs. Suppose it is predetermined that there are K clusters in the data set. Furthermore, let μ_i be the codevector for cluster i (i = 1, 2,…, K) and C_i be the number of samples within cluster i, then the within class scatter matrix W is defined as

$$W = \sum_{i=1}^{K} \sum_{j=1}^{Ci} (x_{ij} - \mu_i)(x_{ij} - \mu_i)^{\mathrm{T}}, \; x_{ij} \in C_i.$$

The scatter should become progressively smaller and smaller every time K-Means goes through the cycles of nearest-neighbor and centroid computations during iterations.

It is also helpful to know the percentage difference between the labeled points of the K-Means in comparison to the Perturbed K-Means algorithm as it goes through different stages of iteration until complete convergence. If A are classes of points whose cluster centers are the codevectors computed using standard K-Means and B are the classes of points whose centers are computed using the Perturbed K-Means, then this percent difference is computed as follows:

(|(A-B) U (B-A)| / |(A U B)|) *100% .

Using this measurement will help us see how fast these two algorithm converge with each other and when in the iteration stage the Perturbed K-Means codevectors contain essentially identical points as that the ordinary K-Means.

4.3 Discussion of Results

This study was undertaken to answer the main question: Can improvements in performance of the K-Means algorithm be realized when the perturbed centroid points are used instead of the standard unperturbed centroids? There is overwhelming evidence, based on the data from experiments, that indeed, we can realize great improvements in the performance when the Perturbed K-Means algorithm is run on data of various sizes, dimensionality and number of K partitions using certain scale values instead of the standard K-Means. There are substantial improvements in performance in terms of reduced total iterations by up to 63% compared to the standard K-Means.

At what scale values do we get these great improvements? Please refer to Figure 6.

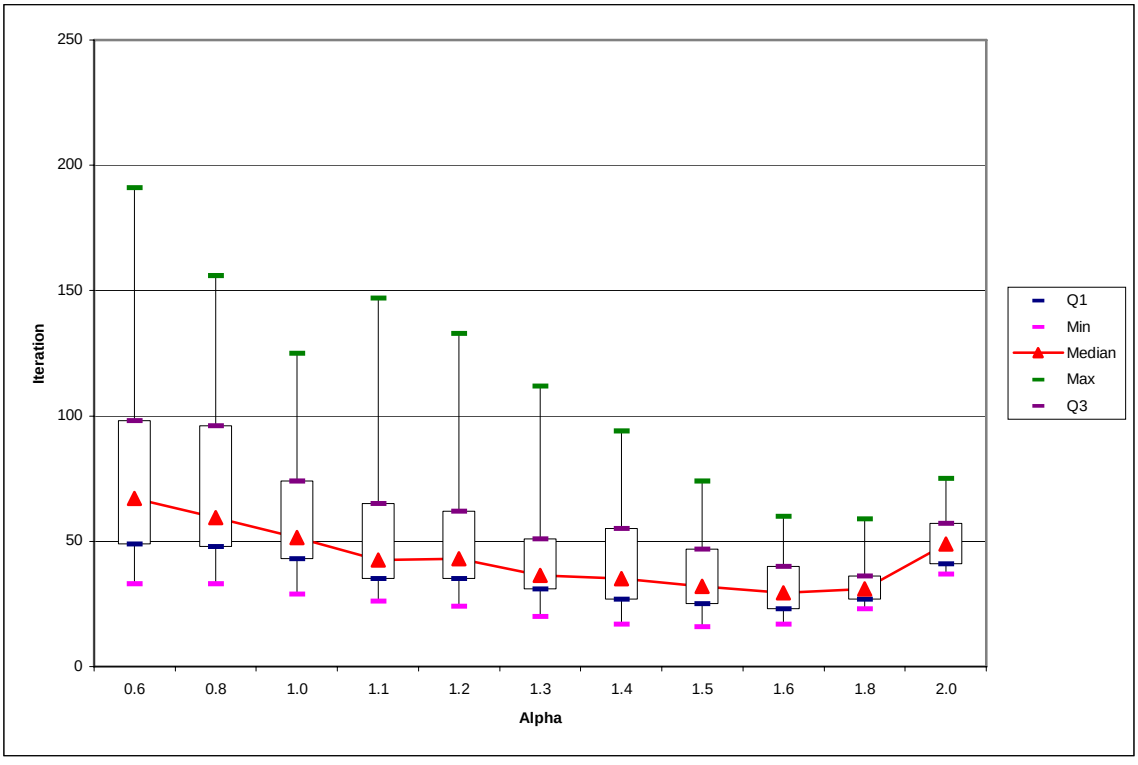

Figure 6. Box Plot of iteration against scale alpha of 2d.txt (16744 datapoints, dimension=2 and K=32) after 40 runs.

Here we have a box plot of a 2-dimensional data containing 16,744 points with K=32. To get the box plot values of the iterations we ran the algorithm 40 times. We wanted to compare how the performances of the different scale values of the Perturbed K-Means compare to the standard K-Means. A cursory look at the box plot graph clearly shows that at scale values less than 1.0 there is dramatic decrease in performance based on the increase in the median values of the iterations of the scales $\alpha = 0.8$ and 0.6 when compared to K-Means ($\alpha = 1.0$). This is also indicated by the negative slope of the lines connecting the median values of the iterations from scale value 0.6 to 1.0. This deterioration in performance is repeated consistently in the rest of the data that were tested (Figures B1-B4, C1-C4 and D1-D5). The only exception was in Figures B5 and C5. In both of these cases, the value K is 2.0. At this low K value, the iterations are very low anyway because the data points are divided into 2 classes only. These 2 exceptions represent only 10 percent of the cases. In the 90 percent of the cases, total iterations tended to increase when the scale values become less than 1. This is not unexpected since as the scale value α at range $0 \leq \alpha < 1.0$ we tend to perturb the codevectors backward toward or near the original seed values. The extreme case is with $\alpha = 0$, in which case we are essentially returning the codevector to its original seed values causing the algorithm to never converge.

The previous scenario is contrasted with the next in which the values of the scale α are in the range $1.0 < \alpha \leq 2.0$. Here we got more interesting results. In the above example (Figure 6), the medians of the iterations of the different values of α when compared to K-Means ($\alpha = 1.0$) are lower. This reduction means improvement in performance. The

lowest total iterations was with the use of $\alpha = 1.6$, where we improve the performance by 42.72% compared to K-Means (Table 1 summarizes improvement in performances for various combinations of scales and K values). The only exception is when the scale $\alpha = 2.0$, which in this case the median value is about the same as that obtained by using K-Means. The box plot values of the iterations at $\alpha = 2.0$ are confined to a narrow range compared to K-Means ($\alpha = 2.0$). These box plot values is in fact still better than the standard K-Means when we compare the middle 50 percent (box) of the iterations data. The top whiskers (maximum iterations values) of the box plot for $\alpha = 1.1$ and 1.2 may bother some because they are longer than that of the standard K-Means. This feature of the box plot may tend to distort the actual distribution of the data. The box plots for these scale values were supplemented by their histograms (Figure 7) so that the actual distributions of the points (iterations) can be seen clearly. The *frequency* (vertical axis) of the histogram represents the number of occurrences of the iterations on a range of iterations (horizontal axis) after 40 runs of the Perturbed K-Means algorithm. From these histograms it is easy to see that the outliers do not represent accurately these particular tail values. In fact, in the outlier points (histograms B and C) represent only one occurrence (2.5%) of the values. The vast majority of the points are still superior to K-Means (histogram A).

Looking at the performance for the other values of K, (K=16, 8, 4, Figures A2, A3, and A4, respectively) the same trend as in Figure 6 (K=32) still holds true. There are clearly improvements in performance vis-a-vis K-Means in the range of scale: $1.1 \leq \alpha \leq 1.8$. The improvements are from 31.25% to 46.03%. The box plot for the case $\alpha = 2.0$

show deterioration in performance compared to the previous scales but then the performance is not really worse off compared to that of the K-Means. The iterations for

Dataset	Scales with improvements in performance relative to 1.0	Greatest Improvement	K	Figure
2d.txt 16744 points 2 dimensions	1.1, 1.2, 1.3, 1.4, 1.5, **1.6**, 1.8	42.72%	32	A1
	1.1, 1.2, 1.3, 1.4, 1.5, **1.6**, 1.8	41.25%	16	A2
	1.1, 1.2, 1.3, **1.4**, 1.5, 1.6, 1.8	46.03%	8	A3
	1.1, 1.2, 1.3, **1.4**, 1.5, 1.6, 1.8	31.25%	4	A4
	1.1	8.30%	2	A5
5Tec.txt 6000 points 5 dimensions	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, **1.8**	53.68%	32	B1
	1.1, 1.2, 1.3, 1.4, 1.5, 1.6, **1.8**	63.27%	16	B2
	1.1, 1.2, 1.3, 1.4, 1.5, **1.6**, 1.8	56.36%	8	B3
	1.1, 1.2, **1.3**, 1.4, 1.5	43.75%	4	B4
	None	0%	2	B5
14Housing.txt 506 points 14 dimensions	1.3, **1.4**, 1.5, 1.8	12%	32	C1
	1.2, 1.3, 1.4, **1.5**, 1.6, 1.8	16.67%	16	C2
	1.1, 1.2, 1.3, **1.4**, 1.5, 1.6, 1.8	18.18%	8	C3
	1.2, **1.3, 1.4, 1.5, 1.6**	14.29%	4	C4
	None	0%	2	C5
SyntheticControl.txt 600 points 60 dimensions	1.1, 1.2, **1.3, 1.4**, 1.5, 1.6	20%	32	D1
	1.3, 1.4, 1.5, 1.6	10%	16	D2
	1.2, 1.3, 1.4, 1.5, 1.6	11.11%	8	D3
	1.3, 1.4	14.29%	4	D4
	None	0%	2	D5

Table 1. Summary of the improvements in performance of various scales and K combinations. Scales in red give maximum improvement.

this case are also more confined to a narrower range of values. For the case K=2, the box plots (Figures A5) show that the iterations are confined to very narrow range for all the scales and their performances are practically equivalent to each other.

The performance of Perturbed algorithm when ran with the data 5Tec.txt (6000 data points with 5 dimensions for each point) is similar to the performance using the previous data set (2d.txt). The pronounced difference is in the rapid deterioration in

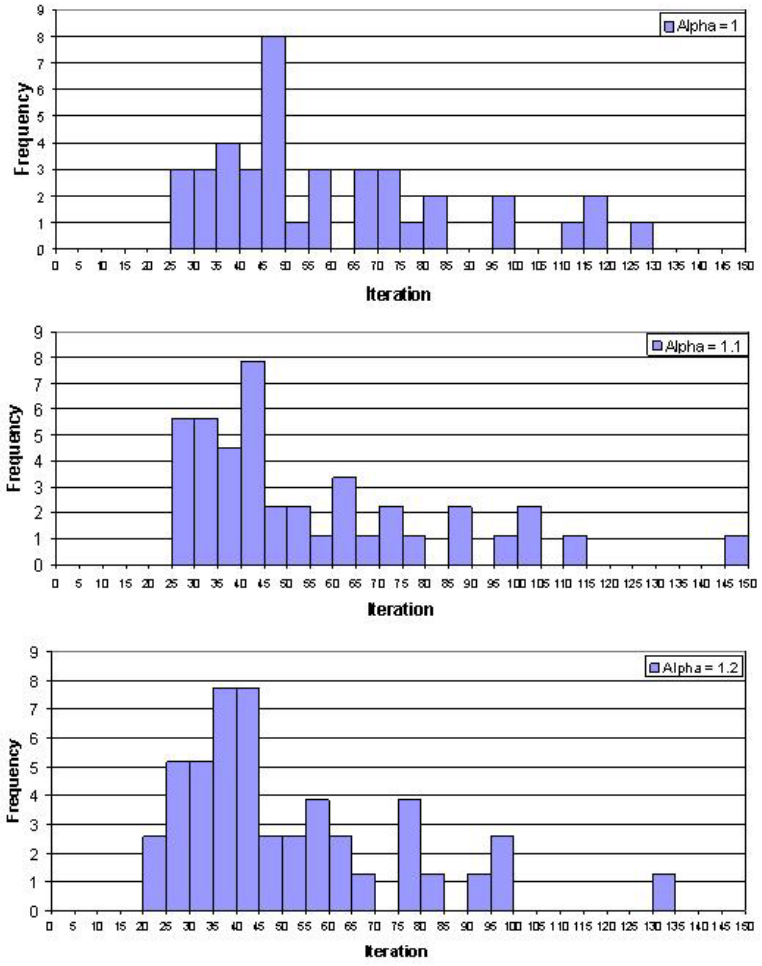

Figure 7. Histograms of 2d.txt (Dimension=2, K=32) to complement the boxplots of Fig.6 for α = 1, 1.1, 1.2. The *frequency* (vertical axis) of the histogram represents the number of occurrences of the iterations on a range of iterations (horizontal axis) after 40 runs of the Perturbed K-Means algorithm.

performance as the scale value becomes α = 2.0 (Figures B1, B2, B3 and B4). The performance was generally good for most of the combinations of K and α values. The highest performance was obtained using α = 1.8 at K=16 which shows 63.27% improvement (Figure B2). Again as in the case of the previous data set, the performances of the Perturbed algorithm using the different scales for K=2 are equivalent to each other, in other words, there's no improvement.

Going to the third data set, 14Housing.txt (506 datapoints with 14 dimensions per point): for K=32 (Figure C1), we realize improvement over K-Means for the scales 1.2, 1.3, 1.4, 1.5 and slightly on 1.8 based on the median values as well as the position of boxes (or middle 50 percent of the data). The highest performance was exhibited by the combination $\alpha = 1.4$ and K=8 which improved by 18.18%. The median for scale $\alpha = 1.1$ is the same as in K-Means but there is more spread of the middle 50% of the points so it's hard to see if there's net improvement here. There is deterioration in performance as we go to the scales 1.6-2.0. For K=16 (Figure C2), based again on the median and the position of the box (middle 50%) there is improvement in performance for the scales 1.1-1.6, slight improvement for 1.8 and deterioration at 2.0. In the case of K=8 (Figure C3), the scales 1.1-1.8 all manifest improvement. Deterioration occurs again at $\alpha = 2.0$. Looking at K=4 (Figure C4), scales 1.2-1.6 represent improvements at 2.0 where there is deterioration. For K=2 (Figure C5), again as in the previous two data change in the scale does not really confer benefit.

For our last data set, which SyntheticControl.txt (600 datapoints with 60 dimensions), for K=32 (Figure D1), clearly there are benefits for Perturbed K-Means in scale range 1.1-1.6. Greatest improvement was 20% at $\alpha = 1.3$, 1.4 and K=32. Then, performance declines at scales 1.8 and 2.0. For K=16 (Figure D2), the scales 1.3-1.6 are good. Then, declines are at 1.8 and 2.0. The same is true for K=8 (Figure D3): improvements at 1.2-1.6 and deteriorations occur at 1.8 and 2.0. At K=4 (Figure D4), basing only on the median improvement occurs only at scales 1.3 and 1.4. On closer inspection of the boxes, the scales 1.1, 1.2 and 1.5 are still good values because the boxes are narrower and are generally located below that of the box for scale 1.0. Performance decreases at 2.0.

Lastly, for K=2 (Figure D5), sets the Perturbed K-Means improvement over the ordinary K-Means is zero.

4.4 Equivalence of Labeled Points

We also wanted to know if the final codevectors obtained from the Perturbed K-Means are of the same or closer quality to those obtained by using K-Means. The measurements of the within-class scatters of the two algorithms' labeled points should become closer in value to each other as they are near the end of their runs. The results were a pleasant surprise. Figures E1-E5 and Figures F1-F5 are the graphs of the ratios of the of the within-class scatters of the Perturbed K-Means (Wp) and the original K-Means (Wc) plotted against the iterations from start to the end of runs of the two data sets, namely 2d.txt (16,744 data points, 2 dimensions) and SyntheticControl.txt (600 data points, 60 dimensions). It is easy to see from these graphs that as the iterations increase, the ratios Wp/Wc slowly approach 1 and at the end of the runs these ratios were practically equal to 1 in all cases. This means that the quality of the labeled points achieved using Perturbed K-Means was comparable to the labeled points from using the standard K-Means. Figures G1-G5 also show the actual values of the within-class scatter of SyntheticControl.txt for the different scale and K values and they all converge to the same value which equal that of K-Means (α =1.0). Figure 8 below is a

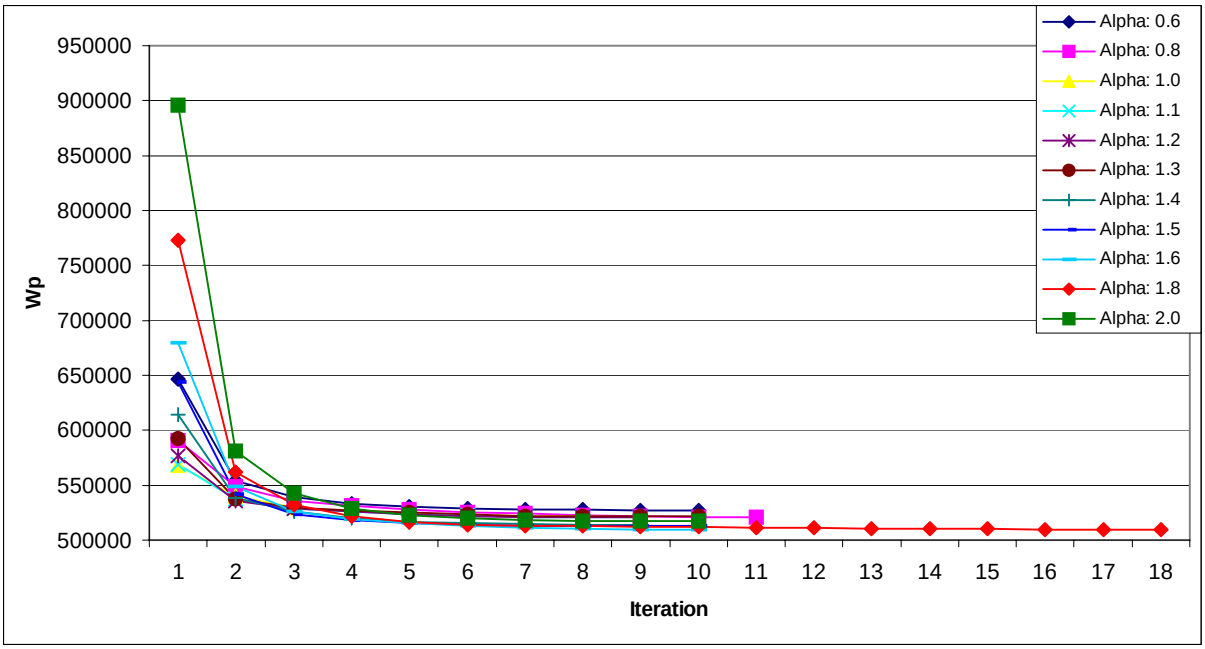

Figure 8. Plot of within class scatter of Perturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=32).

Alpha	0.6	0.8	1	1.1	1.2	1.3	1.4	1.5	1.6	1.8	2
Wp	526670	521294	521319	521095	520997	521615	513753	512648	509735	509787	517342
Total Iterations after 1 run	10	11	9	9	9	10	9	10	10	18	10

Table 2. Actual values of within-class scatters and corresponding scale and total iterations after 1 run of the Perturbed K-Means algorithm for data set SyntheticControl.txt (600 data points, dimension=60 and K=32).

reproduction of Figures G1 from the Appendix. The actual values of the within-class scatter for the graphs are found on Table 2. Here, the actual within-class scatter for the standard K-Means ($\alpha = 1$) is 521,319 at the end of its run. The actual within-class scatters for the different scales are all very close to the value of the standard K-Means. These differences vary from 0.06% ($\alpha = 1.3$) to 2.22% ($\alpha = 1.6$). This means that the Perturbed K-Means descend to more or less the same local minimum.

4.5 Percent Difference Between Labeled Points

We also want to know at what point in the cycle of iterations the Perturbed K-Means become identical to the standard K-Means in the sense that their particular codevectors own practically the same clusters of data points. This comparison can be achieved by comparing the percentage difference between the labeled points of the Perturbed K-Means and the labeled points of the standard K-Means. Figures H1-H5 and Figures I1-I5 are the plots of these differences for the data sets 2d.txt and SyntheticControl.txt, respectively. As we can see from these plots, initially at the beginning stages of the iterations the difference between the labeled points are high, in some cases approaching 35% (e.g. Figure I1). But these differences quickly dissipate towards zero percent value, sometimes during the middle stages in the iterations, and especially towards the end stages when the differences become zero or almost zero. This zero difference happens because as the iterations progress, the effect of the scales become less and less pronounced and the Perturbed K-Means codevectors become identical with the standard K-Means.

V. Conclusions and Future Work

5.1 Conclusions

Based on the four data sets that we tested, it has been shown that it is certainly possible to realize improvements on performance of the K-Means by applying some scaling mechanism to the codevectors. This modified version of the K-Means algorithm, called Perturbed K-Means, was able to obtain these enhanced performances without sacrificing the quality of the final codevectors. Using the Perturbed K-Means on the four data sets above, we were able to achieve performance on K-Means algorithm by as much 63.27%. The labeled points that were derived by using the Perturbed algorithm were the same, quality-wise, from the labeled points derived by using the standard K-Means. The benefit is faster convergence because of reduced total iterations needed to reach local minimum.

This benefit was achieved by applying perturbations to the codevectors that were computed after a cycle of nearest-neighbor and centroid computations. These perturbed points were used as the new cluster seeds for next cycle of the nearest-neighbor and the centroid computations. The general locations of the perturbed codevectors were along the lines connecting the current codevectors and the previous codevectors. These lines become our general estimator of the directions of the momenta of the codevectors as they lose energy during converge. Furthermore, the amounts of perturbations were scaled in direct proportion to the length of the distance between the current unperturbed codevector and the previous codevector.

The Perturbed K-Means algorithm was also superior in performance across varying degrees of data dimensionality. And it was effective for different values K initial centers that were used with the exception of K=2, at which value, the Perturbed K-Means does not seem to exhibit much improvement over the standard K-Means. The algorithm does not guarantee superior result every time it is used. But by picking just the right kind of scale to use the Perturbed algorithm is on average better than the standard K-Means.

5.2 Future Work

This work is by no means an exhaustive. Much work lies ahead and there is certainly more room for improvement. The Perturbed method needs to be tested using data sets with different size properties in order for us to validate if the improvements we got were consistent. It needs to be run using larger datasets. Future studies may also be done to test its sensitivity to other parameters. For instance, how does this algorithm behave when, instead of using the usual Euclidean metric, we use other distance measurements like the city block, the Mahalanobis or the Canberra metrics? Future study can also be done to test the algorithm for its sensitivity to some data set properties. For example is it sensitive to the ratio of data size to K? Does higher ratio reduce the effectiveness of the algorithm? Note that at K=2, which means higher ratio of data size to K, the algorithm becomes almost useless.

Studies can also be done to determine if the Perturbed algorithm may be combined with other performance improvement techniques that have been proposed and have been proved successful by other investigators in order to further fine tune this algorithm.

REFERENCES

[1] Peng Yuqing, Hou Xiangdan and Liu Shang. The K-Means Clustering Algorithm Based on Density and Ant Colony. *IEEE Int. Conf. Neural Networks & Signal Processing*, pages 457-460, December 14-17, 2003, Nanjing China.

[2] Cheng-Fa Tsai, Han-Chang Wu, and Chun-Wei Tsai. A New Data Clustering Approach for Data Mining in Large Databases. *Proceedings of the International Symposium on Parallel Architectures*, pages 278-283, 2002.

[3] Anjan Goswami, Ruoming Jin and Gagan Agrawal. Fast and Exact Out-of-Core K-Means Clustering. *Proceedings of the Fourth IEEE International Conference on Data Mining(ICDM'04),* pages 83-90, 2004.

[4] Fang Yuan, Zeng-hui Meng, Hong-Xia Zhang and Chun-Ru Dong. A New Algorithm To Get the Initial Centroids. *Proceedings of the Third International Conference on Machine Learning and Cybernetics,* pages 1191-1193, Shanghai, August 26-29, 2004.

[5] Gurmeet Singh, Ashish Panda, Saurav Bhattacharyya and Thambipillai Srikanthan. Vector Quantization Techniques for GMM Based Speaker Verification. *ICASSP 2003,* pages II 65-II 68.

[6] Teuvo Kohonen. *Self-Organizing Maps*, third ed., pages 59-62. Germany, Springer-Verlag Heidelberg, 2001.

[7] Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory,* Pages 129-137, Vol. IT-28, No. 2, March 1982.

[8] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An Efficient k-Means Clustering Algoritm: Analysis and Implementation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, VOL. 24, NO. 7, pages 881-892, JULY 2002.

[9] Leonid I. Perlovsky. *Neural Networks and Intellect: Using Model-Based Concepts*. New York, Oxford university Press, 2001.

[10] Lowell L. Winger. Linearly Constrained Generalized Lloyd Algorithm for Virtual Codebook Vector Quantization, *Proceedings on International Conference on Image Processing*, Volume 2, Pages 171-174, Vancouver, BC, Canada, 2000.

[11] Dan Pelleg and Andrew Moore. Accelerating Exact k-means Algorithms with Geometric Reasoning. *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages: 277 – 281, San Diego, California, 1999.

[12] Paul S. Bradley, Usama M. Fayyad and Cory A. Reyna. Scaling Clustering Algorithms to Large Databases. *Proceedings of the 4th International Conference on Knowledge Discovery & Data Mining (KDD98),* R.Agrawal, P. Stolorz and G. Piatetsky-Shapiro (eds.), pp. 9-15. AAAI Press, Menlo Park , CA, 1998.

[13] Fredrik Farnstrom, James Lewis and Charles Elkan. Scalability for Clustering Algorithms Revisited. *ACM SIGKDD Explorations Newsletter*, Volume 2, Issue 1, pages: 51- 57, June 2000.

[14] Geoff Hulten and Pedro Domingos. Mining Complex Models from Arbitrarily Large Databases in Constant Time. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, Pages 525-531. ACM Press July 2002.

[15] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, Volume 220 Number 4598 Pages 671-680, 13 May 1983.

[16] Eyal Yair, Kenneth Zeger and Allen Gersho. Competitive Learning and Soft Competition for Vector Quantizer Design. *IEEE Transactions on Signal Processing*, Vol. 40 No.2, Pages 294-309, February 1992.

[17] K. Zeger and A. Gersho. Stochastic Relaxation Algorithm for Improved Vector Quantiser Design. *Electronic Letters*, Vol. 25 No. 14, Pages 896-898, 6th July 1989.

[18] Kenneth Zeger, Jacques Vaisey, and Allen Gersho. Globally Optimal Vector Quantizer Design by Stochastic Relaxation. *IEEE Transactions on Signal Processing*, Vol. 40 No. 2, Pages 310-322, February 1992.

[19] Brian S. Everett. *Cluster Analysis,* Third Edition. John Wiley & Sons Inc., 605 Third Avenue, NewYork.

[20] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms,* Pages 284-285. Cambridge University Press, 2003.

[21] James Theiler and Galen Gisler. A Contiguity-Enhanced K-means Clustering Algorithm for Unsupervised Multispectral Image Segmentation. *Proc. SPIE 3159*, Pages 108-118, 1997.

[22] H. Ando, S. Suzuki, and T. Fujita. Unsupervised Visual Learning of Three-Dimensional Objects Using a Modular Network Architecture. *Neural Networks 12*, Pages 1037-1051, 1999.

[23] Nargess Memarsadeghi, David M. Mount, Nathan S. Netanyahu, and Jacqueline Le Moigne. A Fast Implementation of the ISOCLUS Algorithm. *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'03)*, Toulouse, France, July 21-25, 2003, Vol. III, Pages 2057-2059.

[24] Hichem Frigui and Raghu Krishnapuram. A Robust Competitive Clustering Algorithm With Applications in Computer Vision. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, VOL. 21, NO. 5, Pages 450-465, May 1999.

[25] Bot, R.S., Yi-fang Brook Wu, Xin Chen and Quanzhi Li. A Hybrid Classifier Approach for Web Retrieved Documents Classification. *Proceedings of the International Conference on Information Technology : Coding and Computing (ITCC 2004)*, Volume 1, Pages 326-330.

APPENDICES

Glossary

A priori – Proceeding from a known or assumed cause to a necessarily related effect.

Bayesian network – It is a directed graph of nodes representing variables and arcs representing dependence relations among the variables.

Codebook – A set of codevectors

Codevector – A point in the Euclidean space $\mathbf{R}^n$ that represent a subset of points in $\mathbf{R}^n$.

Euclidean metric – the function $d : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}$ that assigns to any two vectors in Euclidean n-space $\boldsymbol{x} = (x_1,\ldots,x_n)$ and $\boldsymbol{y} = (y_1,\ldots,y_n)$ the number

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2},$$

and so gives the "standard" distance between any two vectors in $\mathbf{R}^n$.

Global minimum – The smallest overall value of a set, function, etc., over its entire range.

Heuristic – In computer science, it is a technique designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem.

Hoeffding's inequality – It is a result in probability theory that gives an upper bound on the probability for the sum of random variables to deviate from its expected value.

K-means algorithm – An algorithm for partitioning (or clustering) N data points into K disjoint subsets S_j containing N_j data points so as to minimize the sum-of-squares criterion

$$J = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - \mu_j|^2,$$

where x_n is a vector representing the nth data point and μ_j is the geometric centroid of the data points in S_j.

Local minimum – A local minimum, also called a relative minimum, is a minimum within some neighborhood that need not be (but may be) a global minimum.

Perturbation – Alteration of the state of a system by addition of noise.

Simulated annealing – It is a method of searching for global minimum in a general system in which an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) is used.

Unsupervised learning – A method of machine learning where a model is fit to observations. It is distinguished from supervised learning by the fact that there is not *a priori* output.

Vector – An element of a vector space. In the commonly encountered vector space $\mathbf{R}^n$ (i.e., Euclidean n-space), a vector is given by n coordinates and can be specified as $(A_1,A_2,\ldots,A_n)$.

Vector quantizer – It maps *n-dimensional* vectors in the vector space $\mathbf{R}^n$ into a finite set of vectors $Y = \{y_i: i = 1, 2, ..., N\}$.

Voronoi diagram – The partitioning of a plane with n points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than to any other. A Voronoi diagram is sometimes also known as a Dirichlet tessellation.

Figure A1

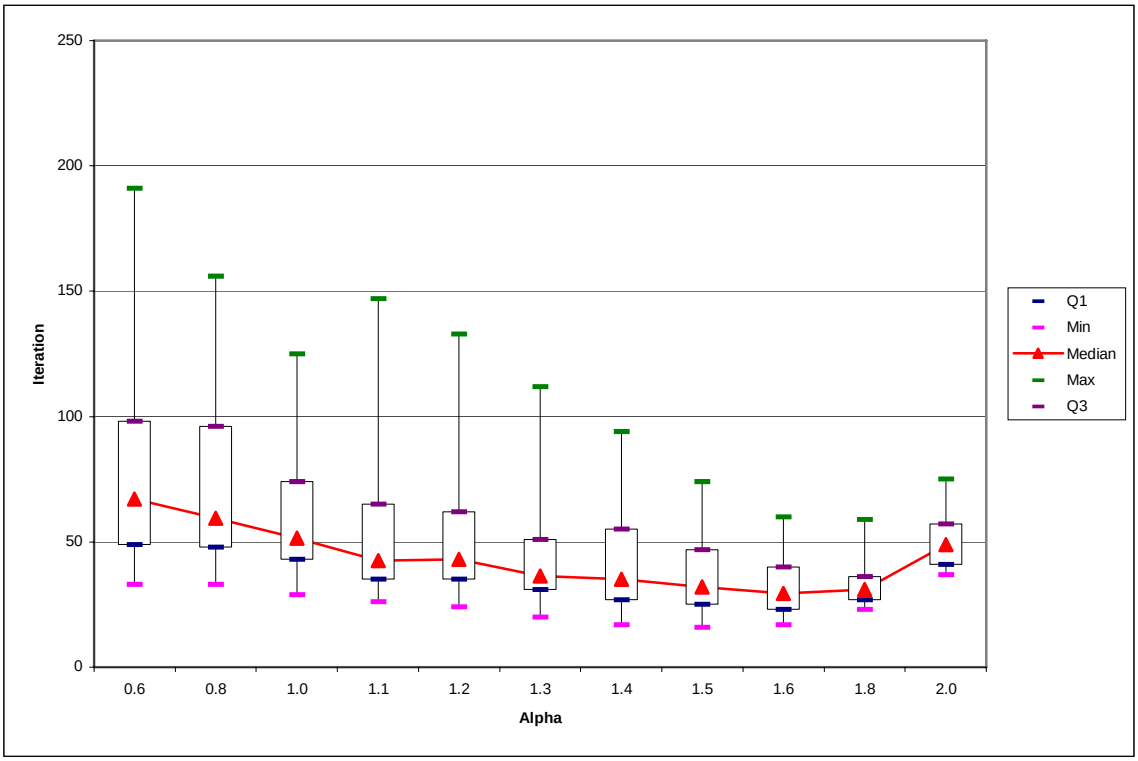

Figure A1. Box Plot of iteration against scale alpha of 2d.txt (16744 datapoints, dimension=2 and K=32) after 40 runs.

Figure A2

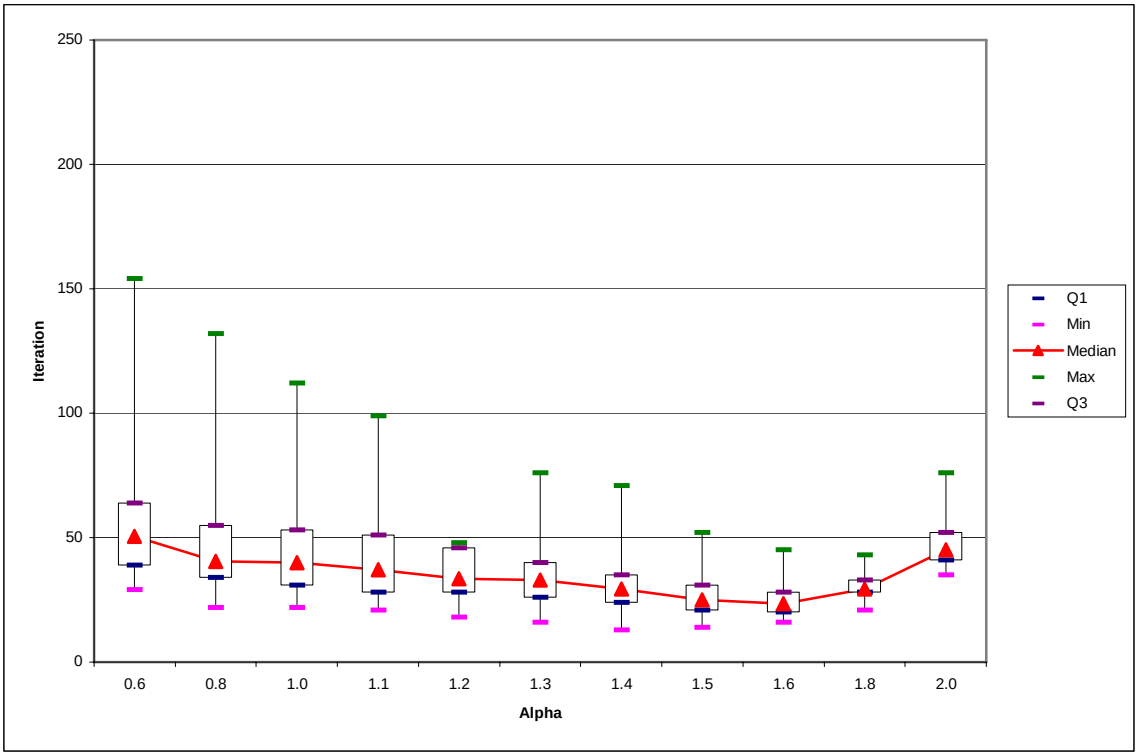

Figure A2. Box Plot of iteration against scale alpha of 2d.txt (16744 datapoints, dimension=2 and K=16) after 40 runs.

Figure A3

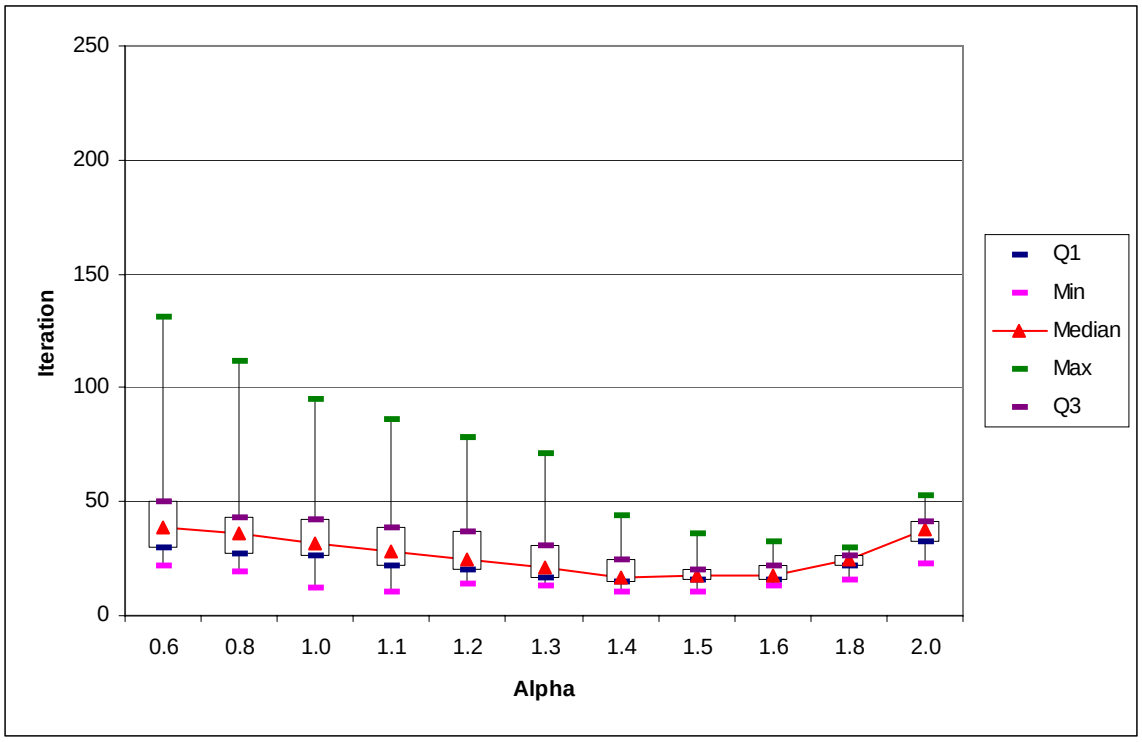

Figure A3. Box Plot of iteration against scale alpha of 2d.txt (16744 datapoints, dimension=2 and K=8) after 40 runs.

Figure A4

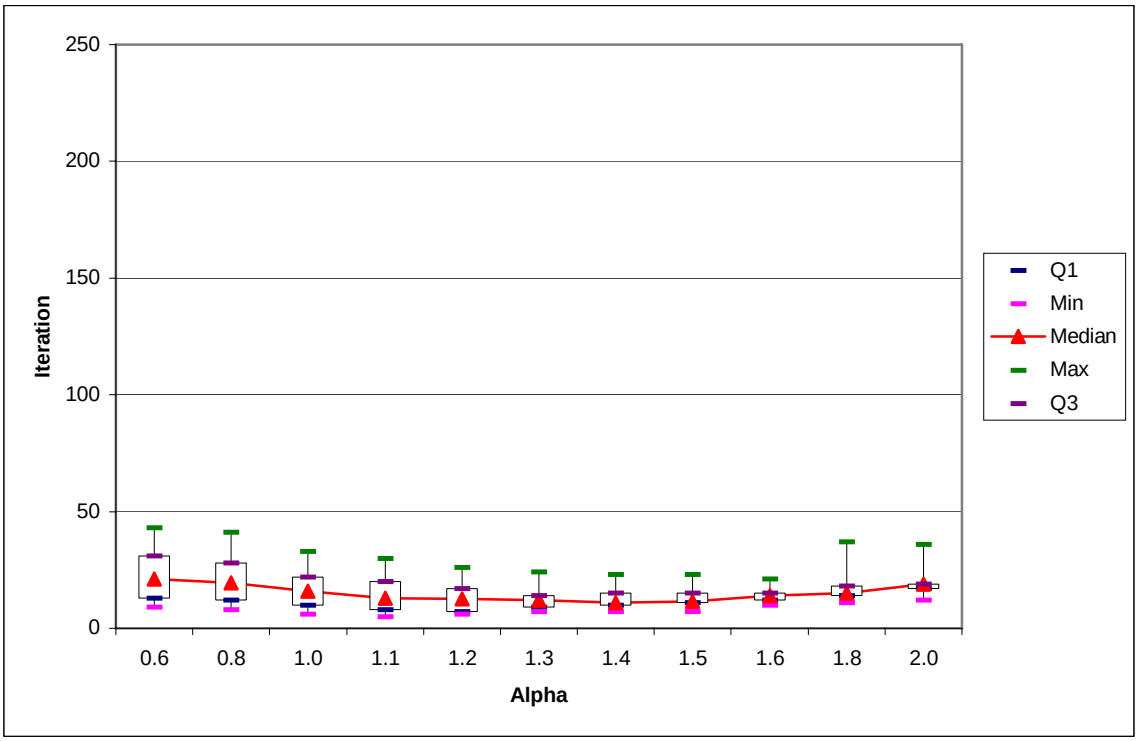

Figure A4. Box Plot of iteration against scale alpha of 2d.txt (16744 datapoints, dimension=2 and K=4) after 40 runs.

Figure A5

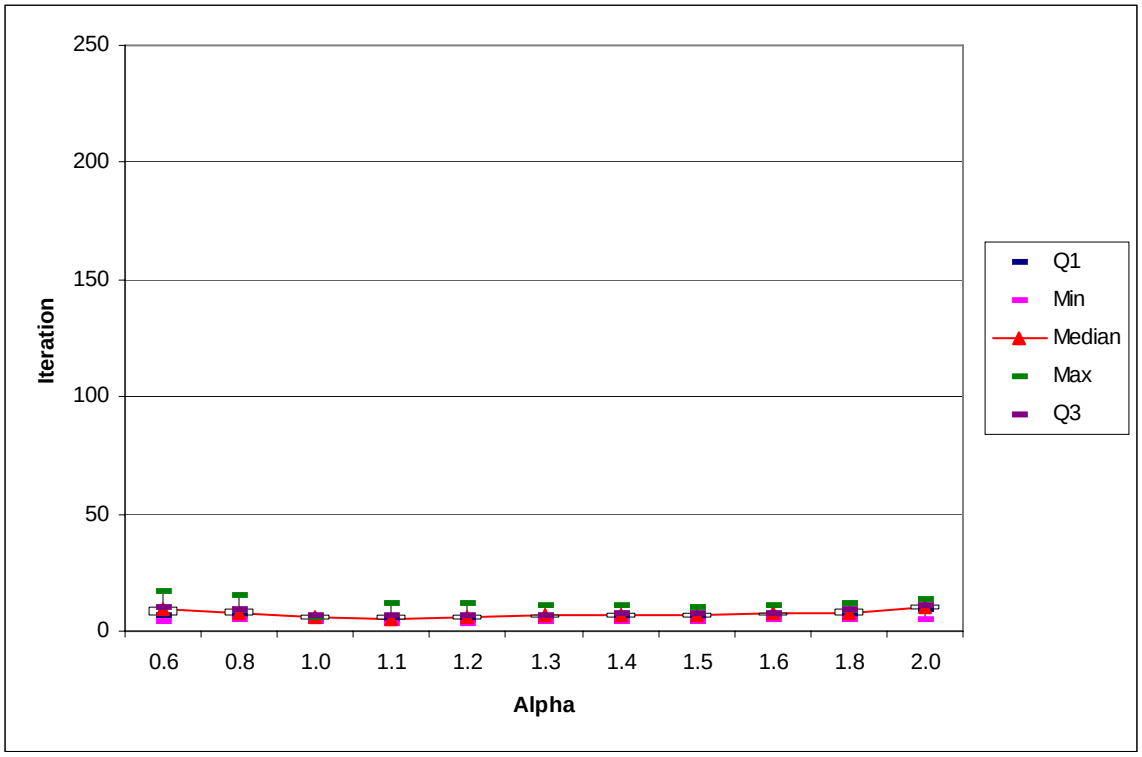

Figure A5. Box Plot of iteration against scale alpha of 2d.txt (16744 datapoints, dimension=2 and K=2) after 40 runs.

Figure B1

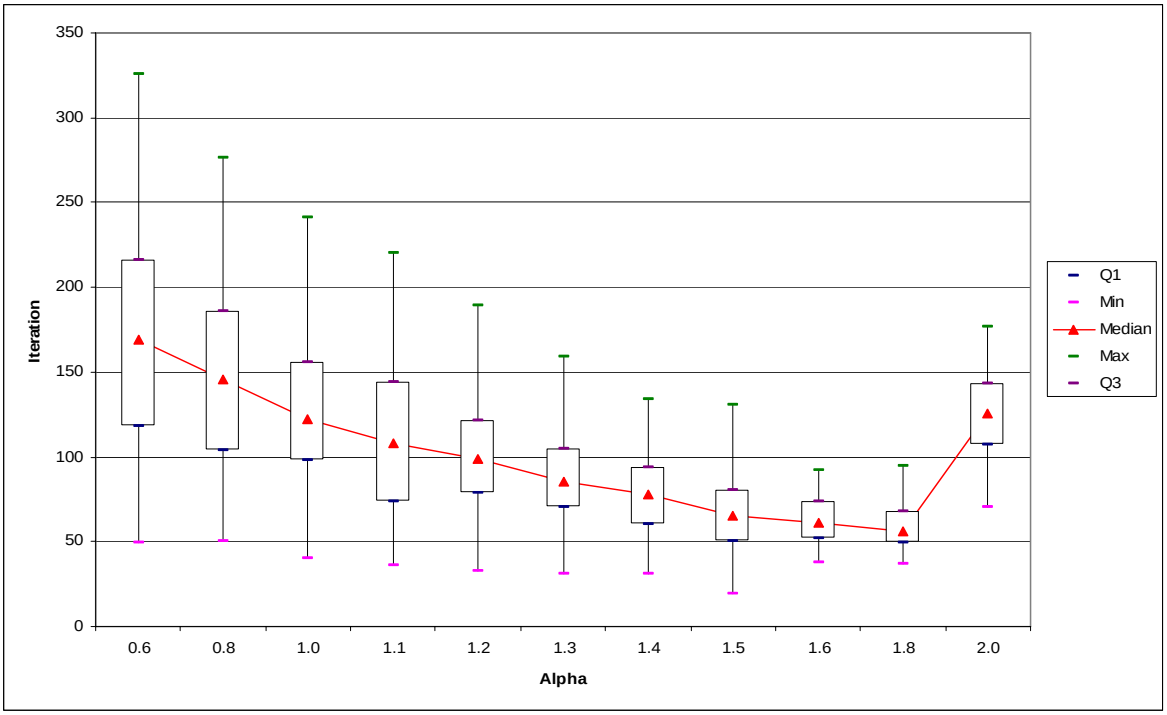

Figure B1. Box plot of iteration against scale alpha of 5Tec.txt (6000 datapoints, dimension=5and K=32) after 40 runs.

Figure B2

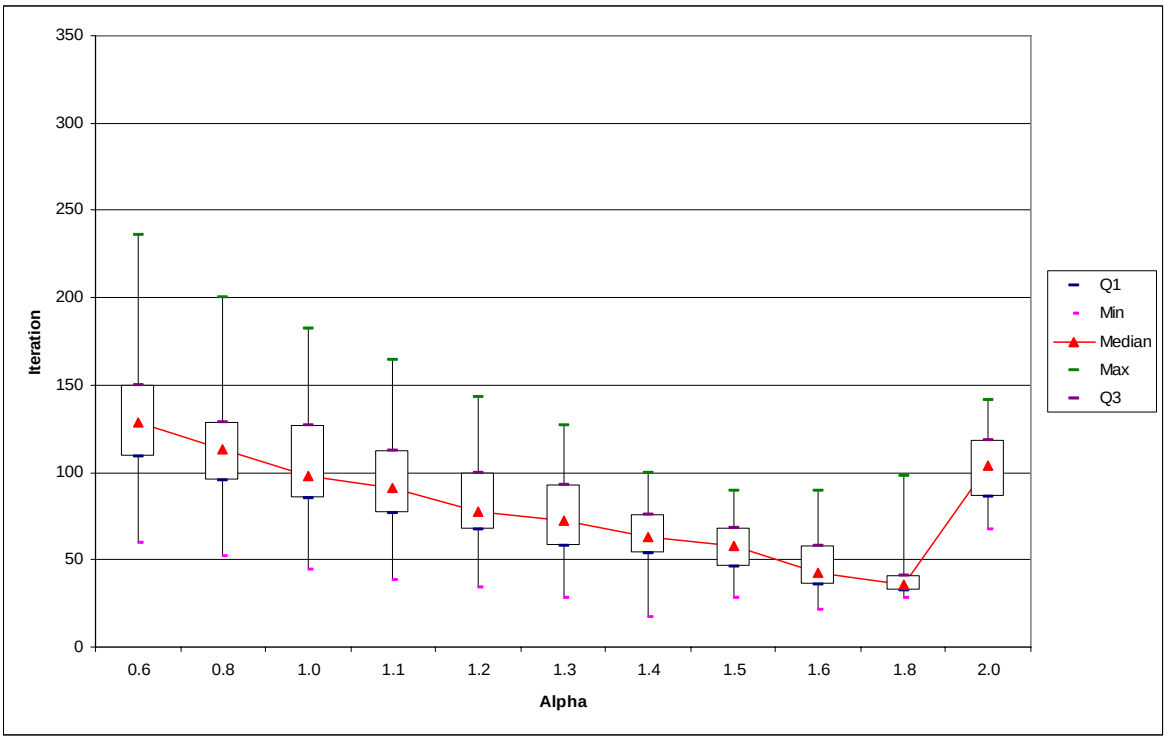

Figure B2. Box plot of iteration against scale alpha of 5Tec.txt (6000 datapoints, dimension=5and K=16) after 40 runs.

Figure B3

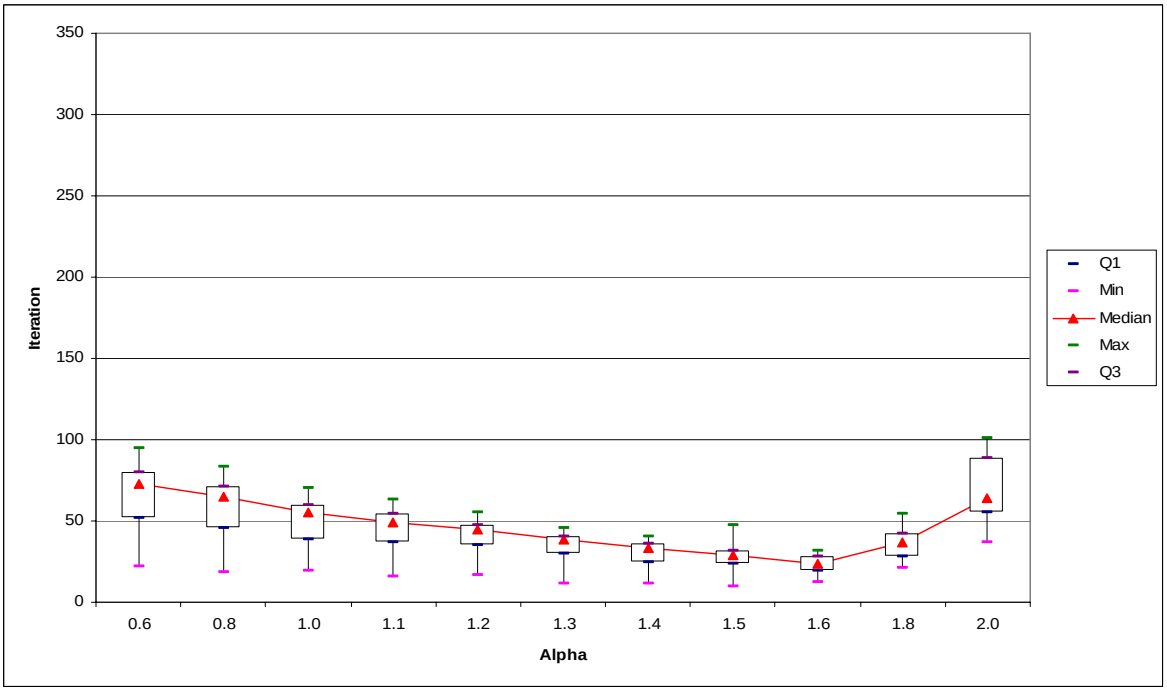

Figure B3. Box plot of iteration against scale alpha of 5Tec.txt (6000 datapoints, dimension=5and K=8) after 40 runs.

Figure B4

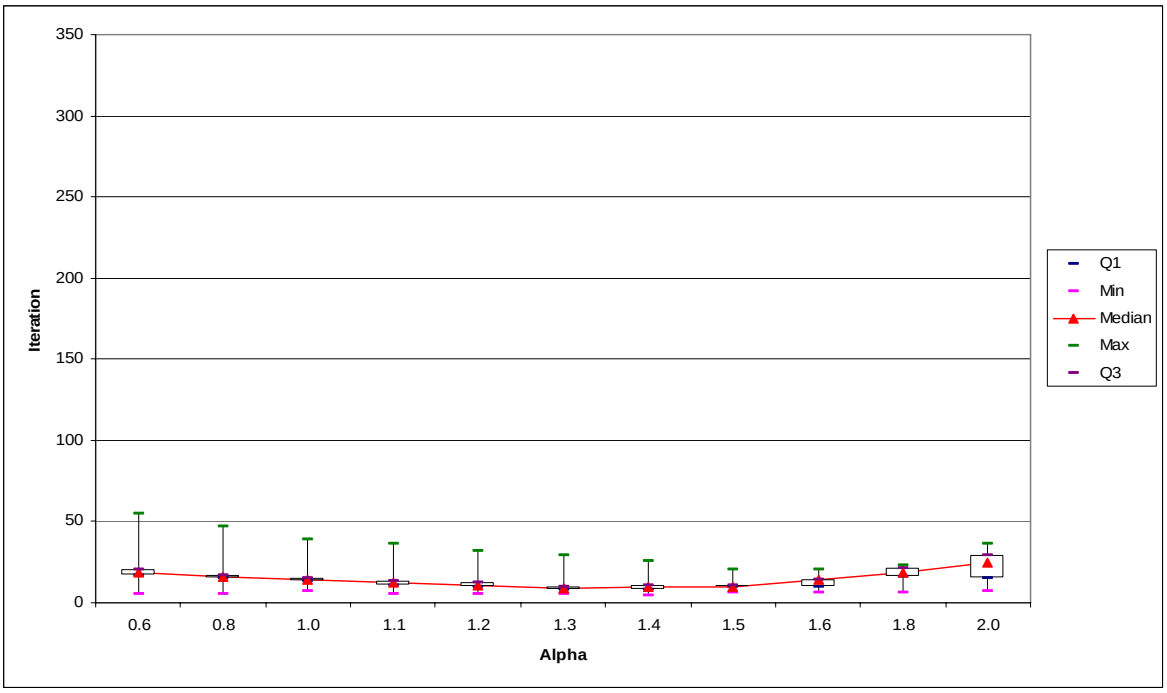

Figure B4. Box plot of iteration against scale alpha of 5Tec.txt (6000 datapoints, dimension=5and K=4) after 40 runs.

Figure B5

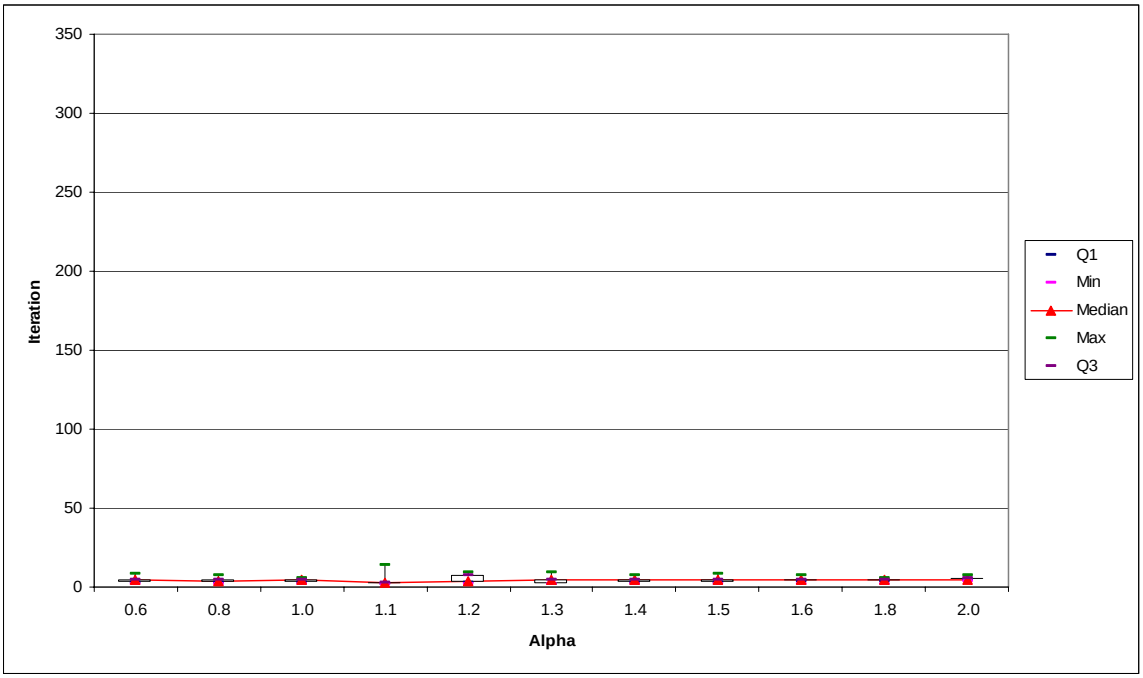

Figure B5. Box plot of iteration against scale alpha of 5Tec.txt (6000 datapoints, dimension=5and K= 2) after 40 runs.

Figure C1

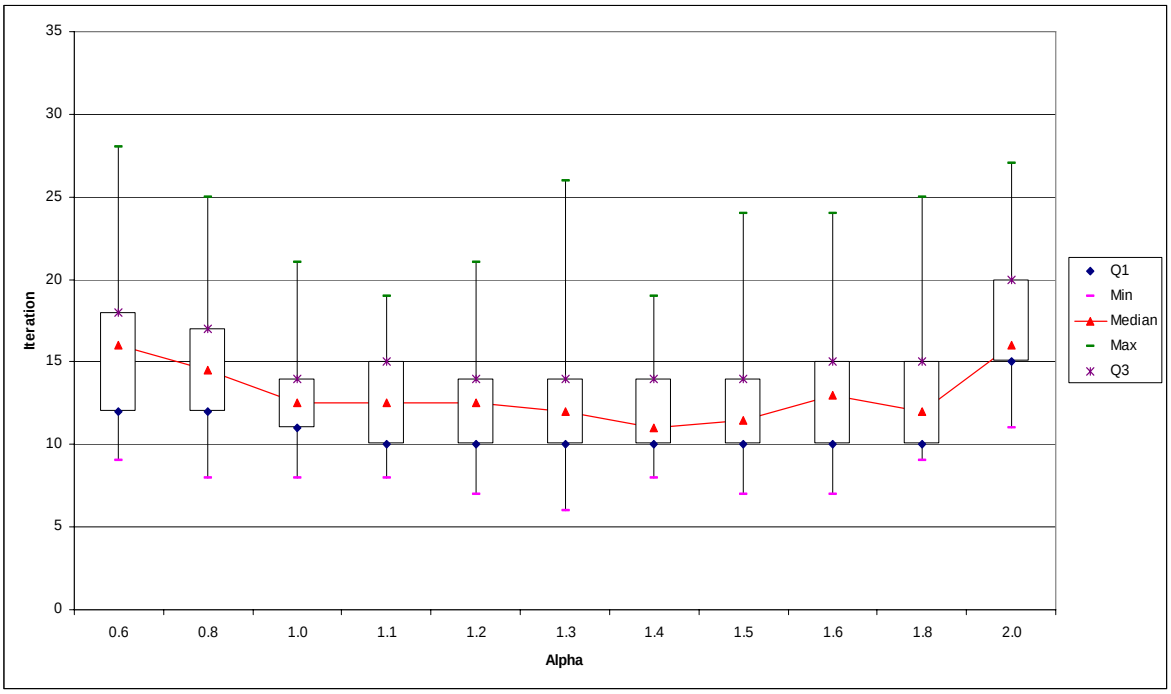

Figure C1. Box plot of iteration against scale alpha of 14Housing.txt (506 datapoints, dimension=14 and K=32) after 40 runs.

Figure C2

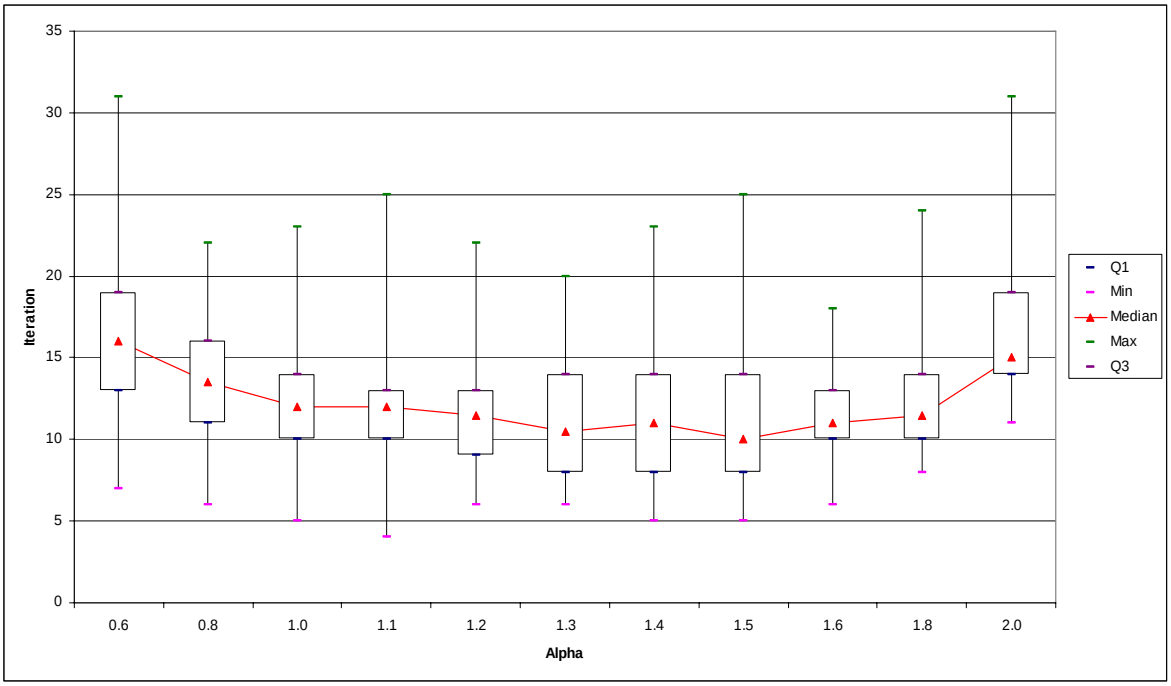

Figure C2. Box plot of iteration against scale alpha of 14Housing.txt (506 datapoints, dimension=14 and K=16) after 40 runs.

Figure C3

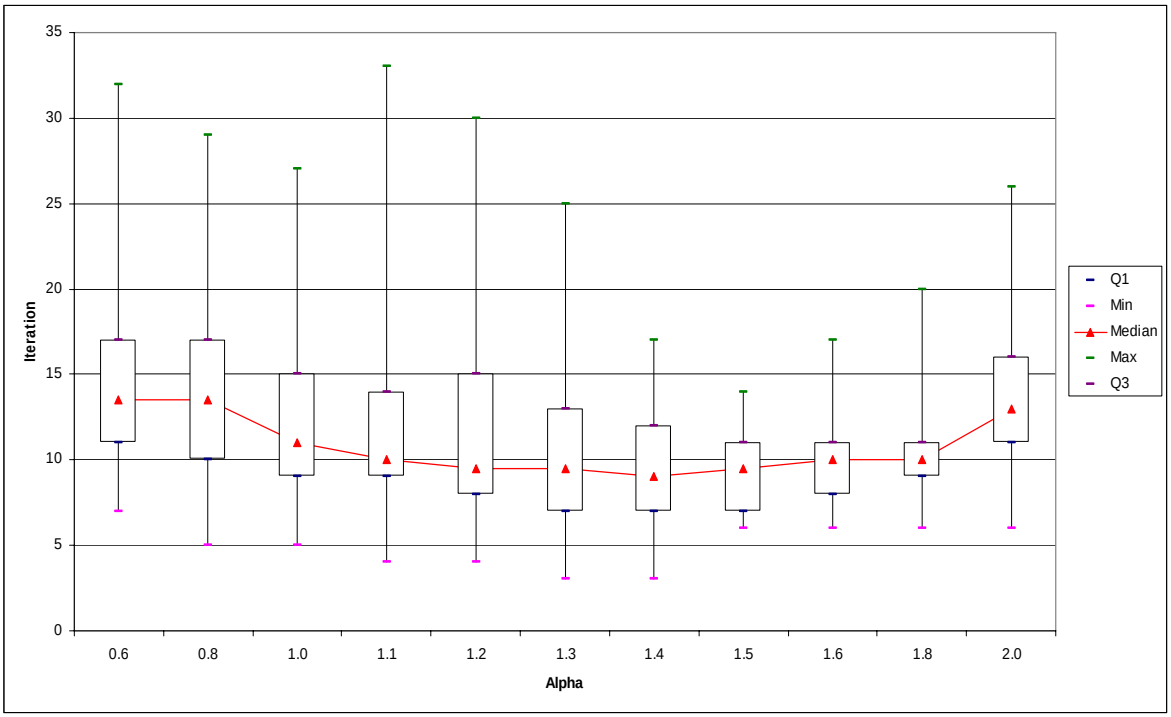

Figure C3. Box plot of iteration against scale alpha of 14Housing.txt (506 datapoints, dimension=14 and K=8) after 40 runs.

Figure C4

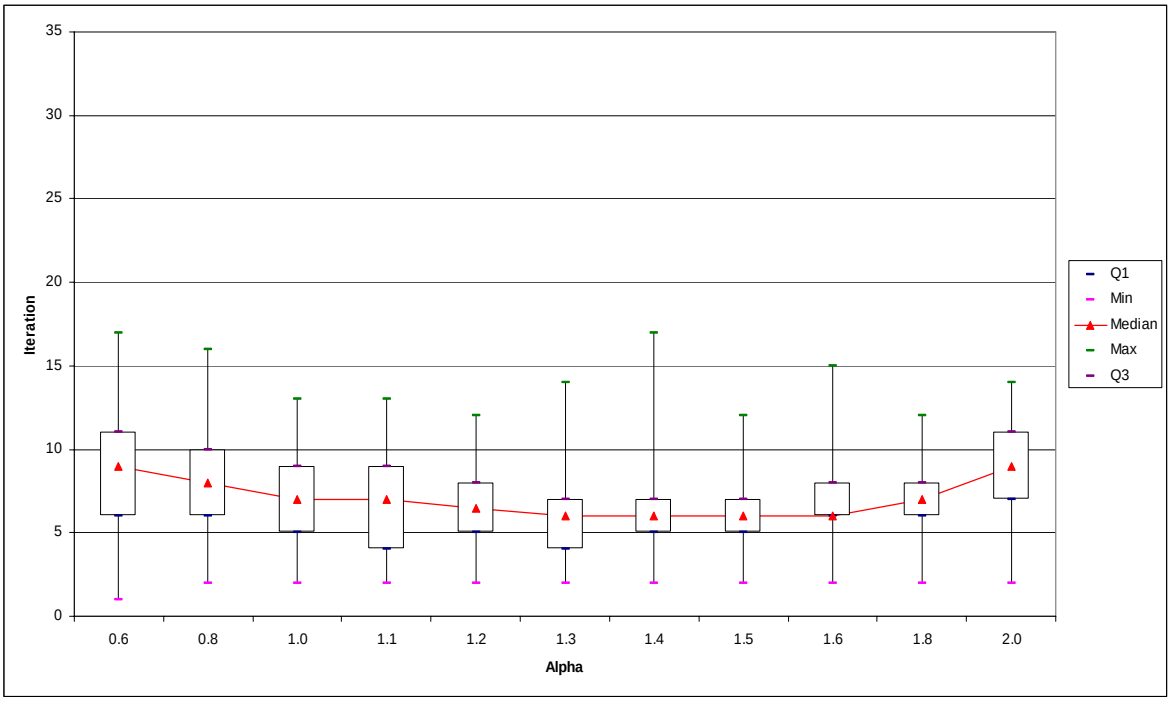

Figure C4. Box plot of iteration against scale alpha of 14Housing.txt (506 datapoints, dimension=14 and K=4) after 40 runs.

Figure C5

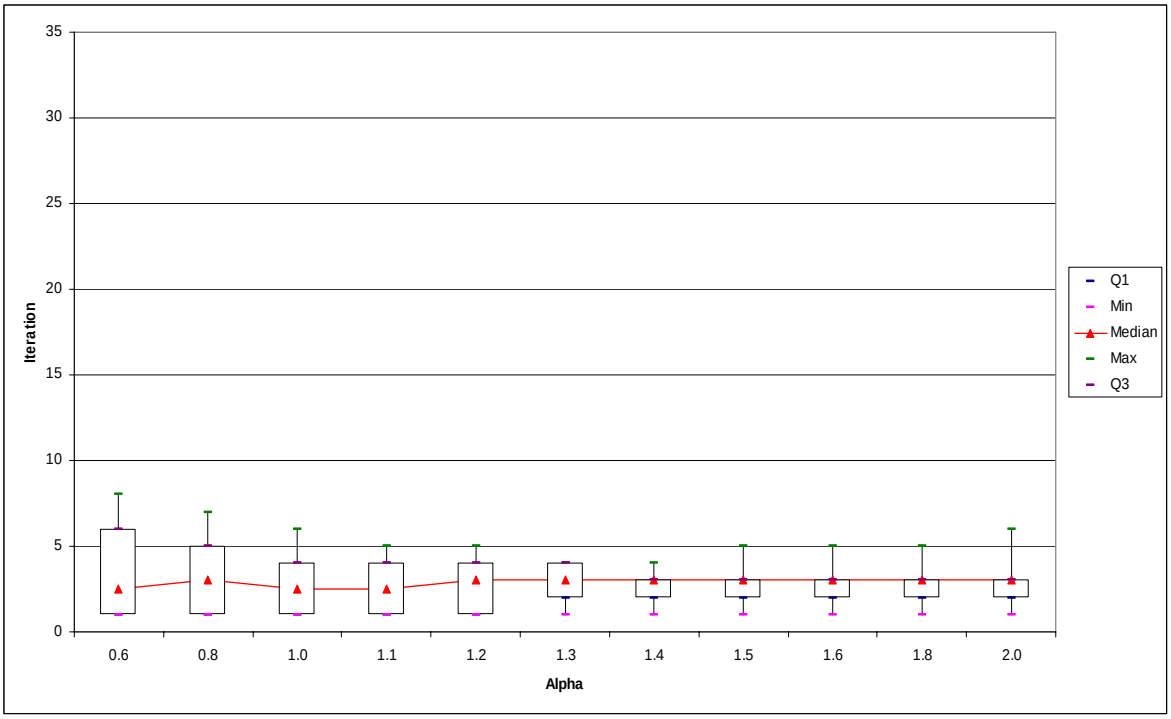

Figure C5. Box plot of iteration against scale alpha of 14Housing.txt (506 datapoints, dimension=14 and K=2) after 40 runs.

Figure D1

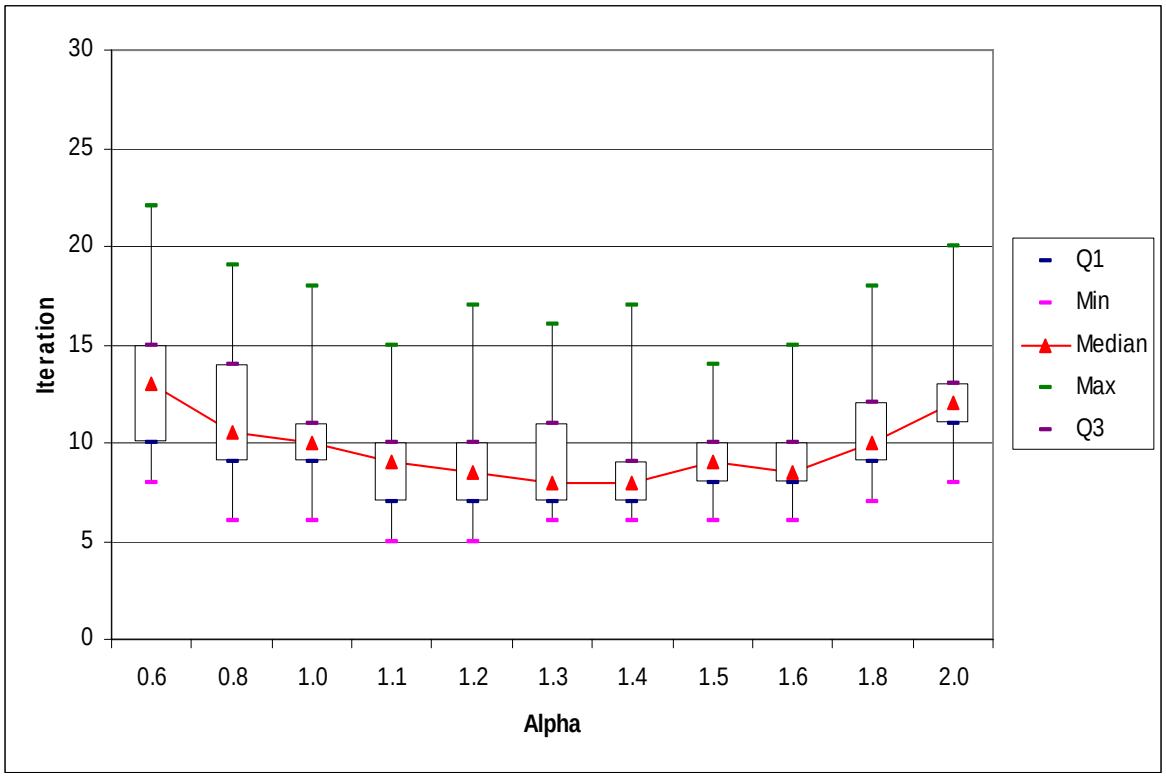

Figure D1. Box plot of iteration against scale alpha of SyntheticControl.txt (600 datapoints, dimension=60 and K=32) after 40 runs.

Figure D2

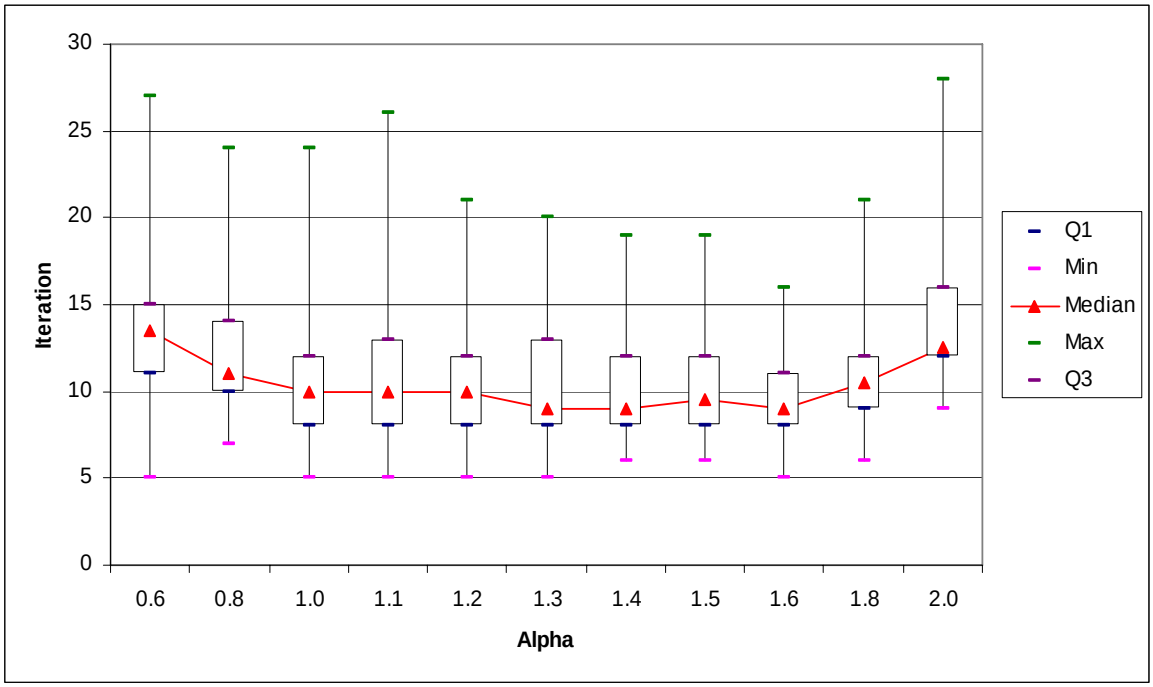

Figure D2. Box plot of iteration against scale alpha of SyntheticControl.txt (600 datapoints, dimension=60 and K=16) after 40 runs.

Figure D3

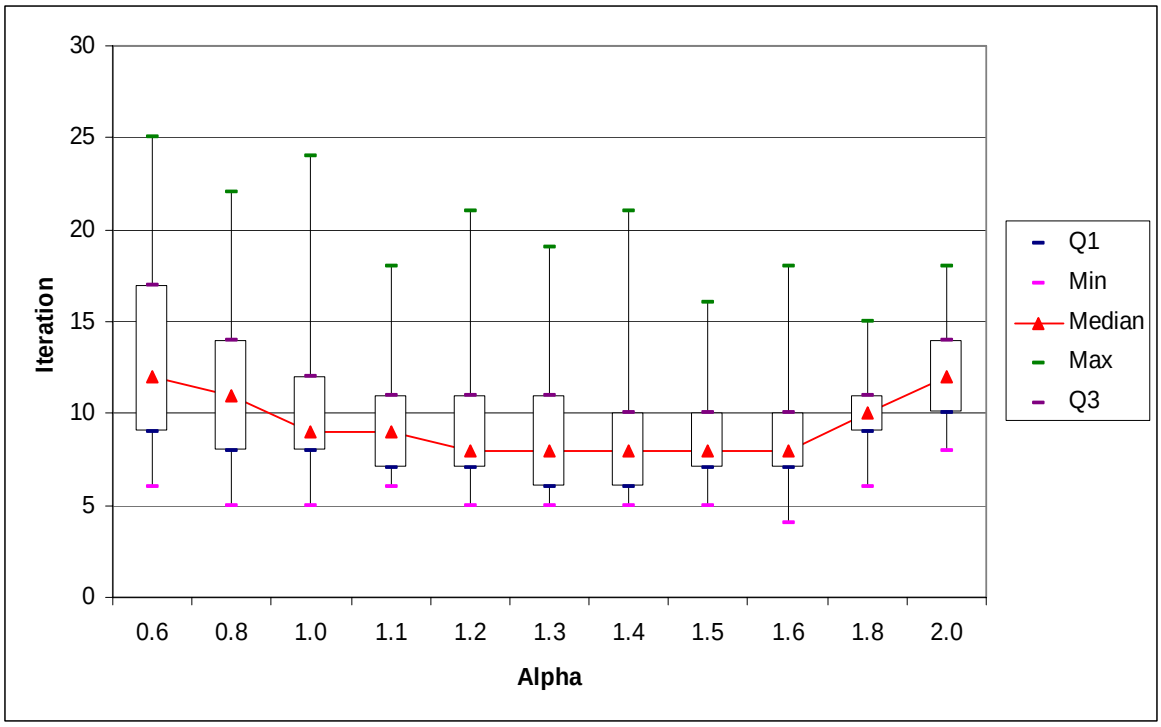

Figure D3. Box plot of iteration against scale alpha of SyntheticControl.txt (600 datapoints, dimension=60 and K=8) after 40 runs.

Figure D4

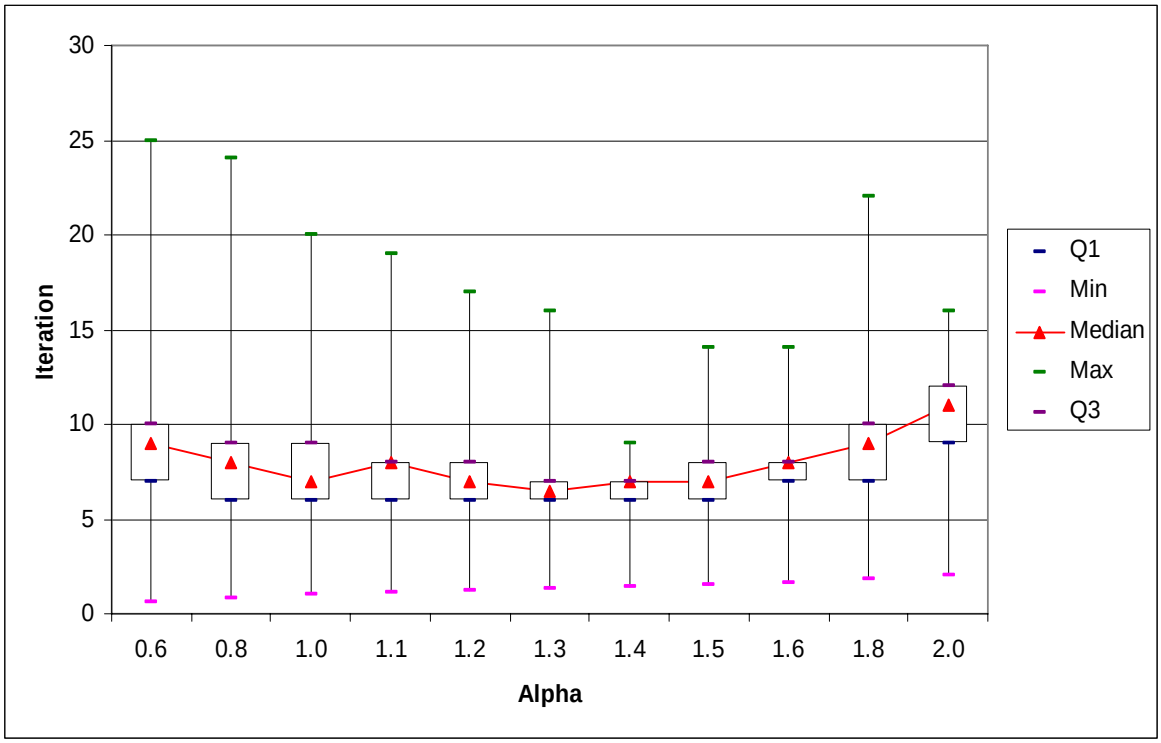

Figure D4. Box plot of iteration against scale alpha of SyntheticControl.txt (600 datapoints, dimension=60 and K=4) after 40 runs.

Figure D5

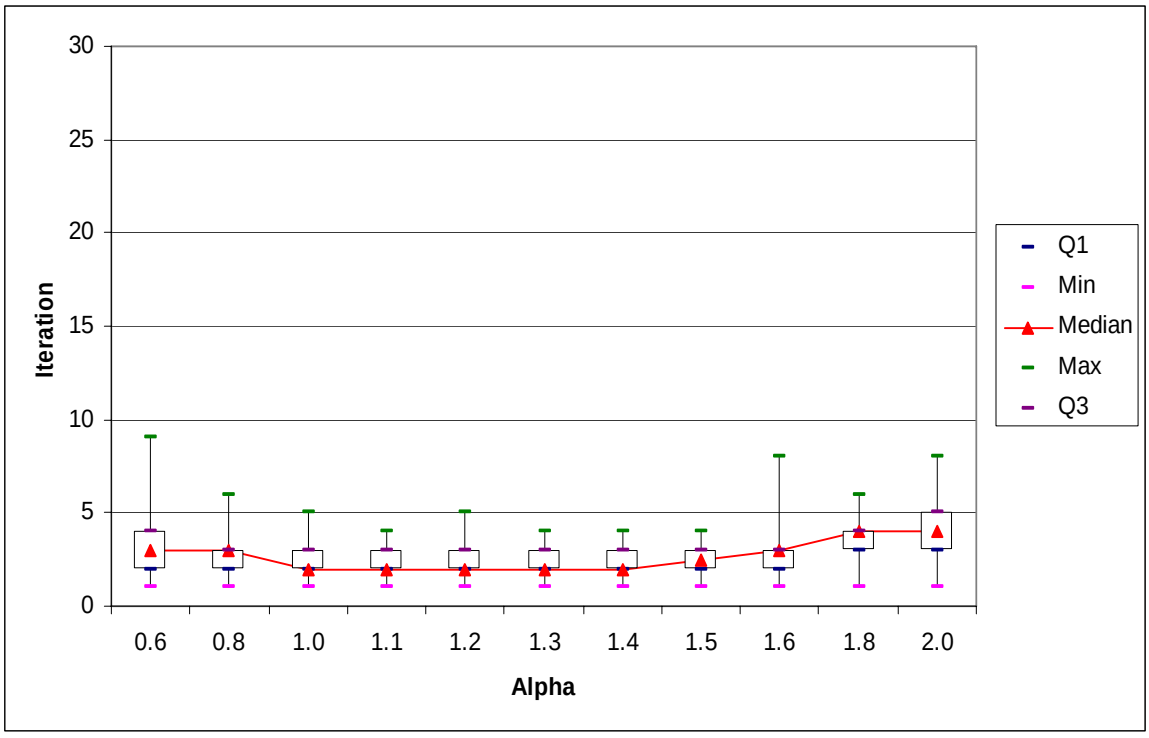

Figure D5. Box plot of iteration against scale alpha of SyntheticControl.txt (600 datapoints, dimension=60 and K=2) after 40 runs.

Figure E1

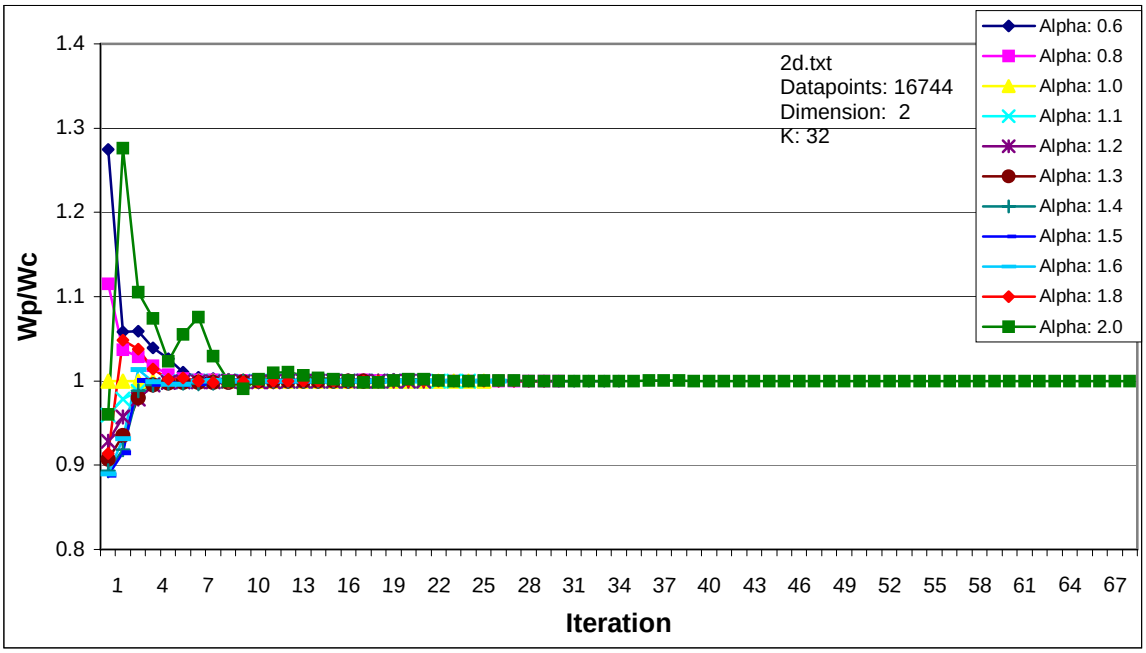

Figure E1. Plot of within class scatter of Perturbed centroids (Wp) / within class scatter of Unperturbed centroids (Wc) against iterations for 2d.txt (16744 datapoints, dimension=2 and K=32).

Figure E2

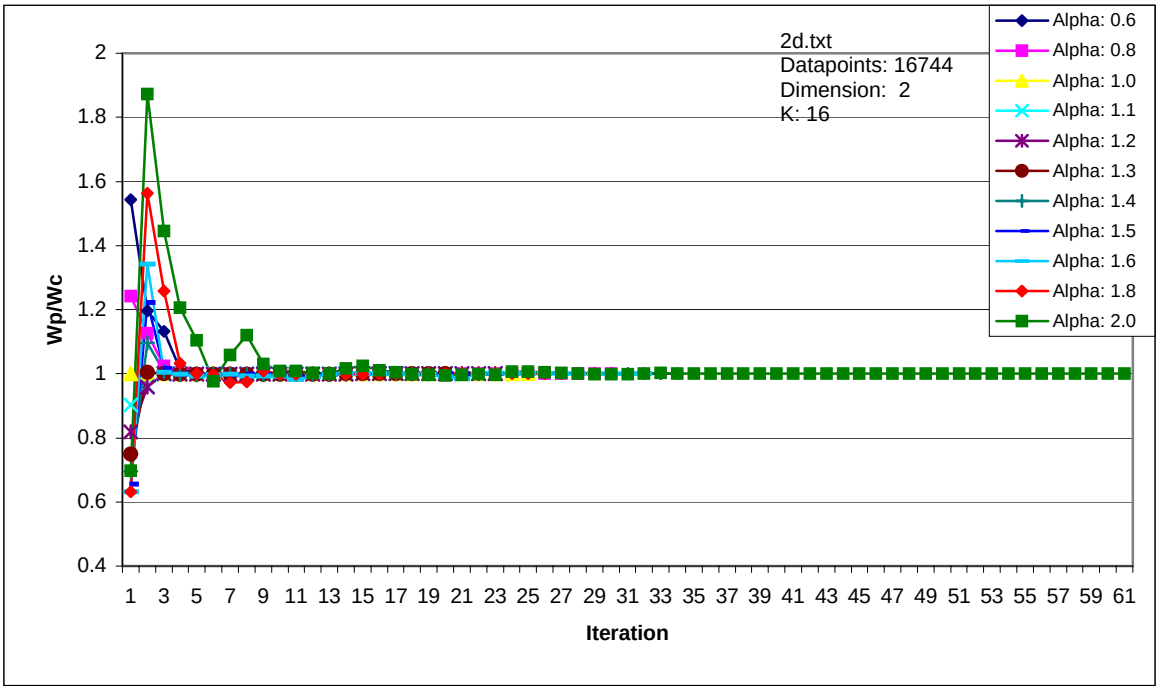

Figure E2. Plot of within class scatter of Perturbed centroids (Wp) / within class scatter of Unperturbed centroids (Wc) against iterations for 2d.txt (16744 datapoints, dimension=2 and K=16).

Figure E3

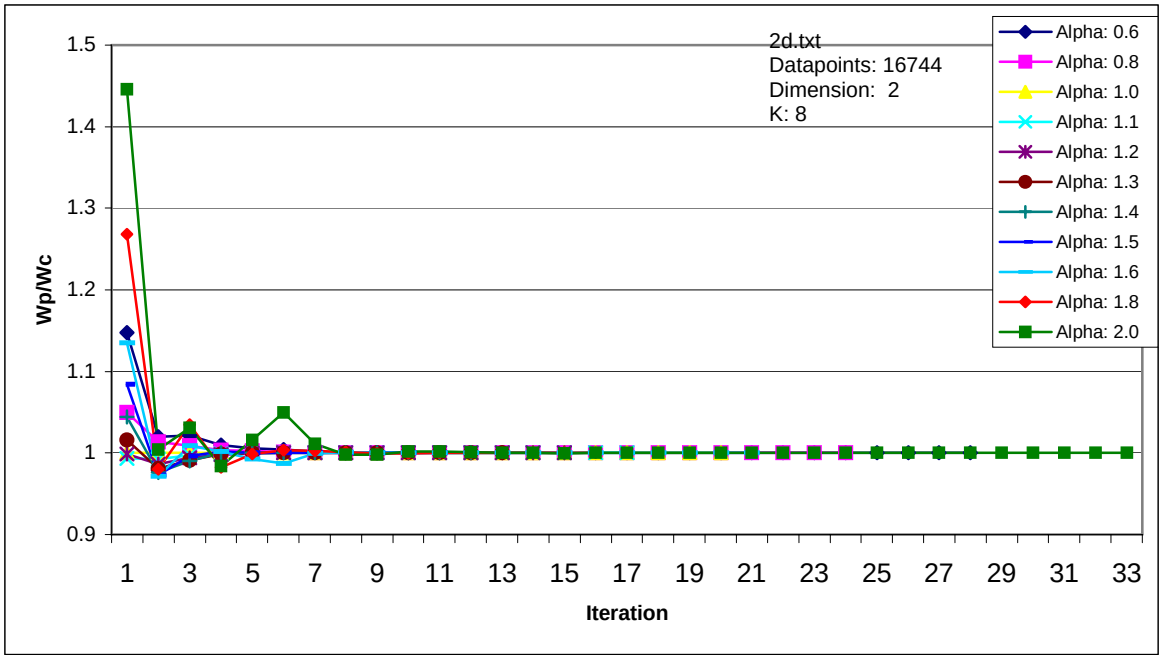

Figure E3. Plot of within class scatter of Perturbed centroids (Wp) / within class scatter of Unperturbed centroids (Wc) against iterations for 2d.txt (16744 datapoints, dimension=2 and K=8).

Figure E4

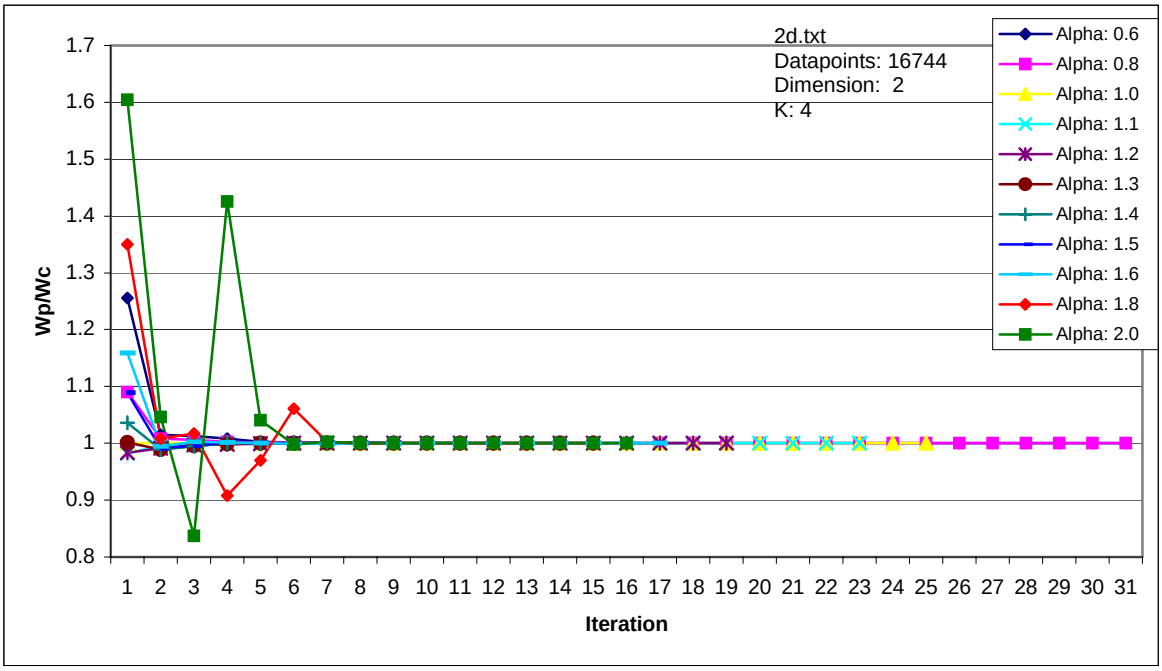

Figure E4. Plot of within class scatter of Perturbed centroids (Wp) / within class scatter of Unperturbed centroids (Wc) against iterations for 2d.txt (16744 datapoints, dimension=2 and K=4).

Figure E5

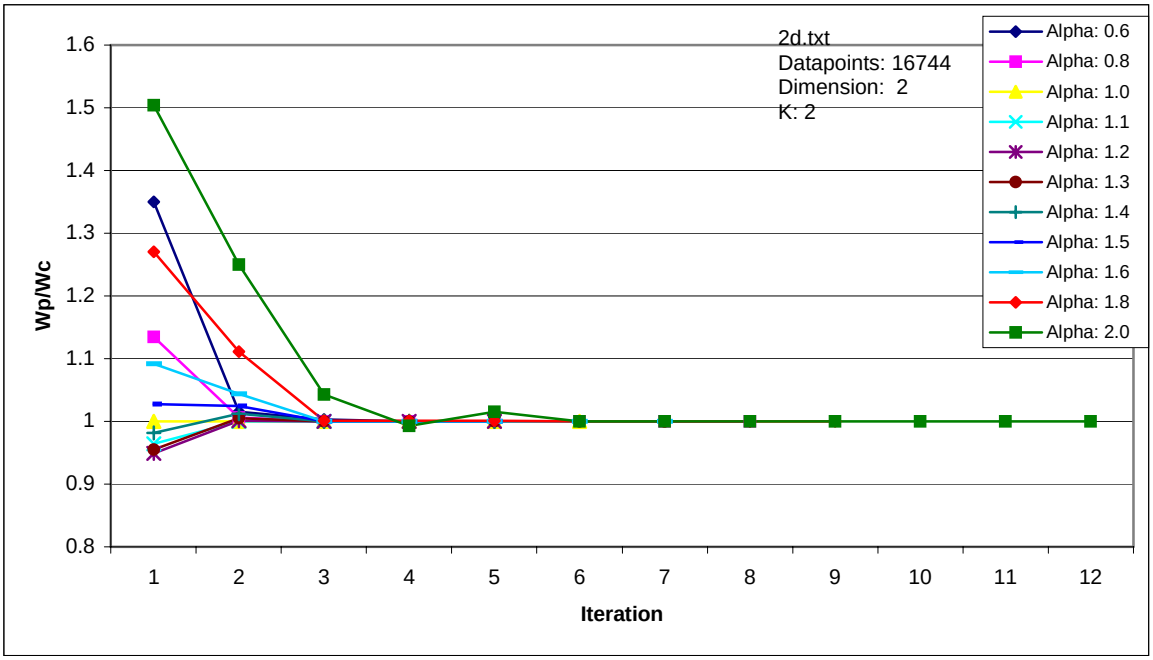

Figure E5. Plot of within class scatter of Perturbed centroids (Wp) / within class scatter of Unperturbed centroids (Wc) against iterations for 2d.txt (16744 datapoints, dimension=2 and K=2).

Figure F1

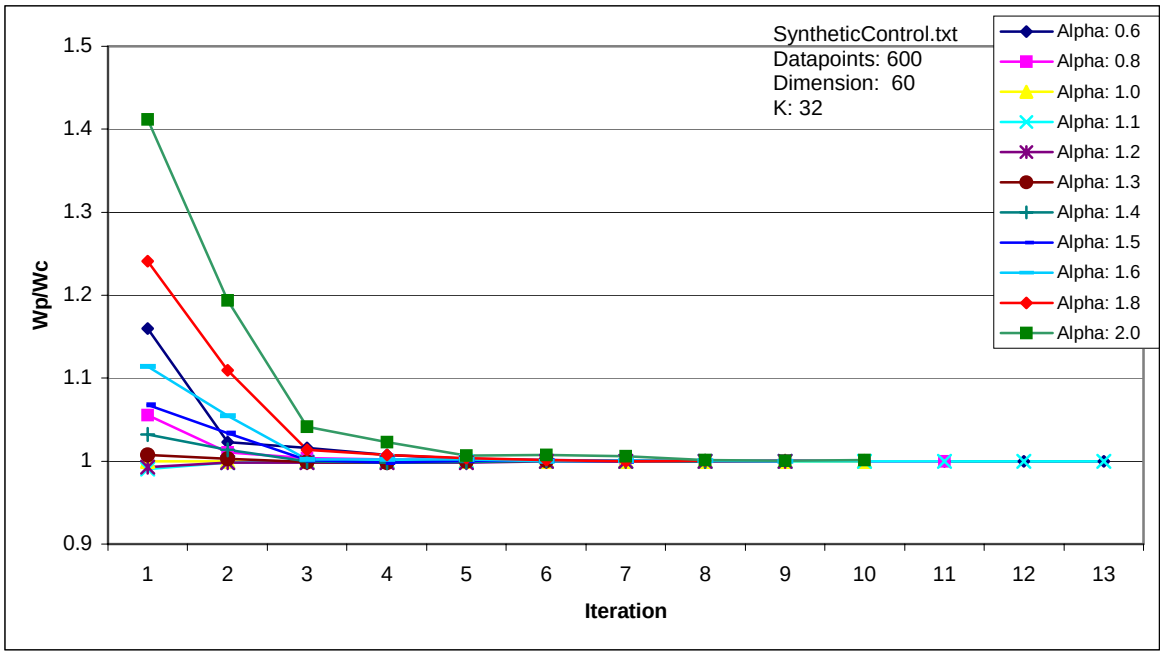

Figure F1. Plot of within class scatter of Perturbed centroid (Wp) / within class scatter of Unperturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=32).

Figure F2

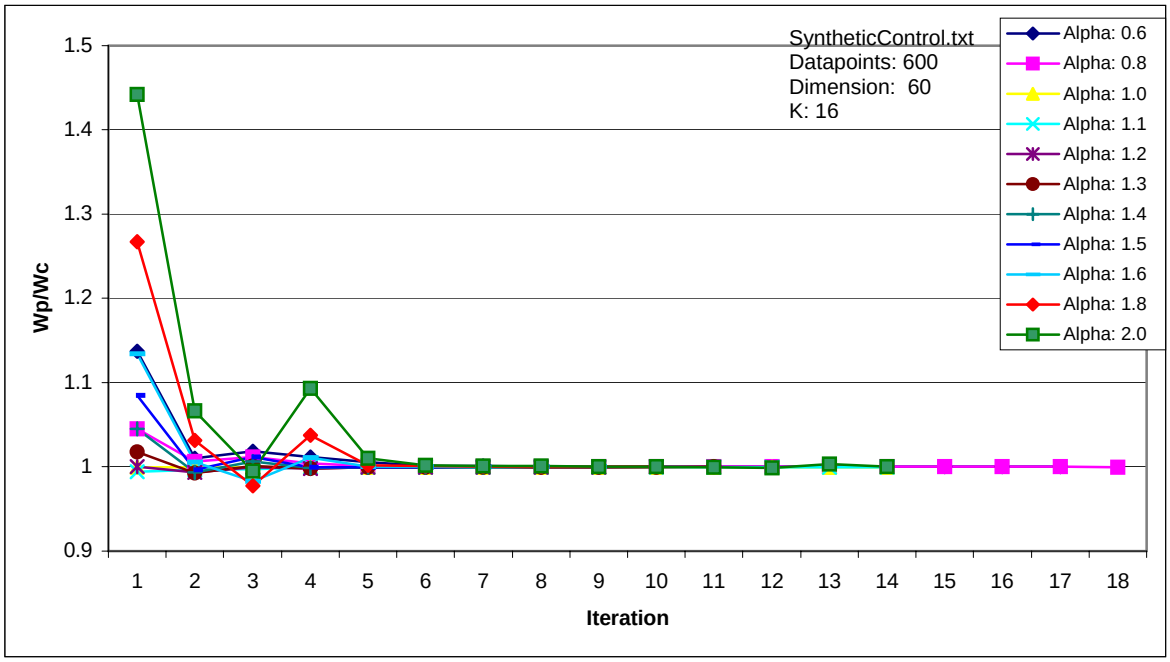

Figure F2. Plot of within class scatter of Perturbed centroid (Wp) / within class scatter of Unperturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=16).

Figure F3

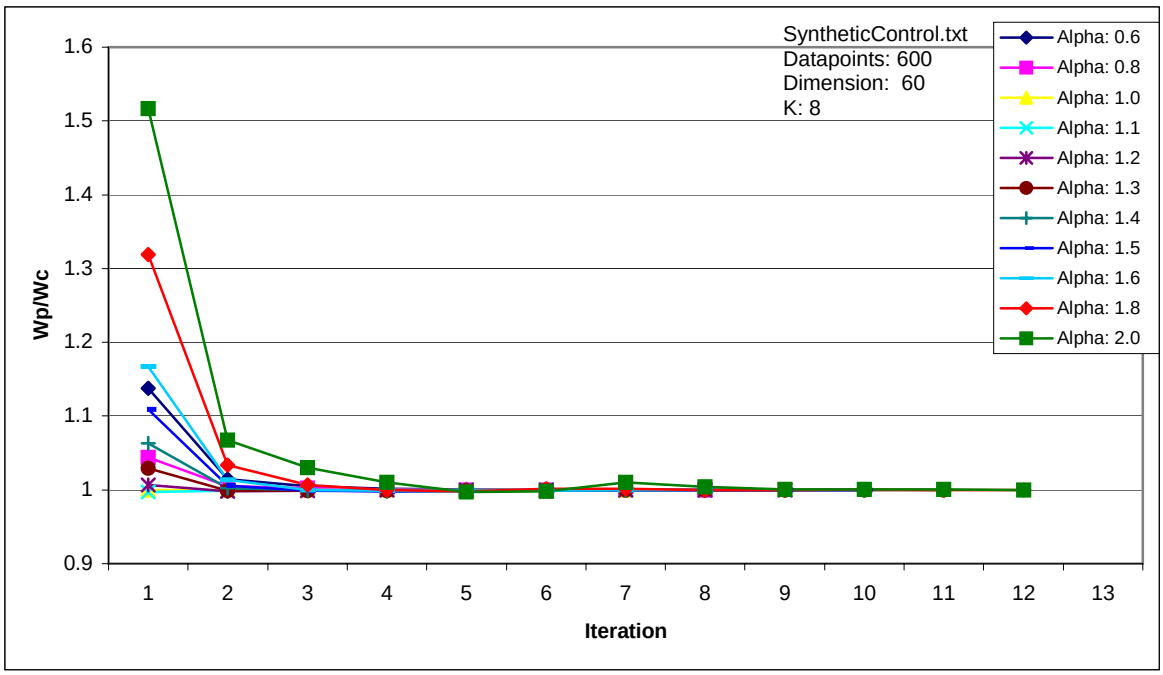

Figure F3. Plot of within class scatter of Perturbed centroid (Wp) / within class scatter of Unperturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=8).

Figure F4

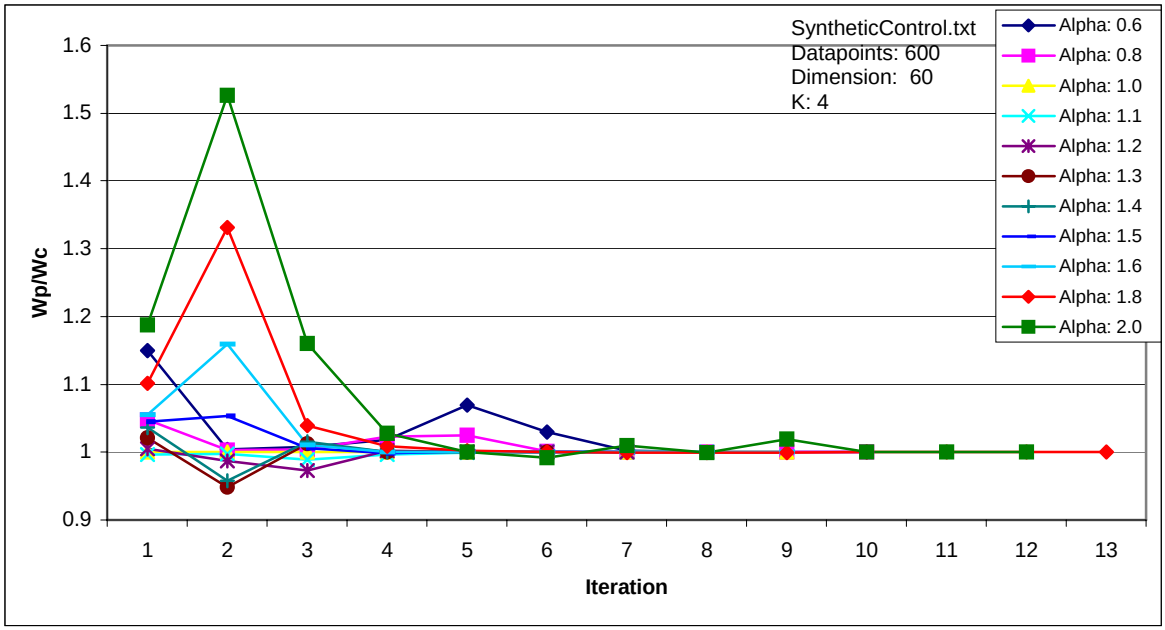

Figure F4. Plot of within class scatter of Perturbed centroid (Wp) / within class scatter of Unperturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=4).

Figure F5

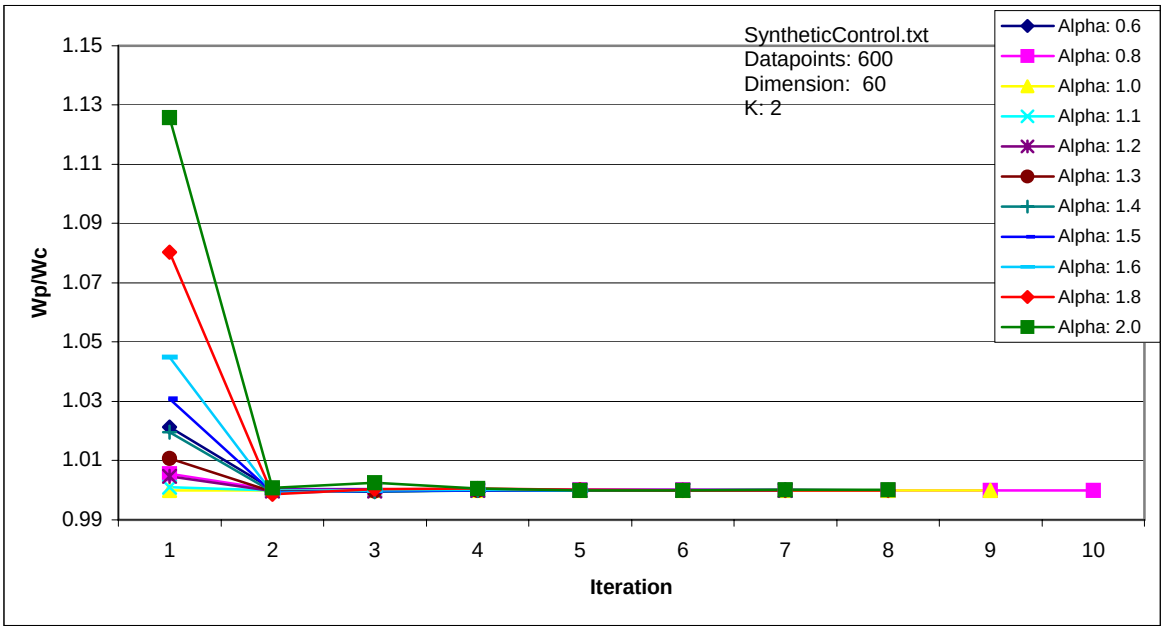

Figure F5- Plot of within class scatter of Perturbed centroid (Wp) / within class scatter of Unperturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=2).

Figure G1

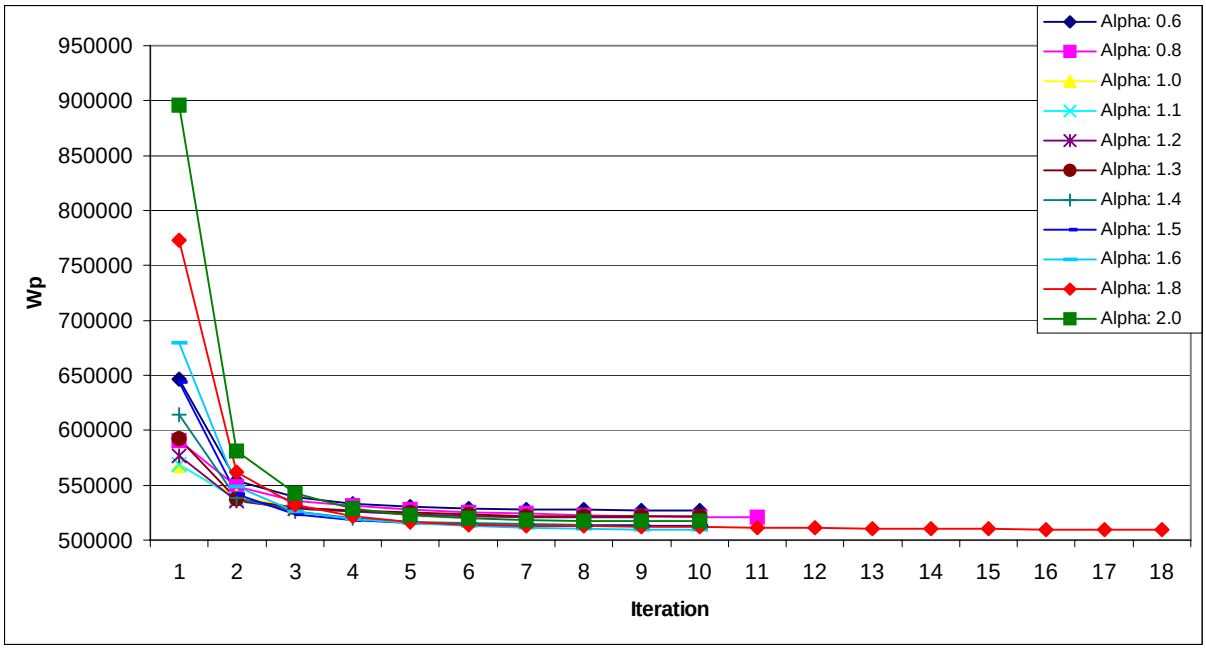

Figure G1. Plot of within class scatter of Perturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=32).

Figure G2

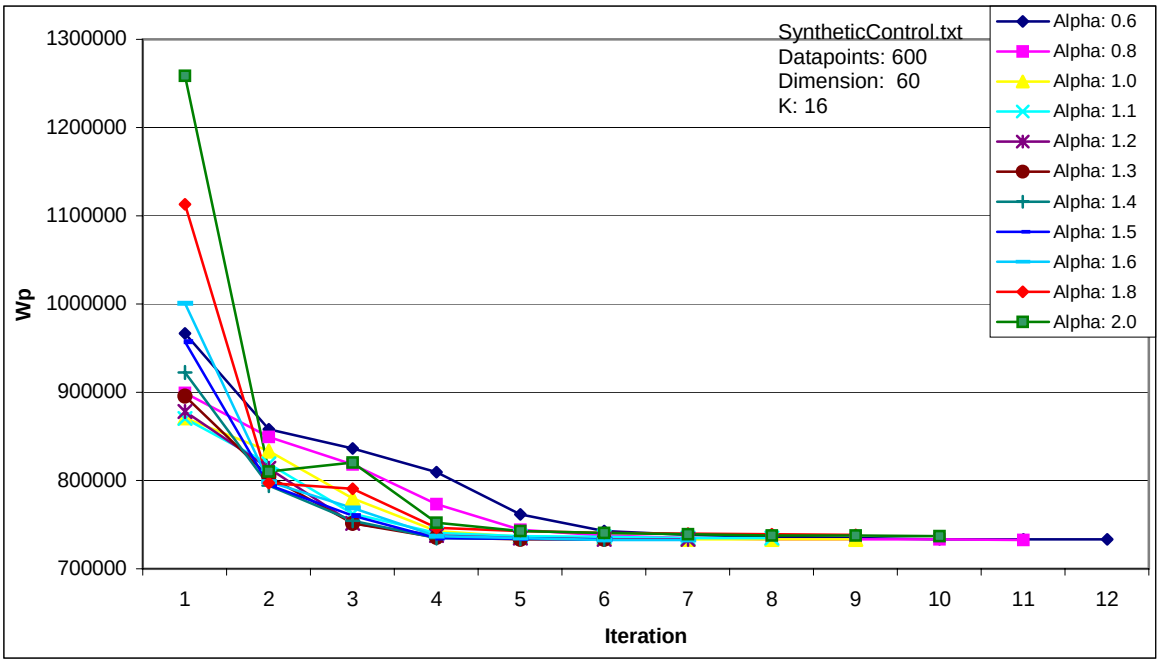

Figure G2. Plot of within class scatter of Perturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=16).

Figure G3

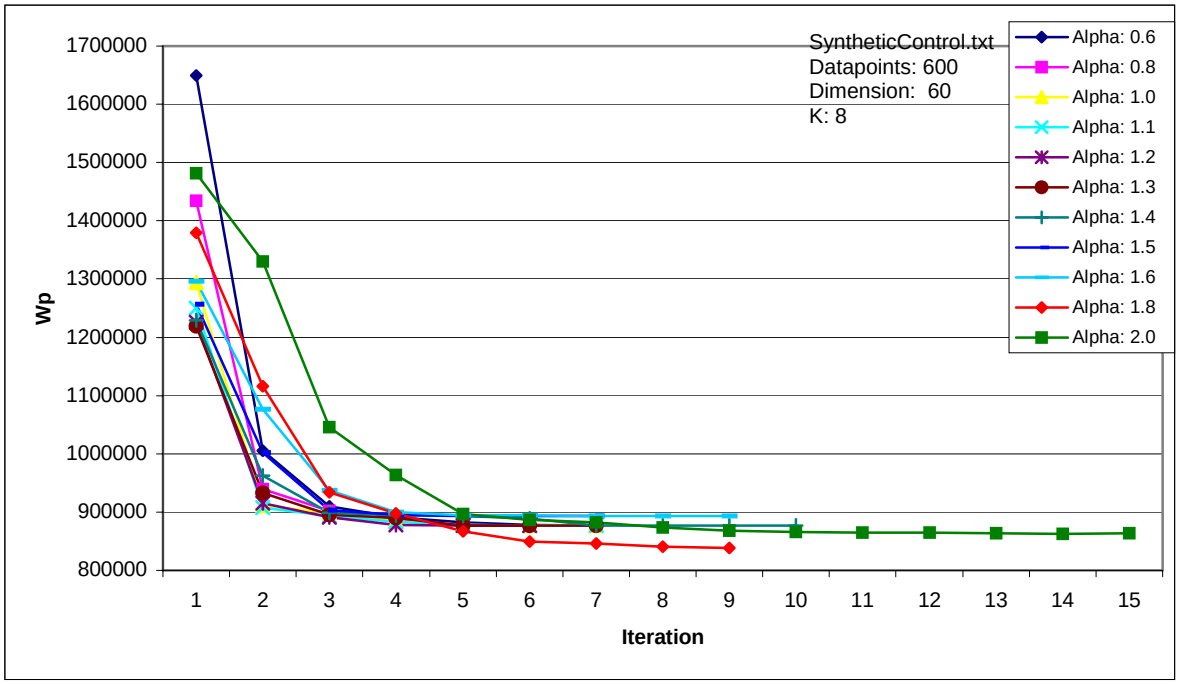

Figure G3. Plot of within class scatter of Perturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=8).

Figure G4

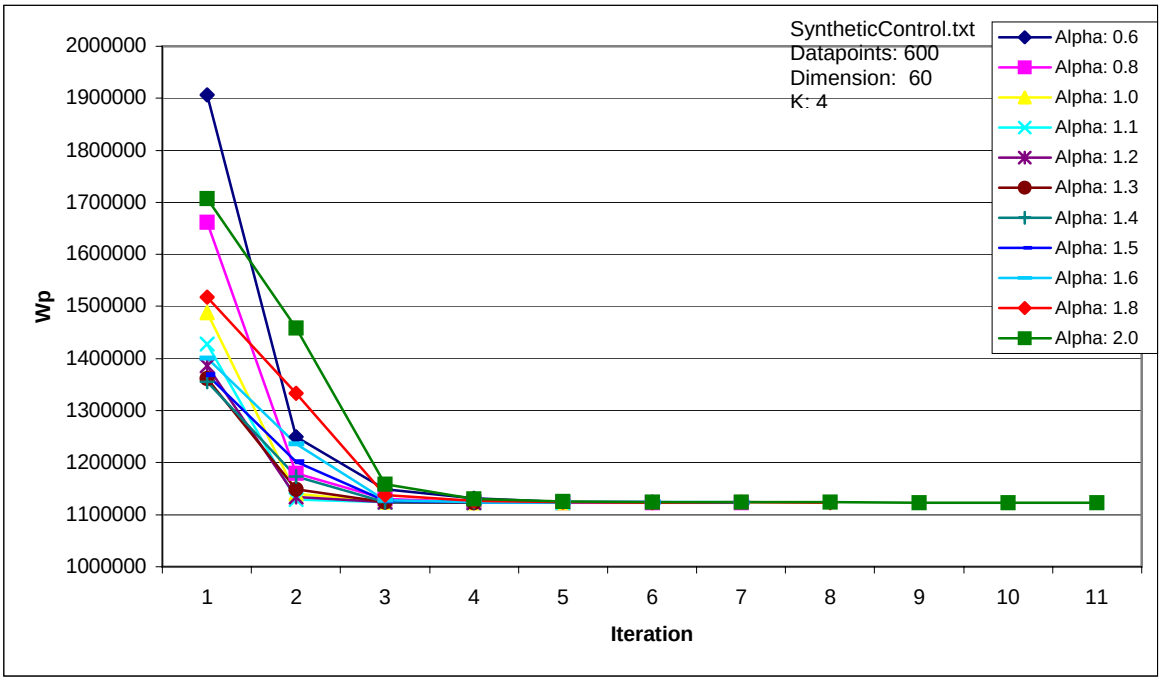

Figure G4. Plot of within class scatter of Perturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=4).

Figure G5

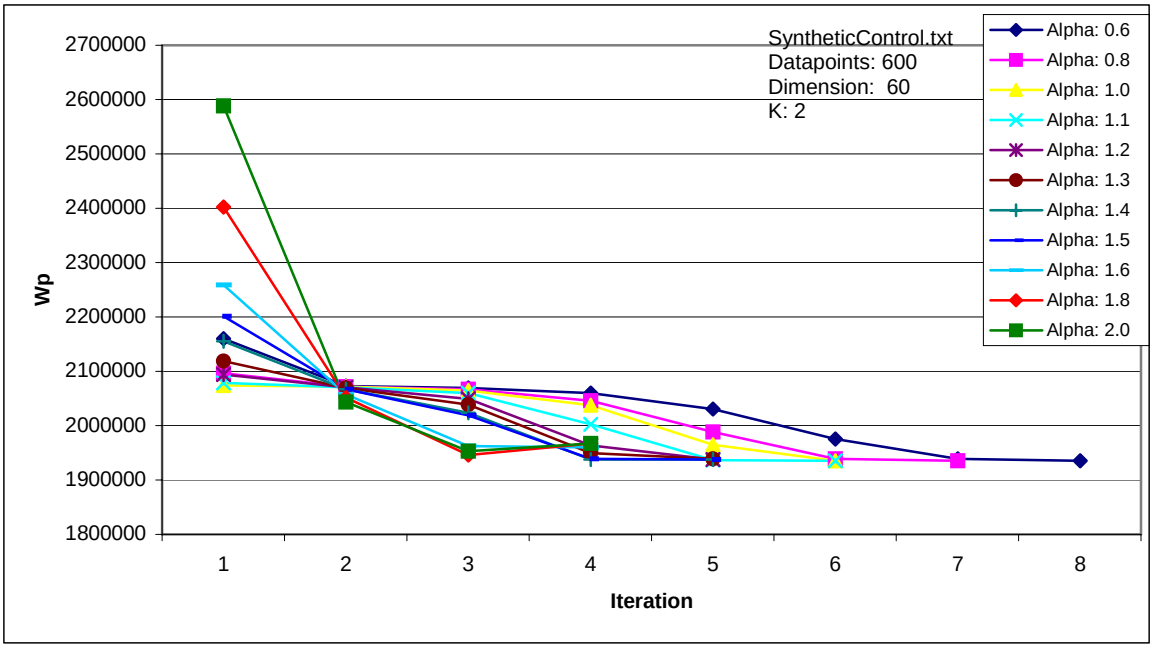

Figure G5. Plot of within class scatter of Perturbed centroid against iteration for SyntheticControl.txt (600 datapoints, dimension=60 and K=2).

Figure H1

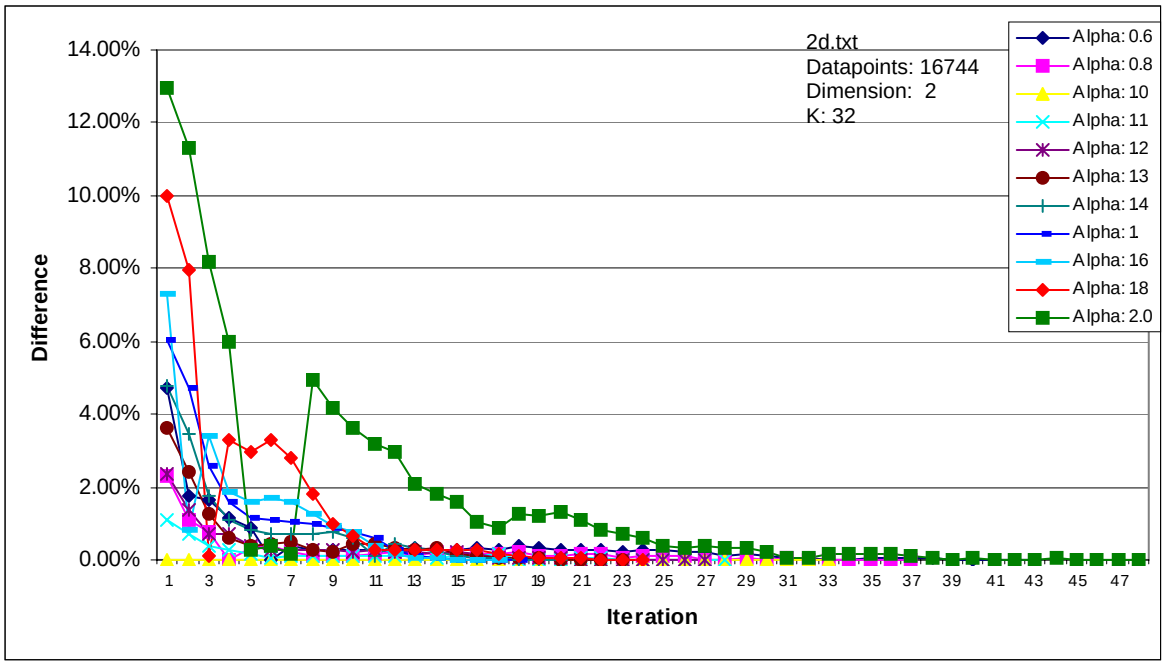

Figure H1. Plot of percentage difference between Perturbed centroid and Unperturbed centroid against iteration for 2d.txt (dimension=60 and K=32).

Figure H2

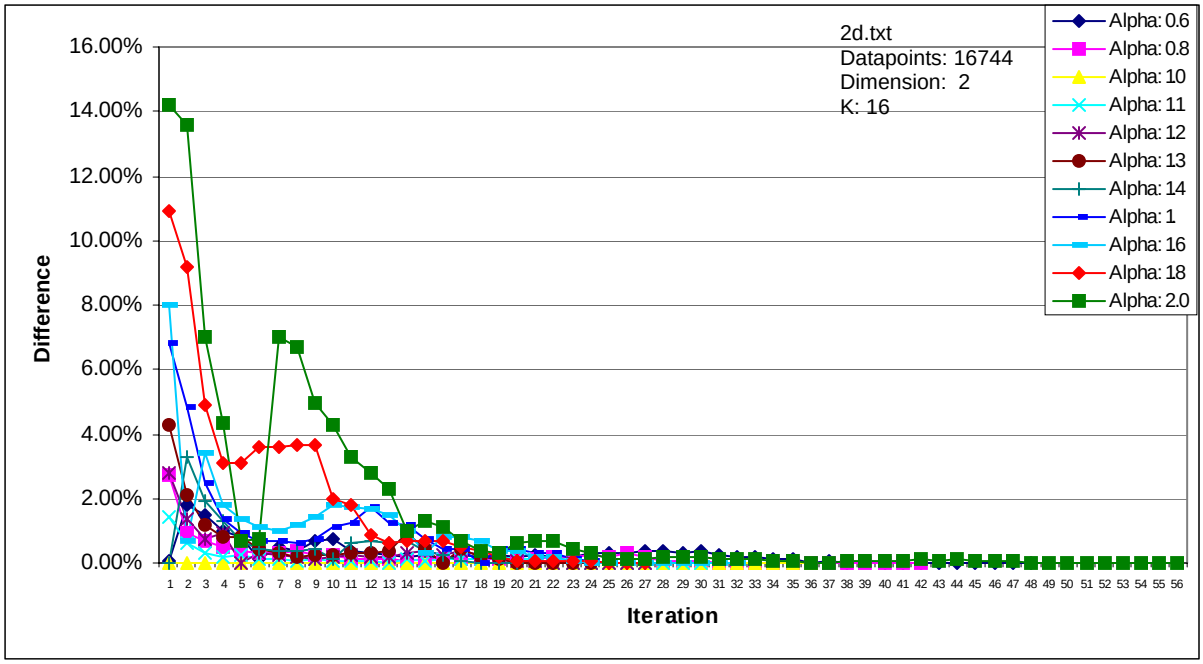

Figure H2. Plot of percentage difference between Perturbed centroid and Unperturbed centroid against iteration for 2d.txt (dimension=60 and K=16).

Figure H3

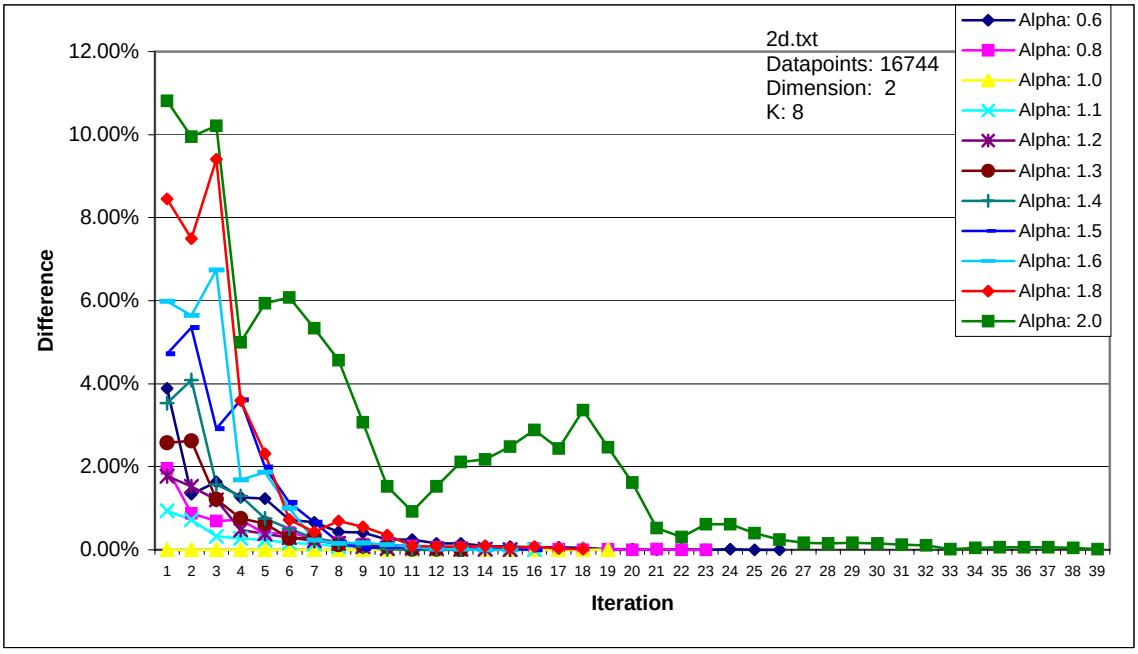

Figure H3. Plot of percentage difference between Perturbed centroid and Unperturbed centroid against iteration for 2d.txt (dimension=60 and K=8).

Figure H4

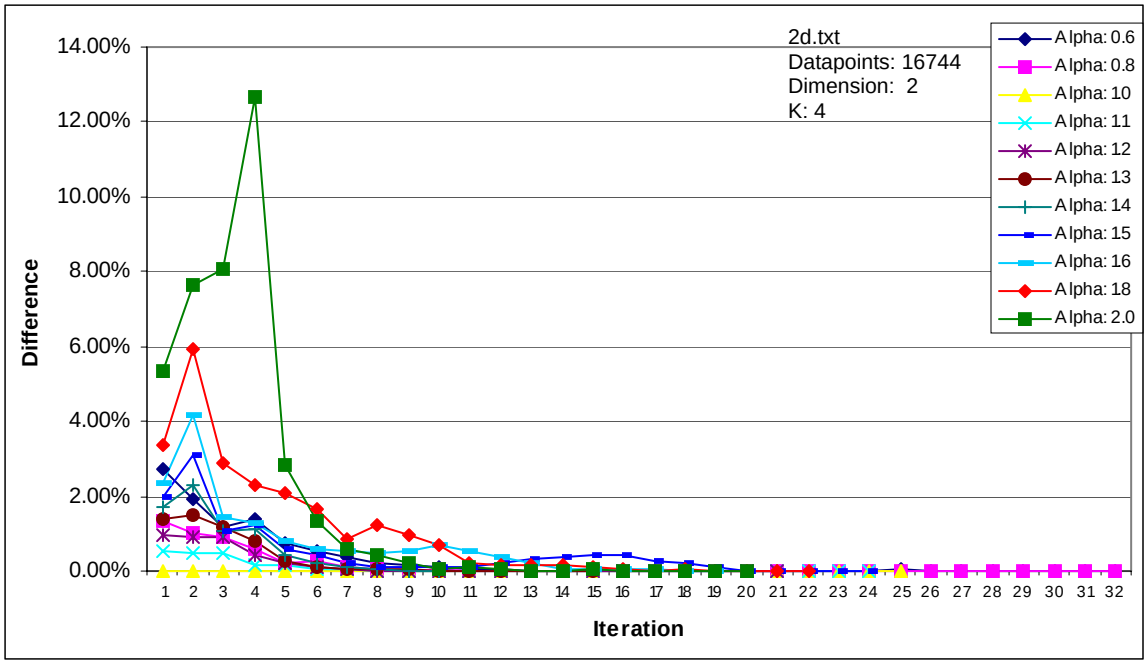

Figure H4. Plot of percentage difference between Perturbed centroid and Unperturbed centroid against iteration for 2d.txt (dimension=60 and K=4).

Figure H5

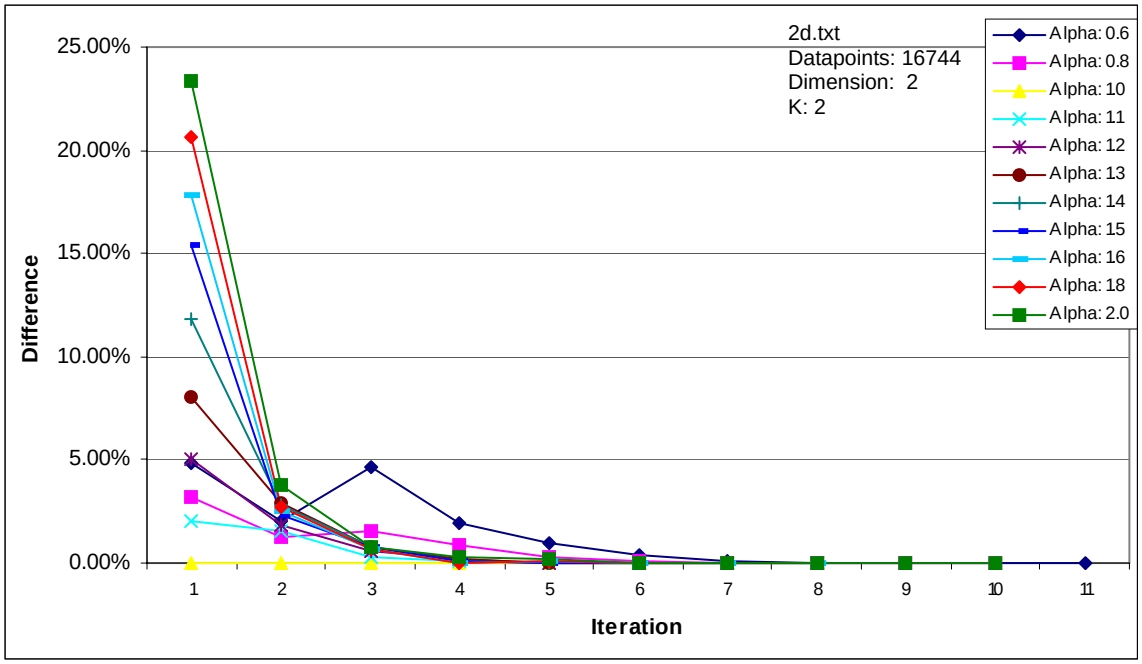

Figure H5. Plot of percentage difference between Perturbed centroid and Unperturbed centroid against iteration for 2d.txt (dimension=60 and K=2).

Figure I1

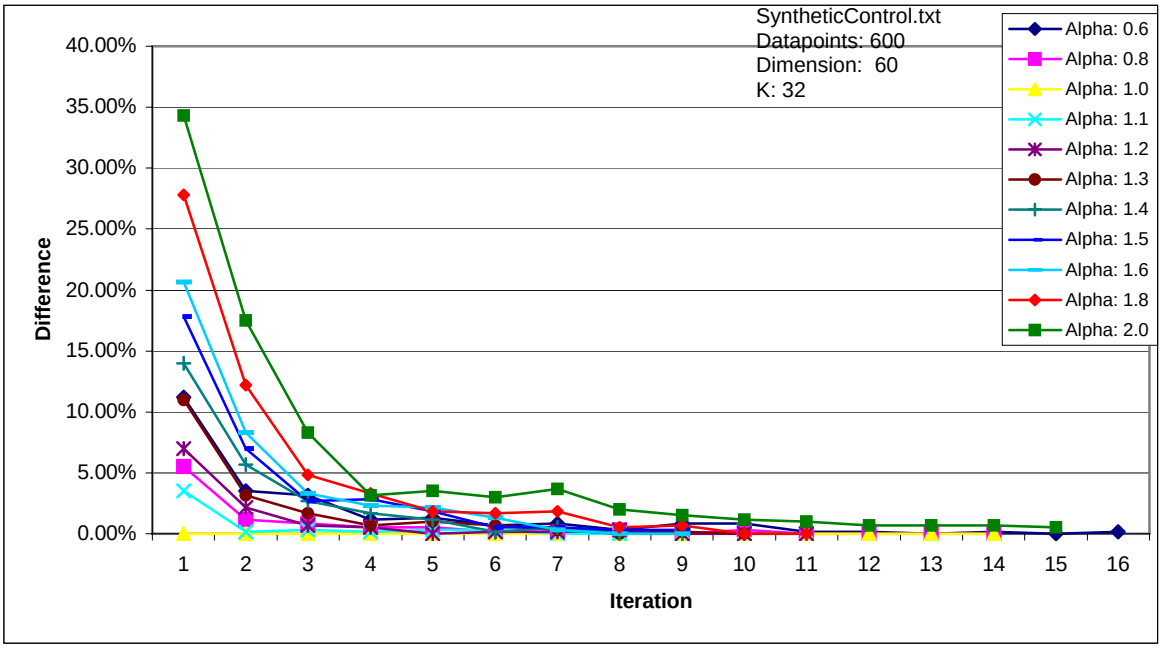

Figure I1. Plot of percentage difference between Perturbed centroid and Unperturbed centroid for SyntheticControl.txt (data size=600, dimension=60 and K=32).

Figure I2

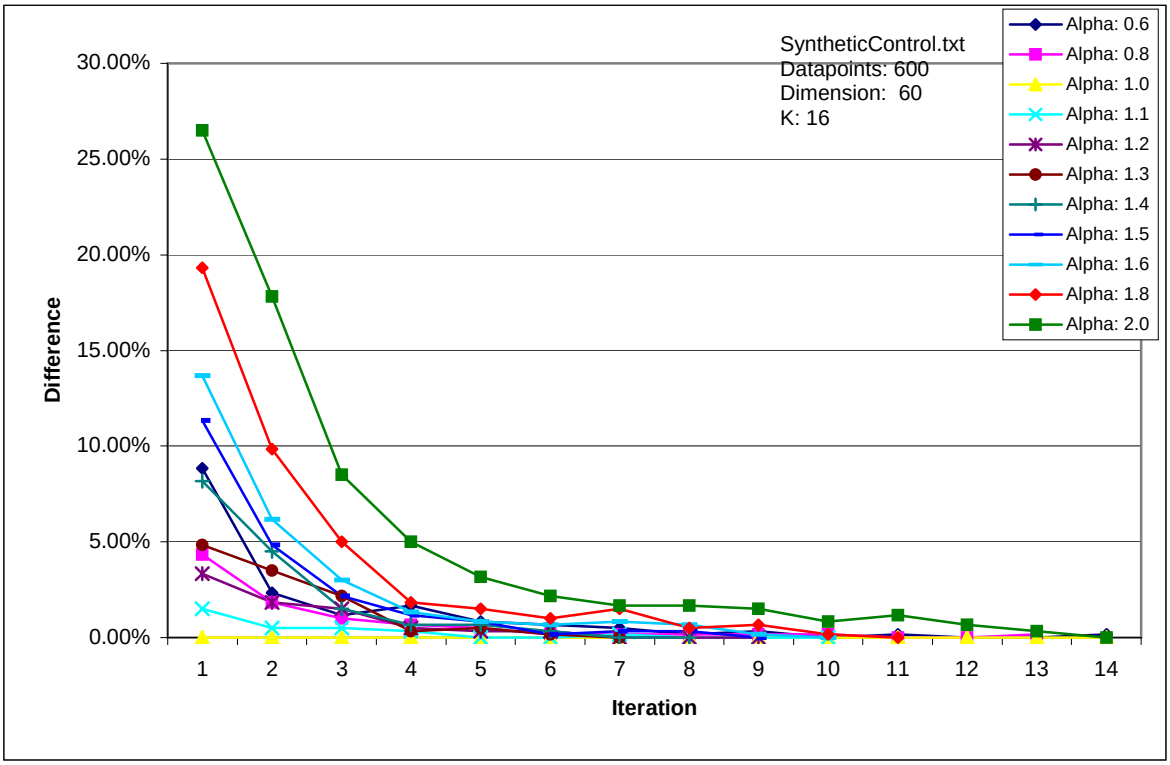

Figure I2. Plot of percentage difference between Perturbed centroid and Unperturbed centroid for SyntheticControl.txt (data size=600, dimension=60 and K=16).

Figure I3

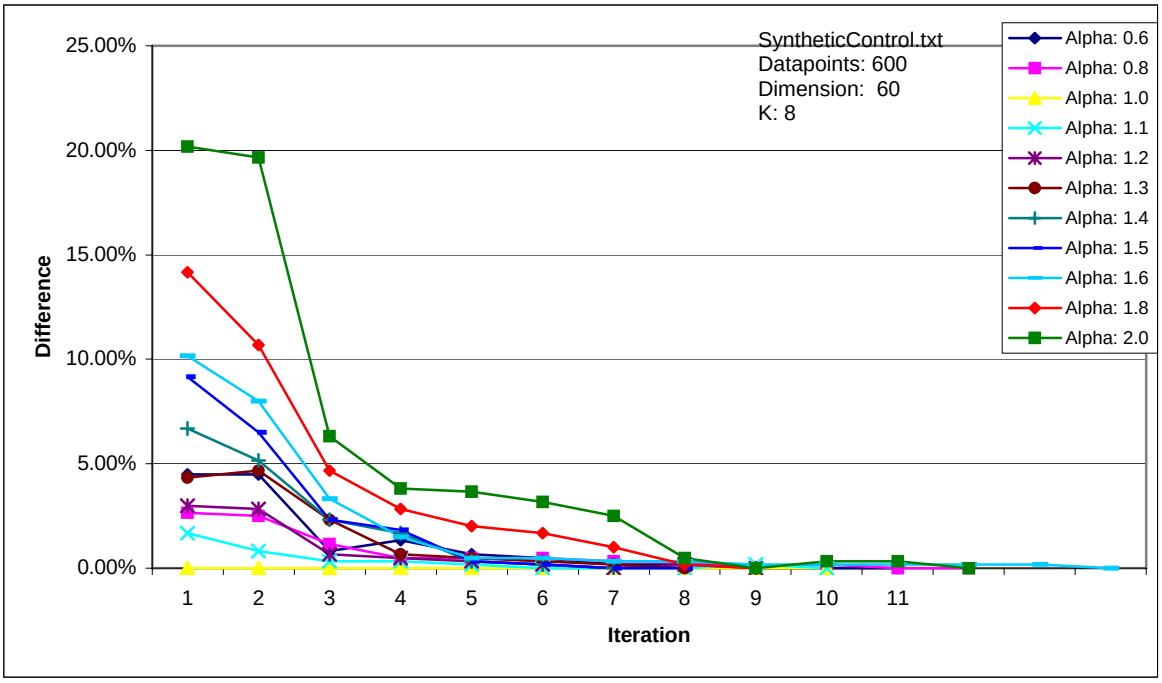

Figure I3. Plot of percentage difference between Perturbed centroid and Unperturbed centroid for SyntheticControl.txt (data size=600, dimension=60 and K=8).

Figure I4

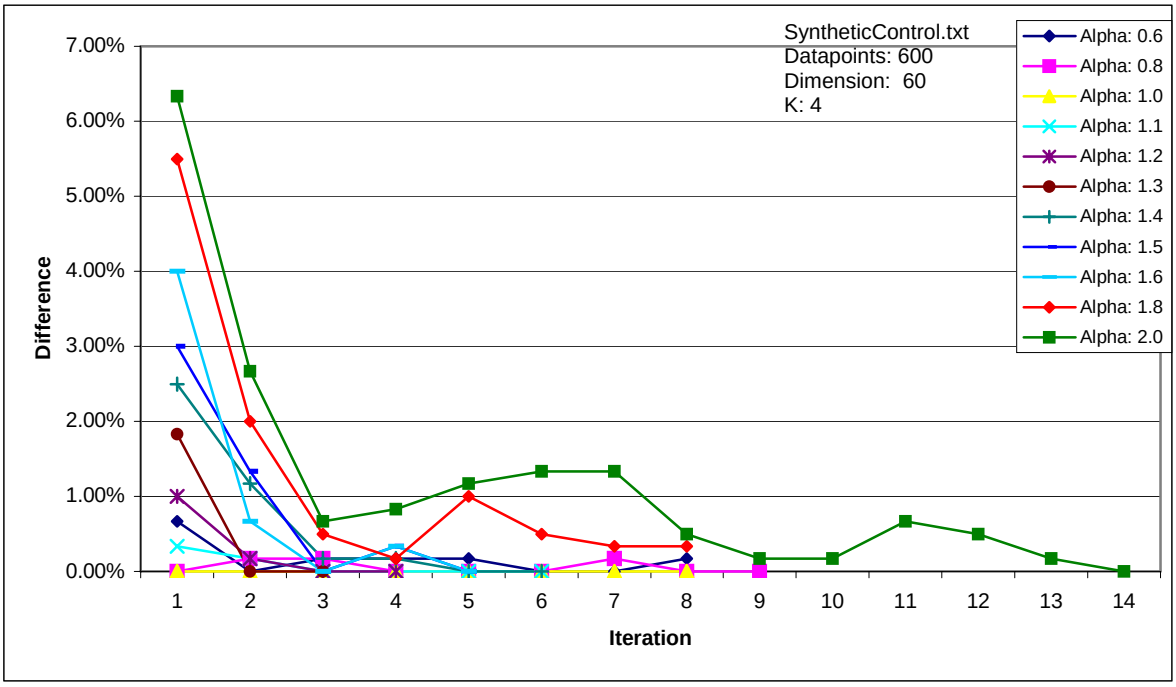

Figure I4. Plot of percentage difference between Perturbed centroid and Unperturbed centroid for SyntheticControl.txt (data size=600, dimension=60 and K=4).

Figure I5

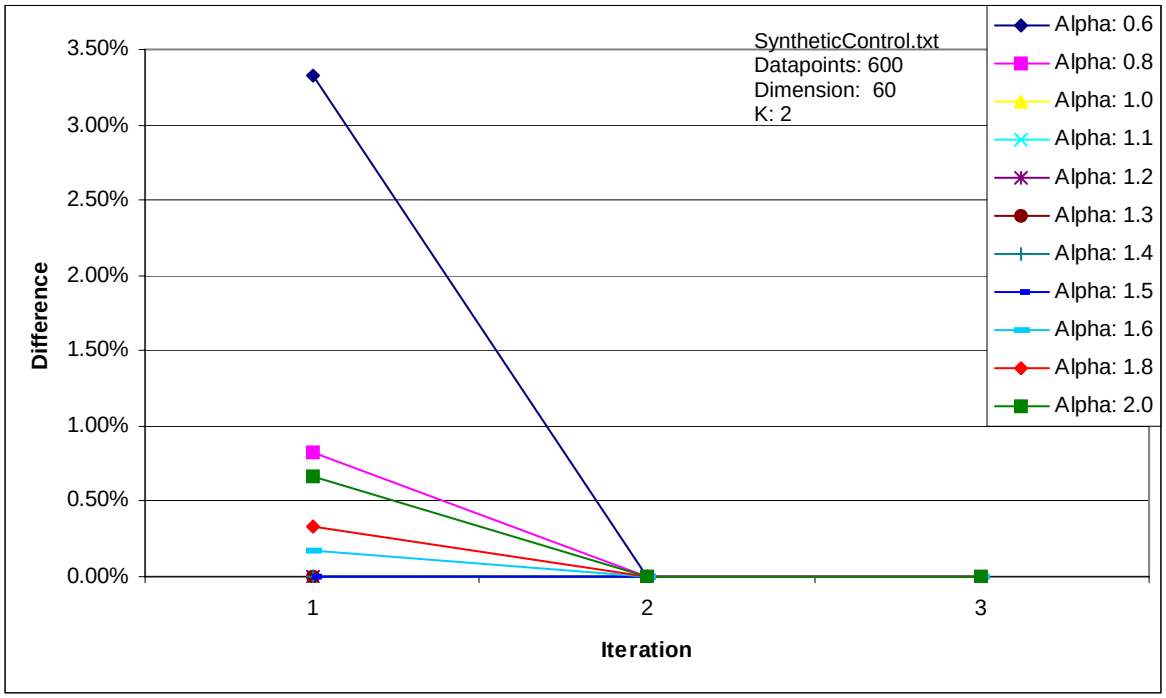

Figure I5. Plot of percentage difference between Perturbed centroid and Unperturbed centroid for SyntheticControl.txt (data size=600, dimension=60 and K=2).

Source Code for the Perturbed K-Means.

```
import java.util.*;
import java.io.*;
import java.math.*;
import java.util.Vector;
import java.text.DecimalFormat;

public class KM
{

  public static void main(String[] args) throws IOException
  {
     int kSize, pointSize, dim;
     DecimalFormat fmt = new DecimalFormat("0.#");
     fmt.setMinimumFractionDigits(1);

     String dataName = "14Housing.txt";
     kSize=2;



     FileWriter writer = new FileWriter("Output.txt");
     BufferedWriter bw = new BufferedWriter(writer);//works across different platform

     bw.newLine();
     bw.write("Data Name:  "+dataName); bw.newLine();
     bw.write("K:          "+kSize); bw.newLine();



     FileReader  file = new FileReader(dataName);
     BufferedReader buff = new BufferedReader(file);

     boolean eof = false;
     pointSize = -1;
     dim=0;
     while(!eof)
     {
       pointSize = pointSize + 1;
       String line = buff.readLine();
       if(pointSize == 1)
       {
```

```
      StringTokenizer tok = new StringTokenizer(line);
      dim = tok.countTokens();
    }

    if(line == null)
       eof = true;

}
buff.close();
System.out.println("dimension = "+dim);
System.out.println("pointSize = "+pointSize);

bw.write("Datapoints: "+pointSize); bw.newLine();
bw.write("Dimension:  "+dim);  bw.newLine();
bw.write("-------------------------------------");

double point [][] = new double[pointSize][dim];


FileReader  file2 = new FileReader(dataName);
BufferedReader buff2 = new BufferedReader(file2);
eof = false;
int kount = -1;

String line;
StringTokenizer tok2;
for(int i=0; i < pointSize; i++)
{
  line = buff2.readLine();

  tok2 = new StringTokenizer(line);
  while(tok2.hasMoreTokens())
  {

     for(int n=0; n < dim; n++)
     {
        point[i][n]=Double.parseDouble(tok2.nextToken());
        //System.out.print(" "+point[i][n]+" ");
     }
     System.out.println();
  }
}
buff2.close();
```

```
    double
dot6=0,dot8=0,one=0,one1=0,one2=0,one3=0,one4=0,one5=0,one6=0,one8=0,two=0;

    double tRuns=40;

    for(int run=1; run <= tRuns; run++)
    {

      System.out.println("Run..."+run);

      bw.newLine();
      bw.write("Run..."+run);
      bw.newLine();

      //Randomly chooses initial centroid from datapoints

      double k[][] = new double[kSize][dim];
      int integ;
      Vector indexOfPoints = new Vector();
      for(int i=0; i < pointSize; i++)
      {
        indexOfPoints.add(new Integer(i));
      }


      int ps = pointSize;
      int inte;
      int inte2;

      for(int c=0; c < kSize; c++)
      {
        inte =  (int)(Math.random()*ps);
        inte2 = ((Integer)(indexOfPoints.remove(inte))).intValue();

        ps=ps-1;

        for(int n=0; n < dim; n++)
        {
          k[c][n]=point[inte2][n];
        }
        //System.out.println();
      }
```

```
double alpha;
double P [] = {0.6,0.8,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.8,2.0};
int sP = P.length;

for(int p=0; p < sP; p++)
{
  alpha = P[p];

  int labelA [] = new int[pointSize];

  int labelB [] = new int[pointSize];



  double centroid [][] = new double[kSize][dim];

  for(int j=0; j < kSize; j++)
  {
    for(int d=0; d < dim; d++)
    {
      centroid[j][d] = k[j][d];
    }
  }

  double prevCentroid [][] = new double[kSize][dim];

  //initialize prevCentroid
  for(int j=0; j < kSize; j++)
  {
    for(int d=0; d < dim; d++)
    {
      prevCentroid [j][d] = k[j][d];
    }
  }

  double perturbed[][] = new double[kSize][dim];



  Vector clus [] = new Vector [kSize];
  for(int j=0; j < kSize; j++)
  {
    clus[j] = new Vector();
  }
```

```
double dSquared [] = new double [kSize];
int index=0;

double tem, temp;

//Initial N-N computation
for(int i=0; i < pointSize; i++)
{

  for(int j=0; j < kSize; j++)
  {
    dSquared[j]=0;
    tem=0;
    for(int d=0; d < dim; d++)
    {
      tem = tem + Math.pow((k[j][d]-point[i][d]),2);
    }

    dSquared[j] = tem;
  }

  //get nearest cluster
  temp=dSquared[0];//initialize temp
  for(int j=0; j < kSize; j++)
  {
    if(dSquared[j] <= temp)
    {
      temp=dSquared[j];
      index=j;
    }
  }
  labelA[i] = index;//label point with winning cluster

//add winning point to the winning cluster
clus[index].add(new Integer(i));//use wrapper class

}




////--------------------------------/////
//Start centroid computation //
```

```
        int iter;
        double tempr [] = new double[dim];

        boolean noChangeInMembership = false;

        iter=0;

        while(noChangeInMembership == false)
        {

          iter++;

          int sizeOfClusj = 0;//number of points in clus j

          for(int j=0; j < kSize; j++)
          {
            sizeOfClusj = clus[j].size();
            //System.out.println("Clus "+j+" has "+sizeOfClusj+ " points.");

            if(sizeOfClusj != 0)
            {
              for(int n=0; n < dim; n++)
              {
                tempr[n]=0;
                for(int s=0; s < sizeOfClusj; s++)
                {
                  tempr[n] = tempr[n] +
point[((Integer)(clus[j].elementAt(s))).intValue()][n];
                }
              }

              for(int n=0; n < dim; n++)
              {
                centroid[j][n] = tempr[n]/sizeOfClusj;
              }

              for(int n=0; n < dim; n++)
              {
                perturbed[j][n] = centroid[j][n]+(alpha-1)*(centroid[j][n]-
prevCentroid[j][n]);
              }

            }//if

          }
```

```
//clear point indexes from clusj
for(int j=0; j < kSize; j++)
{
  clus[j].clear();
}


//Nearest-neighbor

for(int i=0; i < pointSize; i++)
{

  for(int j=0; j < kSize; j++)
  {
    dSquared[j]=0;
    tem=0;
    for(int d=0; d < dim; d++)
    {
      tem = tem + Math.pow((perturbed[j][d]-point[i][d]),2);
    }

    dSquared[j] = tem;
  }

  //get nearest cluster
  temp=dSquared[0];//initialize temp

  for(int j=0; j < kSize; j++)
  {
    if(dSquared[j] <= temp)
    {
      temp=dSquared[j];
      index=j;
    }
  }

  labelB[i] = index;//label point with winning cluster,perturbed point used

  //add winning point to the winning cluster

  clus[index].add(new Integer(i));//use wrapper class
}//for
```

```
  //Compare labelA and labelB
  int count=0;
  for(int i=0; i < pointSize; i++)
  {
    if(labelA[i] == labelB[i])
      count = count + 1;
  }

  if(count == pointSize)//no more change in membership
    noChangeInMembership = true;

  else//update labelA, prevCentroid
  {
    for(int i=0; i < pointSize; i++)
    {
      labelA[i] = labelB[i];
    }

    //Update value of prevCentroid
    for(int j=0; j < kSize; j++)
    {
      for(int d=0; d < dim; d++)
      {
        prevCentroid[j][d] = centroid[j][d];
      }
    }

  }//else


}// end of while loop


System.out.println("Alpha:    "+alpha);
System.out.println("Iteration: "+iter);

bw.write("Alpha:      "+alpha);  bw.newLine();
bw.write("Iteration:   "+iter);  bw.newLine();


if(alpha == 0.6)
  dot6=dot6+iter;
else
```

```
      if(alpha == 0.8)
        dot8=dot8+iter;
      else
      if(alpha == 1.0)
        one=one+iter;
      else
      if(alpha == 1.1)
        one1=one1+iter;
      else
      if(alpha == 1.2)
        one2=one2+iter;
      else
      if(alpha == 1.3)
        one3=one3+iter;
      else
      if(alpha == 1.4)
        one4=one4+iter;
      else
      if(alpha == 1.5)
        one5=one5+iter;
      else
      if(alpha == 1.6)
        one6=one6+iter;
      else
      if(alpha == 1.8)
        one8=one8+iter;
      else
        two=two+iter;

    }//end of alpha loop

    bw.newLine();
    System.out.println();

  }//end of random runs

  //System.out.println();



 // bw.newLine();



  System.out.println("alpha=.6  : "+dot6/tRuns);
```

```
        System.out.println("alpha=.8  : "+dot8/tRuns);
        System.out.println("alpha=1.0 : "+one/tRuns);
        System.out.println("alpha=1.1 : "+one1/tRuns);
        System.out.println("alpha=1.2 : "+one2/tRuns);
        System.out.println("alpha=1.3 : "+one3/tRuns);
        System.out.println("alpha=1.4 : "+one4/tRuns);
        System.out.println("alpha=1.5 : "+one5/tRuns);
        System.out.println("alpha=1.6 : "+one6/tRuns);
        System.out.println("alpha=1.8 : "+one8/tRuns);
        System.out.println("alpha=2.0 : "+two/tRuns);


        bw.write("Average Iteration:");       bw.newLine();
        bw.write("----------------------");    bw.newLine();

        bw.write("alpha=.6  : "+dot6/tRuns);  bw.newLine();
        bw.write("alpha=.8  : "+dot8/tRuns);  bw.newLine();
        bw.write("alpha=1.0 : "+one/tRuns);   bw.newLine();
        bw.write("alpha=1.1 : "+one1/tRuns);  bw.newLine();
        bw.write("alpha=1.2 : "+one2/tRuns);  bw.newLine();
        bw.write("alpha=1.3 : "+one3/tRuns);  bw.newLine();
        bw.write("alpha=1.4 : "+one4/tRuns);  bw.newLine();
        bw.write("alpha=1.5 : "+one5/tRuns);  bw.newLine();
        bw.write("alpha=1.6 : "+one6/tRuns);  bw.newLine();
        bw.write("alpha=1.8 : "+one8/tRuns);  bw.newLine();
        bw.write("alpha=2.0 : "+two/tRuns);



        bw.close();
    }//method main


}//class KM
```

VITA

Sumakwel Muralla

Candidate for the Degree

Master of Science

Thesis: A METHOD OF ACCELERATING K-MEANS BY DIRECTED PERTURBATION OF THE CODEVECTORS

Major Field: Computer Science

Education: Graduated from the University of the Philippines, Diliman, Quezon City, Manila, Philippines on April 1989; received a degree in Bachelor of Science in Civil Engineering; completed the requirements for a Master of Science Degree in Computer Science at Oklahoma State University in July 2006.

Experience: Worked as a practicing professional engineer from 1989-2000 in Manila. Currently works as a programmer at the Information Systems Division, Walmart, Main Office, Bentonville, Arkansas.

Name: Sumakwel R. Muralla Date of Degree: July, 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A METHOD OF ACCELERATING K-MEANS BY DIRECTED PERTURBATION OF THE CODEVECTORS

Pages in Study: 97 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: K-Means clustering algorithm is a simple and yet very powerful technique of partitioning datasets. This study presents a method of decreasing the total computational iterations needed to run K-Means. This is done by adding perturbations to the cluster centroids and using the perturbed centroids as the new seed values for computing the next codevectors. The method was tested using synthetic and real-world data sets.

Findings and Conclusions:Running the new method on these datasets significantly improved the performance of K-Means by reducing the total computational iterations needed to run K-Means. Furthermore, the quality of the final cluster centroids was preserved under the new method. Based on these findings, the new method is very promising and deserves further testing by subjecting it to a variety of data sets before it is recommended for general greater use.

ADVISOR'S APPROVAL: Douglas R. Heisterkamp