Technical University of Crete
School of Electrical and Computer Engineering

Course: **Randomized Algorithms**

Exercise 4

Angelopoulos Dimitris - 2020030038

Dimensionality Reduction

In the first part we will study the Johnson-Lindenstrauss Lemma. First of all, we will generate a random data matrix $\mathbf{X} \in \mathbb{R}^{d \times m}$ and compute all the pairwise distances $\|\mathbf{x}_i - \mathbf{x}_j\|$ where $i, j = 1, \ldots, m$.

After computing these distances, we will use the J-L Lemma to find the reduced dimension n. For given ϵ, δ and $|\mathbf{Q}| = \frac{m(m-1)}{2}$. The reduced dimension is given by:

$$n = \frac{6 \log(2|\mathbf{Q}|/\delta)}{\epsilon^2}$$

Generating a random dimensionality reduction matrix $\mathbf{W} \in \mathbb{R}^{n \times d}$ and applying it to the data matrix, we get $\mathbf{Y} = \mathbf{W}\mathbf{X}$. We then compute the pairwise distances of the reduced data matrix $\mathbf{Y}$, $\|\mathbf{y}_i - \mathbf{y}_j\|$ where $i, j = 1, \ldots, m$.

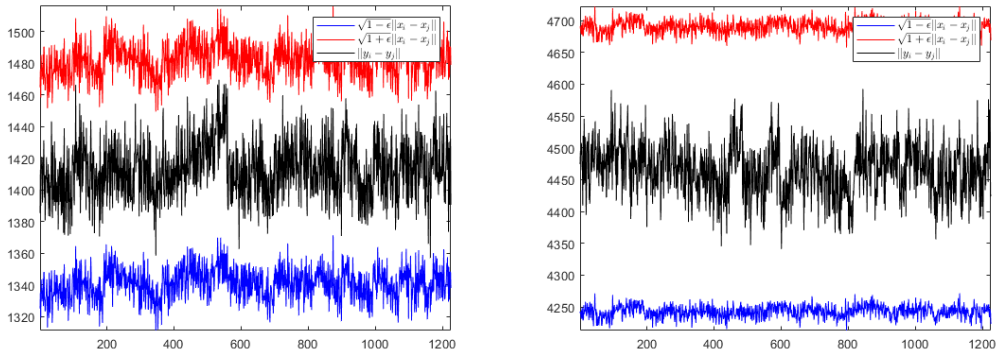

Figure 1: Both figures correspond to $m = 50$ data points in dimensions $d = 10^4, 10^5$ respectively.

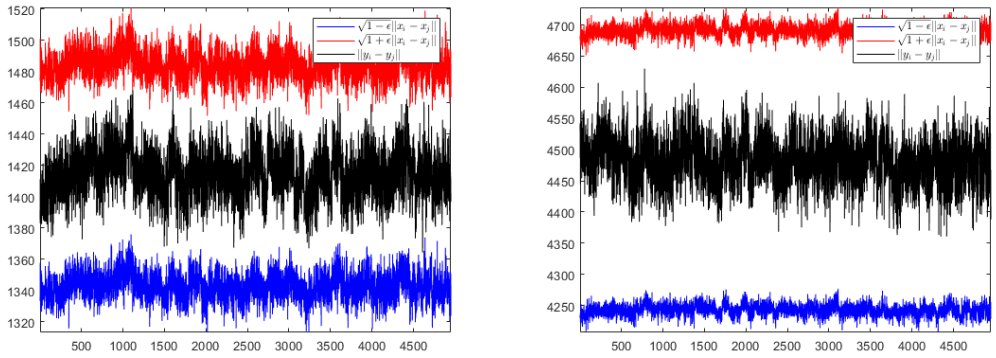

Figure 2: Both figures correspond to $m = 100$ data points in dimensions $d = 10^4, 10^5$ respectively.

For both cases we can see that the pairwise distances are bounded by the the quantities $\sqrt{1 \pm \epsilon}\|\mathbf{x}_i - \mathbf{x}_j\|$. That means that we do not lose any information. One more way to validate our results comes from computing the empirical probability of success. We found that this probability is always 1 throughout our experiments.

Randomized 2-SAT Algorithm

In the second part of this exercise we will generate random (satisfiable) 2-clauses and test whether our algorithm finds a solution to these random expressions.

To generate satisfiable clauses, we start by defining some random expression (without negations) and some random arbitrary truth assignment $\mathbf{s} \in \{0, 1\}^n$. Using the aforementioned we generate a matrix `lit_clauses` (which is complementary to `clauses`) and denotes the positions where negations occur. After generating this framework, we write a function that checks whether some truth assignment $\mathbf{x} \in \{0, 1\}^n$ is a solution to the expression. Now we have all the tools we need to develop the algorithm.

First of all, we will test that the algorithm generates a correct answer to a simple expression, such :

$$(\neg x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$$

The algorithm indeed finds some assignment that satisfies this formula.

Now we proceed with testing the algorithm for larger parameters. More specifically, we fixed $n = 100$ and $K = 100n$ and started to experiment with the parameter m. We observed that for $m \geq 1$ the algorithm always finds a solution. Likewise for $m \to 0$ the algorithm never finds a solution. Having said all that we started experimenting with values of $m \in (0, 1)$. For $m = n^{-1}$ the algorithm almost always fails to find a satisfiable assignment. Increasing the value of the power we observe that the algorithm start to fail less. For example, for $m = n^{-3/4}$ the algorithm will return both failures and successes.