

# Updating Kernel Methods in Spectral Decomposition by Affinity Perturbations

Yaniv Shmueli<sup>a,\*</sup>, Guy Wolf<sup>a</sup>, Amir Averbuch<sup>a</sup>

<sup>a</sup>*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel*

---

## Abstract

Many machine learning based algorithms contain a training step that is done once. The training step is usually computational expensive since it involves processing of huge matrices. If the training profile is extracted from an evolving dynamic dataset, it has to be updated as some features of the training dataset are changed. This paper proposes a solution how to update this profile efficiently. Therefore, we investigate how to update the training profile when the data is constantly evolving. We assume that the data is modeled by a kernel method and processed by a spectral decomposition. In many algorithms for clustering and classification, a low dimensional representation of the affinity (kernel) graph of the embedded training dataset is computed. Then, it is used for classifying newly arrived data points. We present methods for updating such embeddings of the training datasets in an incremental way without the need to perform the entire computation upon the occurrences of changes in a small number of the training samples. Efficient computation of such an algorithm is critical in many web based applications.

*Keywords:* Perturbation theory, Eigenvalue problem, Diffusion maps, Dimensionality reduction

---

## 1. Introduction

Studying a dataset by extracting constructive information from it is a challenging task. The computational complexity increases when we process evolving data that requires frequent updates of the profile that represents the training set we use. As time advances, the training profile, which was previously extracted from evolving dynamic data, may not represent accurately the behavior of the current data. Therefore, the extracted profile has to be updated frequently.

A straightforward approach to update the training profile is to repeat the whole computational process that generated previously this training profile.

---

\*Yaniv Shmueli, Tel: +972-54-6900863, Fax: +972-3-6422020

*Email addresses:* yanivshm@post.tau.ac.il (Yaniv Shmueli), guy.wolf@cs.tau.ac.il (Guy Wolf), amir@math.tau.ac.il (Amir Averbuch)

However, it becomes computationally impractical when dealing with large-scale data while the changes in the training data are small. For example, consider a face recognition application. Assume the training dataset reflects many facial features such as color, glasses, haircut, age, etc. Assume some of the features (not all of them) were modified slightly. This happens often. The paper proposes a method how can we update efficiently the training profile while performing a limited and efficient computation that takes into consideration only the current modified features and not taking all the features.

A common practice in kernel methods is to extract features from a large finite high dimensional dataset that becomes a training dataset. Then, a similarity graph between the features in the training dataset is formed. We will use the Diffusion Maps (DM) methodology [1, 2] as our exemplary kernel method (see section 3.1 for the description of this method) to compute an embedding of this graph into a low dimensional space. This embedding is accomplished by eigenvectors computation of the graph affinity matrix, which belongs to the largest eigenvalues. Changes in the affinity matrix will result in changes in the eigenvectors, thus, it will force us to compute them frequently. In this work, we propose a solution that is based on the Power Iteration algorithm combined with the first order approximation of the perturbed eigenvectors and eigenvalues (eigenpairs) that enables us to update the training profile dataset by considering only the changes in the training dataset. By using ideas from perturbation theory, we update the eigenpairs that are based on the perturbations (changes) in the affinity matrix that eventually require less computational efforts. We tested our algorithm on affinity matrices that were generated by the diffusion operator of the training similarity graph in the DM methodology.

Consider a set of  $n$  sensors. Each sensor measures  $m$  different parameters (features). Suppose that we get the measurement data matrix  $X$  of size  $n \times m$ , where the cell  $(i, j)$  represents the  $j$  measurement of sensor  $i$ . Therefore, each sensor is a data point in  $\mathbb{R}^m$ . We want to embed the sensed data in a low dimensional space for clustering and classification [3]. We can also predict the parameters values in locations nearby our sensors by using out-of-sample extension methods [4, 5, 6]. Embedding the data into a low dimensional space is usually done by computing the Singular Value Decomposition (SVD) of the sensors affinity matrix or by some variation of it (kernel matrix, probability matrix or diffusion operator matrix). The values of the eigenvectors are used as the coordinates in the low dimensional space. This affinity can be defined in several ways and it usually depends on the measurements types. DM, for example, provides an affinity among the features (sensor measurements). Since sensor data is dynamic and evolving, the embedded low dimensional space have to be updated as the training data does not represent adequately the incoming data that did not participate in the training phase. Even if most of the sensors readings were unchanged, we will still need to preform the entire computation since we cannot determine the effect of such a change on the embedded space. Therefore, the goal of the paper is to provide an efficient way to update the embedding coordinates without the need to re-compute the entire SVD again and again provided that the sensors reading are regarded as the perturbations

from the original readings of the sensors affinity matrix. The perturbations are assumed to be sufficiently small.

The paper has the following structure: Related works are described in Section 2. Section 3 provides a formal definition of the problem. The main algorithm and its validity are given in Section 4. Experimental results are presented in Section 5.

## 2. Related Work

There are several works [7] that describe how to adapt elements from matrix perturbation theory to achieve eigenpairs approximation. Many of these works focus on updating the left principal eigenvector  $\pi$  of a stochastic matrix  $P$  where  $\pi = \pi P$  for eigenvalue 1. Here,  $\pi$  is the stationary distribution of a Markov chain defined by  $P$ . By the Ergodic Theorem for Markov chains,  $\pi$  is unique if  $P$  is aperiodic and irreducible [8]. The above updating methods can help to accelerate algorithms such as Pagerank and HIT (see [9] for more details) that use the stationary distribution values as rating scores. Recently, new algorithms [8, 10] were introduced to further improve the convergence rate of Google’s Pagerank algorithm. These types of methods are suitable only for updating the first eigenvector of the perturbed matrix, whereas we have to update the first  $k$  dominant eigenvectors to create the embedding.

Another way to approximate the eigenvectors is by using the group inverse of  $A$  [11]. The group inverse  $A^\#$  of  $A$  is defined as the matrix that satisfies  $AA^\#A = A$ ,  $A^\#AA^\# = A^\#$ , and  $AA^\# = A^\#A$ . If the matrix  $A$  is non-singular then  $A^\# = A^{-1}$ . By using the group inverse we can approximate the perturbed eigenvectors to be  $\tilde{\phi}_i = \phi_i - (A - \lambda_i I)^\# \tilde{A} \phi_i$ . However, calculating the group inverse is not trivial even if we only compute its approximation. Computing it for each new perturbation update of  $A$  is inefficient.

Alternatively, random algorithms such as [12] are proved to be effective in a direct computation of the SVD approximation for large scale matrices. These methods treat the perturbed matrix  $\tilde{A}$  as a new SVD approximation problem and neither use  $A$  nor its eigenpairs. Traditional computational methods such as the Power Iteration [13], Inverse Iteration and the Lanczos [14] methods operate in the same way and compute the eigenpairs of each update of the perturbed matrix. Here, the computation is preformed with a random guess as the initial input without taking the unperturbed matrix and its eigenpairs into consideration.

Incremental versions of low dimensional embedding algorithms were tailored specifically to fit Local Linear Embeddings (LLE) [15] and ISOMAP [16]. These algorithms utilize manifold learning methods. They modify the original LLE and ISOMAP algorithms to process the data iteratively rather than by batch processing. When a new data point arrives, these algorithms add it to the embedding and then efficiently update all the existing data points in the low dimensional space.

### 3. Problem Description

#### 3.1. Finding a low dimensional embedded space

We are given a set of data points  $x_i$  in  $\mathbb{R}^D$ . We want to find a set of data points  $y_i$  in  $\mathbb{R}^d, d \ll D$ , that preserves up to a small controlled distortion the affinities between them. In other words, nearby data points remain nearby while distant data points remain distant. This general framework has three steps:

1. Build a graph  $G(V, E)$  which represents the data points  $x_i$ . Nearby points are connected with an edge.
2. Build a similarity matrix using weights on the edges  $E$  in  $G$ .
3. Build an embedded graph by using the eigenvectors of the distance matrix.

The changes between each dimensionality reduction method are driven from the way we build the similarity matrix in step 2.

In words, we build an observational method of the inspected data that can be a large computer network for example. The fundamental ingredient is the ability to organize and model the data into a simple (reduced dimension) geometry. Vectors of observations on the data are collected. They are organized as a graph in which various vectors of observations are linked by their similarity (affinity). Then, a second graph is built in which the actual entries in the observation vector are linked through their mutual dependence. Spectral and harmonic analysis of the similarity matrix (or dependence matrix) is performed thus enabling the organization of the empirical observations into simpler low dimensional structures. Nonlinear extension of conventional linear statistical tools such as principal components analysis (PCA), and independent components analysis (ICA) are used. These methods reduce the observed data to allow a small number of parameters (coordinates) to model all the variabilities in the observations. A robust similarity relationship between two observation vectors is computed as a combination of all chains of pairs that link them. In the DM case ([2]), these are the diffusion inference metrics. Clustering in this metric leads to robust clustering of the observations and their characterization. Various local criteria of linkage between observations lead to distinct geometries. In these geometries, the user can redefine relevance and filter away unrelated information. The top eigenfunctions of the matrix define the pair linkages to provide global organization of the given set of observations. DM embeds the data into a low dimensional Euclidean space that converts isometrically the (diffusion) relational inference metric to the corresponding Euclidean distance. Diffusion metrics can be computed efficiently as an ordinary Euclidean distance in a low dimensional embedding by the DM. Total computational time scales linearly with the data size we get, and it can be updated on line. The diffusion geometry, which is induced by various chains of inference, enables a multiscale hierarchical organization of regional folders of observations corresponding to various states of the network [17].

To better understand the proposed algorithm, we review the Diffusion Maps (DM) methodology [2, 1] that performs non-linear dimensionality reduction.

Given our sensor reading matrix  $X$ , we define a weighted graph over the sensor set, where the weight between sensor  $i$  and  $j$  is given by the kernel

$$k(i, j) \triangleq e^{-\frac{\|x_i - x_j\|}{\varepsilon}}. \quad (3.1)$$

The degree of a sensor (vertex)  $i$  in this graph is

$$d(i) \triangleq \sum_j k(i, j). \quad (3.2)$$

Normalizing the kernel with this degree produces an  $n \times n$  row stochastic transition matrix whose cells are  $[P]_{ij} = p(i, j) = k(i, j)/d(i)$  for sensors  $i$  and  $j$ . This defines a Markov process over the sensors set.

The dimensionality reduction achieved by this diffusion process is a result of the spectral analysis of the kernel. Thus, it is preferable to work with a symmetric conjugate to  $P$  that we denote by  $A$  and its cells are denoted by

$$[A]_{ij} = a(i, j) = \frac{k(i, j)}{\sqrt{d(i)}\sqrt{d(j)}} = \sqrt{d(i)}p(i, j)\frac{1}{\sqrt{d(j)}}. \quad (3.3)$$

The eigenvalues  $1 = \lambda_1 \geq \lambda_2 \geq \dots$  of  $A$  and their corresponding eigenvectors  $\phi_k$  ( $k = 1, 2, \dots$ ) are used to obtain the desired dimensionality reduction by mapping each  $i$  onto the data point  $\Phi(i) = (\lambda_2\phi_2(i), \lambda_3\phi_3(i), \dots, \lambda_\delta\phi_\delta(i))$  for a sufficiently small  $\delta$ , which is dependent on the decay of the spectrum of  $A$ .

### 3.2. Updating the Embedding

We are given the perturbation matrix  $\tilde{A}$  of the matrix  $A$ . We can assume that the perturbations are sufficiently small, that is  $\|\tilde{A} - A\| < \varepsilon$  for some small  $\varepsilon$ . We also assume that  $\tilde{A}$  is symmetric since we compute it in the same way as  $A$  was computed. We wish to update the eigenpairs of  $\tilde{A}$  based on  $A$  and its eigenpairs. We now present the problem in mathematical terms.

Given a symmetric  $n \times n$  matrix  $A$  where its  $k$  dominant eigenvalues are  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k$  and its eigenvectors are  $\phi_1, \phi_2, \dots, \phi_k$ , respectively, and a perturbed matrix  $\tilde{A}$  such that  $\|\tilde{A} - A\| < \varepsilon$ , find the perturbed eigenvalues  $\tilde{\lambda}_1 \geq \tilde{\lambda}_2 \geq \dots \geq \tilde{\lambda}_k$  and its eigenvectors  $\tilde{\phi}_1, \tilde{\phi}_2, \dots, \tilde{\phi}_k$  of  $\tilde{A}$  in the most efficient way.

## 4. The Recursive Power Iteration (RPI) Algorithm

### 4.1. First Order Approximations

To efficiently update each eigenpair of the perturbed matrix  $\tilde{A}$ , we will first compute the first order approximation of each eigenpair. Later, it will be used in our algorithm as the initial guess to the RPI algorithm.

Given an eigenpair  $(\phi_i, \lambda_i)$  of a symmetric matrix  $A$  where  $A\phi_i = \lambda_i\phi_i$ , we compute the first order approximation of the eigenpair of the perturbed matrix  $\tilde{A} = A + \Delta A$ . We assume that the change  $\Delta A$  is sufficiently small, which result

in a small perturbation in  $\phi_i$  and  $\lambda_i$ . We look for  $\Delta\lambda_i$  and  $\Delta\phi_i$  that satisfy the equation

$$(A + \Delta A)(\phi_i + \Delta\phi_i) = (\lambda_i + \Delta\lambda_i)(\phi_i + \Delta\phi_i). \quad (4.1)$$

This equation is expanded to

$$A\phi_i + [\Delta A]\phi_i + A[\Delta\phi_i] + [\Delta A][\Delta\phi_i] = \lambda_i\phi_i + \lambda_i[\Delta\phi_i] + [\Delta\lambda_i]\phi_i + [\Delta\lambda_i][\Delta\phi_i].$$

It becomes

$$[\Delta A]\phi_i + A[\Delta\phi_i] = \lambda_i[\Delta\phi_i] + [\Delta\lambda_i]\phi_i + O(\Delta^2). \quad (4.2)$$

For the rest of the computation, we will ignore the term  $O(\Delta^2)$  as it provides relatively small error. Since  $A$  is symmetric, its eigenvectors are orthogonal and can be used as a basis for the perturbed eigenvector  $\Delta\phi_i = \sum_{j=1}^N \epsilon_{ij}\phi_j$  for some constants  $\epsilon_{ij}$ . Substituting this construction in Eq. 4.2, we get

$$[\Delta A]\phi_i + A\left(\sum_{j=1}^N \epsilon_{ij}\phi_j\right) = \lambda_i\left(\sum_{j=1}^N \epsilon_{ij}\phi_j\right) + [\Delta\lambda_i]\phi_i,$$

or

$$[\Delta A]\phi_i + \sum_{j=1}^N \epsilon_{ij}A\phi_j = \lambda_i\left(\sum_{j=1}^N \epsilon_{ij}\phi_j\right) + [\Delta\lambda_i]\phi_i.$$

By using that fact that  $A\phi_j = \lambda_j\phi_j$ , we get

$$[\Delta A]\phi_i + \sum_{j=1}^N \epsilon_{ij}\lambda_j\phi_j = \lambda_i\left(\sum_{j=1}^N \epsilon_{ij}\phi_j\right) + [\Delta\lambda_i]\phi_i. \quad (4.3)$$

We can now simplify Eq. 4.3 by multiplying both sides by  $\phi_i^T$

$$\phi_i^T[\Delta A]\phi_i + \sum_{j=1}^N \epsilon_{ij}\lambda_j\phi_i^T\phi_j = \lambda_i\left(\sum_{j=1}^N \epsilon_{ij}\phi_i^T\phi_j\right) + [\Delta\lambda_i]\phi_i^T\phi_i.$$

By using the fact that for  $j \neq i$ ,  $\phi_j$  are orthogonal to  $\phi_i$ , we get

$$\phi_i^T[\Delta A]\phi_i + \epsilon_{ii}\lambda_i\phi_i^T\phi_i = \lambda_i\epsilon_{ii}\phi_i^T\phi_i + [\Delta\lambda_i]\phi_i^T\phi_i,$$

and since  $\phi_i^T\phi_i = 1$  the equation becomes

$$[\Delta\lambda_i] = \phi_i^T[\Delta A]\phi_i. \quad (4.4)$$

By multiplying both sides of Eq. 4.3 by  $\phi_k^T$ ,  $k \neq i$ , we get

$$\phi_k^T[\Delta A]\phi_i + \sum_{j=1}^N \epsilon_{ij}\lambda_j\phi_k^T\phi_j = \lambda_i\left(\sum_{j=1}^N \epsilon_{ij}\phi_k^T\phi_j\right) + [\Delta\lambda_i]\phi_j^T\phi_i.$$

Therefore,

$$\phi_k^T [\Delta A] \phi_i + \epsilon_{ik} \lambda_k \phi_k^T \phi_k = \lambda_i [\epsilon_{ik} \phi_k^T \phi_k] + [\Delta \lambda_i] \phi_k^T \phi_i.$$

Since  $\phi_k^T \phi_k = 1$  and  $\phi_k^T \phi_i = 0, k \neq i$ , we get

$$\phi_k^T [\Delta A] \phi_i + \epsilon_{ik} \lambda_k = \epsilon_{ik} \lambda_i + 0,$$

which yields

$$\epsilon_{ik} = \frac{\phi_k^T [\Delta A] \phi_i}{\lambda_i - \lambda_k}. \quad (4.5)$$

Finally, we require that the perturbed eigenvectors will also be an orthonormal basis and  $(\phi_i + [\Delta \phi_i])^T (\phi_i + [\Delta \phi_i]) = 1$ . If we expand this equation, we get

$$\phi_i^T \phi_i + 2\phi_i^T [\Delta \phi_i] + [\Delta \phi_i]^T [\Delta \phi_i] = 1.$$

After the removal of high order terms, we get  $1 + 2\phi_i^T [\Delta \phi_i] = 1$ , or  $\phi_i^T [\Delta \phi_i] = \langle \phi_i, [\Delta \phi_i] \rangle = 0$ . Since this product is  $\epsilon_{ii}$ , therefore  $\epsilon_{ii} = 0$ . Using  $\epsilon_{ii} = 0$  and Eq. 4.5, then  $[\Delta \phi_i]$  becomes

$$[\Delta \phi_i] = \sum_{j=1}^N \epsilon_{ij} \phi_j = \sum_{j \neq i} \frac{\phi_j^T [\Delta A] \phi_i}{\lambda_i - \lambda_j} \phi_j.$$

To conclude, we obtained that the following first order approximations for the eigenvalues and eigenvectors of  $\tilde{A}$  by

$$\tilde{\lambda}_i = \lambda_i + \phi_i^T [\Delta A] \phi_i \quad (4.6)$$

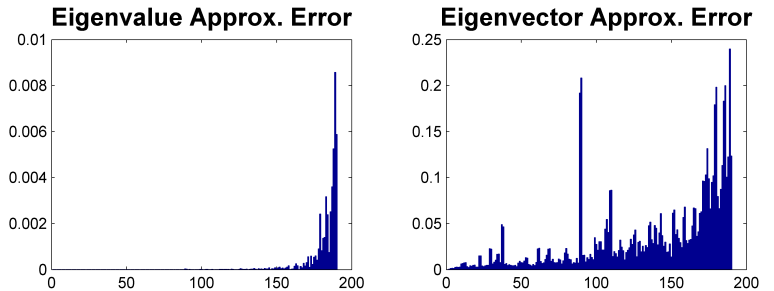
and

$$\tilde{\phi}_i = \phi_i + \sum_{j \neq i} \frac{\phi_j^T [\Delta A] \phi_i}{\lambda_i - \lambda_j} \phi_j. \quad (4.7)$$

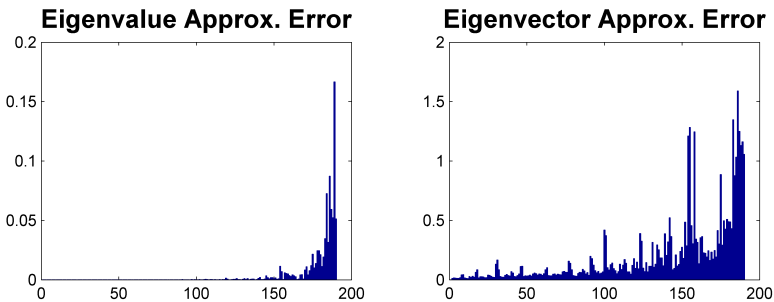
The relative error of the perturbed  $\tilde{x}$  from  $x$  is defined by

$$err_x = \frac{\|\tilde{x} - x\|_{L_2}}{\|x\|_{L_2}}. \quad (4.8)$$

To analyze the accuracy of these approximations, we calculated the relative error of a  $200 \times 200$  random matrix  $A$ , where  $[A]_{ij} \sim U(0, 1)$ . The perturbed matrix  $\tilde{A}$  was created by adding some noise to the elements of  $A$  at a rates of 1% and 5%.



(a) Eigenpairs approximation errors for 1% perturbations



(b) Eigenpairs approximation errors for 5% perturbations

Figure 4.1: Approximation error rates. The  $x$ -axis is the index of the ordered eigenvalues. The  $y$ -axis is the relative error of the approximated value (Eq. 4.8).

We can see that the error increases for eigenvectors that correspond to eigenvalues of smaller magnitudes. In other words, the error does not affect the largest eigenpairs. Since the embedding into a lower dimension space by DM depends on a few largest eigenvalues, the error does not affect the quality and the validity of the lower dimension space as a faithful representation of the original source space.

While this method provides a fast computation for the perturbed values, it is limited as it only uses the first order approximations, which might not be sufficiently accurate for our needs.

#### 4.2. The Recursive Power Iteration Method

Power Iteration method has proved to be effective when calculating the principle eigenvector of a matrix [8]. However, this method cannot find the other eigenvectors of the matrix. In general, an initial guess of the eigenvector is also important to guarantee fast convergence of the algorithm. In the algorithm, which we call Recursive Power Iteration (RPI), the original eigenvectors of  $A$  will be the initial guess for each power iteration (eventually this choice will be refined in Section 4.3). Once the eigenvector  $\tilde{\phi}_i$  is obtained in step  $i$ , we transform  $\tilde{A}$  into a matrix that has  $\tilde{\phi}_{i+1}$  as its principle eigenvector. We iterate this

step until we recover the  $k$  dominant eigenvectors of  $\tilde{A}$ .

---

**Algorithm 4.1:** Recursive Power Iteration Algorithm

---

**Input:** Perturbed symmetric matrix  $\tilde{A}_{n \times n}$ , number of eigenvectors to calculate  $k$ , initial eigenvectors guesses  $\{v_i\}_{i=1}^k$ , admissible error  $err$

**Output:** Approximated eigenvectors  $\{\tilde{\phi}_i\}_{i=1}^k$ , approximated eigenvalues

$\{\tilde{\lambda}_i\}_{i=1}^k$

- 1: **for**  $i = 1 \rightarrow k$  **do**
- 2:    $\phi \leftarrow v_i$
- 3:   **repeat**
- 4:      $\phi_{next} \leftarrow \frac{\tilde{A}\phi}{\|\tilde{A}\phi\|}$
- 5:      $err_\phi \leftarrow \|\phi - \phi_{next}\|$
- 6:      $\phi \leftarrow \phi_{next}$
- 7:   **until**  $err_\phi \leq err$
- 8:    $\tilde{\phi}_i \leftarrow \phi$
- 9:    $\tilde{\lambda}_i \leftarrow \frac{\tilde{\phi}_i^T \tilde{A} \tilde{\phi}_i}{\tilde{\phi}_i^T \tilde{\phi}_i}$
- 10:  $\tilde{A} \leftarrow \tilde{A} - \tilde{\phi}_i \tilde{\lambda}_i \tilde{\phi}_i^T$
- 11: **end for**

---

The correctness of the RPI algorithm is proved based on the fact that the power iteration method converges and on the spectral decomposition properties of  $\tilde{A}$ .

**Proposition 4.1.** *Algorithm 4.1 finds the first  $k$  eigenpairs of  $\tilde{A}$ .*

*Proof.* We prove the correctness of the algorithm by induction on  $k$  in the algorithm.

For  $k = 1$ , we apply the power iteration method on  $\tilde{A}$  that converges to the principle component  $\tilde{\phi}_1$  with its corresponding eigenvalue  $\tilde{\lambda}_1$ .  $\tilde{\lambda}_1$  is the largest eigenvalue of  $\tilde{A}$ .

Let us assume that the algorithm found the first  $k$  eigenpairs of  $\tilde{A}$ . In each step, we subtract the matrix  $\tilde{\phi}_i \tilde{\lambda}_i \tilde{\phi}_i^T$  from  $\tilde{A}$ . Then, in step  $k + 1$ , we apply the power iteration loop to the matrix  $B = \tilde{A} - \sum_{i=1}^k \tilde{\phi}_i \tilde{\lambda}_i \tilde{\phi}_i^T$ .  $\tilde{A}$  is symmetric and has a spectral decomposition of the form  $\tilde{A} = \sum_{i=1}^n \tilde{\phi}_i \tilde{\lambda}_i \tilde{\phi}_i^T$ , where  $\tilde{\phi}_i, \tilde{\lambda}_i$  are the eigenpairs of  $\tilde{A}$ . Therefore,  $B = \sum_{i=k+1}^n \tilde{\phi}_i \tilde{\lambda}_i \tilde{\phi}_i^T$ . Since  $\tilde{\lambda}_{k+1} \geq \tilde{\lambda}_{k+2} \geq \dots \geq \tilde{\lambda}_n$ , the principle component of  $B$  is  $\tilde{\phi}_{k+1}$ . This principle component is found once we apply the power iteration method to  $B$ , which is exactly what happens in step  $k + 1$ . Therefore, in step  $k + 1$  of the algorithm, the power iteration method will recover the eigenvector  $\tilde{\phi}_{k+1}$  of  $\tilde{A}$ . After  $k + 1$  steps, the algorithm recovers the  $k + 1$  dominant eigenpairs of  $\tilde{A}$ .  $\square$

To analyze the computational complexity of the RPI algorithm, we observe

that during the execution of the algorithm, we perform  $(I_1 + \dots + I_k)C_{\tilde{A}}$  operations, where  $I_m$  is the number of iterations needed in step  $m$  and  $C_{\tilde{A}}$  is the cost of applying the matrix  $\tilde{A}$  to a vector followed by its normalization. We also need to update  $\tilde{A}$  during the  $k$  steps which costs  $kn^2$  operations. Therefore, the total complexity of this algorithm is  $O(kn^2 + (I_1 + \dots + I_k)C_{\tilde{A}})$ .

#### 4.3. RPI Algorithm that Uses the First Order Approximations

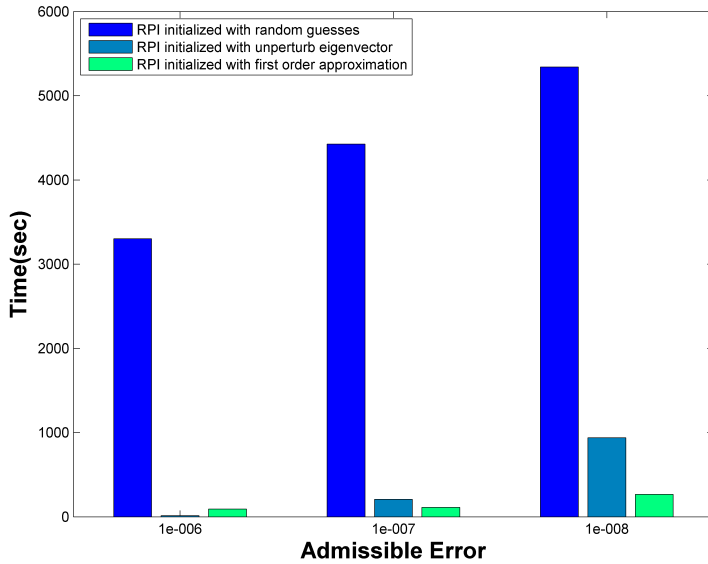
The RPI finds in each step the principle eigenvector of the modified  $\tilde{A}$  by iterating the equation  $v_{k+1} = \frac{Av_k}{\|Av_k\|}$ . The convergence rate depends on the initial guess which we provide to the iteration loop. Algorithm 4.1 uses the unperturbed eigenvector as the initial guess. To improve the convergence rate, we apply Algorithm 4.1 but use the eigenvectors of the first order approximation, which were computed in section 4, as our initial guess.

The justification for this approach is that the first order approximation of the perturbed eigenvector is inexpensive, and each RPI step will guarantee that this approximation converges to the actual eigenvector of  $\tilde{A}$ . The first order approximation should be close to the actual solution we seek and therefore requires fewer iterations steps to converge. A comparison between the number of iteration needed to compute the eigenpairs is given in Section 5 for different variations of the RPI algorithm.

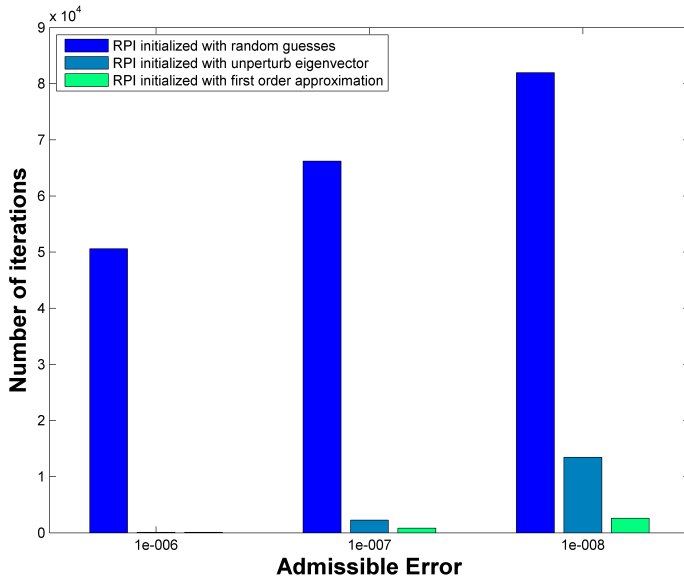
The fact that Algorithm 4.1 iterates over  $k$  to recover the perturbed eigenvectors can assist us in automatically selecting the number of eigenvectors to approximate, which is the dimension of the embedded space. If we approximate the perturbed matrix  $\tilde{A}$  by the first  $k$  eigenvectors that form the matrix  $\tilde{A}_k$ , which has rank  $k$ , we know that the approximated error is  $\|\tilde{A} - \tilde{A}_k\| \leq \lambda_k$ . Since in each iteration we also calculate the eigenvalues, we know to stop when  $\lambda_k$  is sufficiently small. This is an advantage over methods that compute the SVD approximation of  $\tilde{A}$  that sometimes require  $k$  as an a priori input parameter.

## 5. Experimental Results

We compare the execution time as well as the total number of iterations  $\sum_{i=1}^k C_i$  of the three variations of the RPI algorithm. The first algorithm uses a random vector as the initial guess in each step. The second algorithm uses the unperturbed (known) vector of  $A$  as the initial guess. The third algorithm uses the first order approximation of the perturbed eigenvector of  $\tilde{A}$  as the initial guess. In this benchmark, we used a  $10^4 \times 10^4$  perturbed diffusion matrix from the DM methodology. The first  $k = 10$  eigenpairs are computed. We compared the results for different admissible errors  $\|\phi - \phi_{next}\|_{L_2}$ .



(a) Comparison of the total wallclock time between different RPI algorithms.



(b) Comparison of the total number of iterations between different RPI algorithms.

Figure 5.1: Performance comparison between the three RPI algorithmic variations. We compute the first 10 eigenpairs of a  $10^4 \times 10^4$  matrix. Each bar represents a variation of the RPI algorithm. Each group of bars is compared within a given admissible error.

As we can see, the RPI algorithm with the first order approximation has the lowest total running time and also needs fewer number of iterations to complete.

## 6. Conclusions

In this work, we presented several contributions. The dominant eigenpairs error is relatively small when the first order approximation is used as the initial guess in the computation. We presented the RPI algorithm, which uses the Power Iteration method, to compute the first  $k$  eigenpairs of a perturbed matrix. The algorithm uses the given eigenpairs of the unperturbed matrix. We improved the algorithm by using the first order approximation of the eigenpairs as the initial guess and showed that it accelerates the convergence rate of each iteration. After proving the correctness of the algorithm, we showed that it also preforms well on real data.

## Acknowledgment

This research was partially supported by the Israel Science Foundation (Grant No. 1041/10).

## References

- [1] S. Lafon, Diffusion maps and geometric harmonics, Ph.D. thesis, Yale University (May 2004).
- [2] R. Coifman, S. Lafon, Diffusion maps, *Applied and Computational Harmonic Analysis* 21 (1) (2006) 5–30.
- [3] G. David, Anomaly detection and classification via diffusion processes in hyper-networks, Ph.D. thesis, School of Computer Science, Tel Aviv University (March 2009).
- [4] R. Coifman, S. Lafon, Geometric harmonics: A novel tool for multiscale out-of-sample extension of empirical functions, *Applied and Computational Harmonic Analysis* 21 (1) (2006) 31–52.
- [5] A. Bermanis, A. Averbuch, R. Coifman, Multiscale data sampling and function extension, *Applied and Computational Harmonic Analysis* <http://dx.doi.org/10.1016/j.acha.2012.03.002>.
- [6] B. Flannery, W. Press, S. Teukolsky, W. Vetterling, *Numerical recipes in C*, Cambridge University Press, 1992.
- [7] G. Stewart, J. Sun, *Matrix perturbation theory*, Vol. 175, Academic press New York, 1990.

- [8] A. Langville, C. Meyer, Updating markov chains with an eye on google's pagerank, *SIAM journal on matrix analysis and applications* 27 (4) (2006) 968–987.
- [9] A. Langville, C. Meyer, A survey of eigenvector methods for web information retrieval, *SIAM review* (2005) 135–161.
- [10] S. Kamvar, T. Haveliwala, G. Golub, Adaptive methods for the computation of pagerank, *Linear Algebra and its Applications* 386 (2004) 51–65.
- [11] C. Meyer, G. Stewart, Derivatives and perturbations of eigenvectors, *SIAM Journal on Numerical Analysis* (1988) 679–691.
- [12] P. Martinsson, V. Rokhlin, M. Tygert, A randomized algorithm for the decomposition of matrices, *Applied and Computational Harmonic Analysis* 30 (1) (2011) 47–68.
- [13] G. Golub, C. Van Loan, *Matrix computations*, Vol. 3, Johns Hopkins Univ Pr, 1996.
- [14] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, United States Governm. Pr. Office, 1950.
- [15] O. Kouropteva, O. Okun, M. Pietikäinen, Incremental locally linear embedding algorithm, *Image Analysis* (2005) 145–159.
- [16] M. Law, A. Jain, Incremental nonlinear dimensionality reduction by manifold learning, *Pattern Analysis and Machine Intelligence*, *IEEE Transactions on* 28 (3) (2006) 377–391.
- [17] G. David, A. Averbuch, Hierarchical data organization, clustering and denoising via localized diffusion folders, *Applied and Computational Harmonic Analysis* <http://dx.doi.org/10.1016/j.acha.2011.09.002>.