10.1504/IJCSE.2011.041221

Principal components analysis by the galaxy-based search algorithm: a novel metaheuristic for continuous optimisation

Hamed Shah-Hosseini

Faculty of Electrical and Computer Engineering,
Shahid Beheshti University, G.C.,
Tehran, Iran
E-mail: h_shahhosseini@sbu.ac.ir
E-mail: tasom2002@yahoo.com

Abstract: In this paper, the principal components analysis (PCA) is formulated as a continuous optimisation problem. Then, a novel metaheuristic inspired from nature is employed to explore the search space for the optimum solution to the PCA problem. The new metaheuristic is called 'galaxy-based search algorithm' or 'GbSA'. The GbSA imitates the spiral arm of spiral galaxies to search its surrounding. This spiral movement is enhanced by chaos to escape from local optimums. A local search algorithm is also utilised to adjust the solution obtained by the spiral movement of the GbSA. Experimental results demonstrate that the proposed GbSA for the PCA or GbSA-PCA is a promising tool for the PCA estimation.

Keywords: principal components analysis; PCA; metaheuristic; optimisation; chaos.

Reference to this paper should be made as follows: Shah-Hosseini, H. (2011) 'Principal components analysis by the galaxy-based search algorithm: a novel metaheuristic for continuous optimisation', *Int. J. Computational Science and Engineering*, Vol. 6, Nos. 1/2, pp.132–140.

Biographical notes: Hamed Shah-Hosseini received his BS in Computer Engineering from the University of Tehran. He also obtained his MS and PhD both in Computer Science (AI) from Amirkabir University of Technology, all with high honours. He is now with the Faculty of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran. His research interests include computational intelligence especially neural networks, evolutionary computation, swarm intelligence, and computer vision. He has introduced the time adaptive self-organizing map (TASOM) networks for both stationary and non-stationary environments. Moreover, he has introduced a bio-inspired swarm-based optimisation algorithm called 'Intelligent Water Drops' algorithm (or IWD algorithm), which is based on actions and reactions occurring in natural rivers. In addition, he presented a generalised Taylor's (G-Taylor's) theorem.

1 Introduction

Principal components analysis (PCA) is a well-known technique in signal and image processing for data reduction, feature extraction, preprocessing (Gonzalez and Woods, 2002), fault diagnosis (Zhao and Lu, 2007), watermarking (Ding et al., 2010), and blind source separation (Diamantaras and Papadimitriou, 2009). Generally, two approaches have been used for the PCA. One approach is the one that involves matrix calculations and it is reported in the textbook and research papers of numerical computation. Another approach involves computational intelligence mostly neural networks for the PCA estimation. Neural networks such as GHA (Sanger, 1989), APEX (Kung and Diamantaras, 1990), SOM (Lopez-Rubio et al., 2004), the TASOM (Shah-Hosseini and Safabakhsh, 2001), and the local PCA-SOM (Huang et al., 2008) are among those neural networks.

This research intends to pave the way for the PCA estimation using metaheuristics. As far as we know, metaheuristics (Holland, 1975; Kennedy and Eberhart, 1995; Dorigo and Stutzle, 2004; Shah-Hosseini, 2007) have not been employed so far for the PCA estimation. Here, a new metaheuristic is introduced to the community and the PCA is formulated as an optimisation problem. Then, the metaheuristic is employed to solve the PCA.

However, a combination of the PCA and metaheuristics has been introduced for problem solving. For example, a combination of the PCA, genetic algorithm (GA), and Tabu search has been used for molecular similarity (Xian et al., 2004). In addition, GA has been used for the selection of principal components (Barros and Rutledge, 1998). It is mentioned that based on a neural network approach, a non-linear PCA estimation using genetic programming has been reported (Hiden et al., 1999).

In this paper, the proposed metaheuristic called the 'galaxy-based search algorithm' or GbSA is introduced and modified for the PCA estimation. The proposed GbSA for the PCA is simply called 'GbSA-PCA'. The GbSA itself imitates an arm of spiral galaxies in the outer space. In other words, a spiral-like arm searches the solution space to find a better solution that the current one. Then, a local search algorithm is activated to fine-tune the obtained solution. This process continues until a termination condition(s) is satisfied. In other words, the PCA is formulated as an optimisation problem, and the GbSA tends to find the optimal solution. Here, an approximation error is defined and the GbSA-PCA seeks to minimise the error to find the principal vectors (eigenvectors).

The rest of the paper is organised as follows: next section includes a review on the PCA and its relation to the *eigenvalue problem*. Section 3 contains the proposed GbSA-PCA. Experimental results are expressed in Section 4. Final section, Section 5, contains the concluding remarks.

2 A review on PCA

In this section, first the main concepts of the PCA are reviewed. Then, some related methods used in the paper are mentioned in the subsequent sections. The following section expresses about the PCA and its relation to the eigenvalue problem.

2.1 The PCA and its relation to the eigenvalue problem

Dimensionality reduction is one of the main applications of the PCA. Specifically, an m-dimensional data vector $\mathbf{x}$ is represented by a p-dimensional vector $\hat{\mathbf{x}}$ such that $p < m$. This dimensionality reduction is implemented by a linear transformation, which is optimum in the mean-square sense. Consider that the m-dimensional random vector $\mathbf{X}$ represents the input environment from which input data vectors are originated. A sample of n realisations of the random vector $\mathbf{X}$ creates the sample (data) matrix $\mathbf{D}$ with m rows and n columns. In other words, the input data vectors form the columns of the m-by-n matrix $\mathbf{D}$. Before processing further, we are assured that the mean of the column vectors in $\mathbf{D}$ is zero. For this purpose, the sample mean of the input vectors is subtracted from each column vector of matrix $\mathbf{D}$. Thus, from now on, it is assumed that $\mathbf{D}$ has column vectors with zero mean.

The m-by-m correlation matrix $\mathbf{R}$ of the column vectors of $\mathbf{D}$ may be estimated by:

$$\mathbf{R} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_j \cdot \mathbf{x}_j^T \tag{1}$$

where $\mathbf{x}_j$ denotes the j^{th} column vector of $\mathbf{D}$ and the symbol T denotes the transpose. Let the eigenvalues of matrix $\mathbf{R}$ be sorted in descending order and be denoted by $\lambda_1, \lambda_2, \ldots, \lambda_m$ where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$. Also, let the corresponding eigenvectors be denoted by $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m$. Therefore, the eigenvalue problem requires that:

$$\mathbf{R}\mathbf{q}_j = \lambda_j \mathbf{q}_j \ , \quad j = 1, 2, \ldots, m \tag{2}$$

Since the correlation matrix $\mathbf{R}$ is symmetric, the eigenvalues λ_j are nonnegative and real values. If the eigenvalues are distinct, then the associated eigenvectors will be unique.

Let $\mathbf{x}$ be a realisation of the random vector $\mathbf{X}$ or simply assume that $\mathbf{x}$ is an input vector coming from the environment. The principal components a_j of the data vector $\mathbf{x}$ are defined by the following projections:

$$a_j = \mathbf{q}_j^T \mathbf{x} \ , \quad j = 1, 2, \ldots, m \tag{3}$$

where $\mathbf{q}_j$ are the unit eigenvectors of the correlation matrix $\mathbf{R}$. The vector $\mathbf{a} = [a_1, a_2, \ldots, a_m]^T$, which contains the principal components is viewed as the *analysis*. To reconstruct the original vector from $\mathbf{a}$, we may use the following formula:

$$\mathbf{x} = \sum_{j=1}^{m} a_j \mathbf{q}_j \tag{4}$$

The above formula may be viewed as the *synthesis*.

However, for dimensionality reduction, only the p dominant eigenvectors $\mathbf{q}_j$ corresponding to the p largest eigenvalues of the correlation matrix $\mathbf{R}$ are used for the reconstruction of the original data vector. Specifically, the data vector $\mathbf{x}$ is approximated by

$$\hat{\mathbf{x}} = \sum_{j=1}^{p} a_j \mathbf{q}_j \tag{5}$$

The error vector $\mathbf{e}$ for the approximation above is then:

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} \tag{6}$$

It is worth mentioning that the error vector $\mathbf{e}$ is orthogonal to the approximating vector $\hat{\mathbf{x}}$. Specifically:

$$\mathbf{e}^T \hat{\mathbf{x}} = 0 \tag{7}$$

The property above is known as the *principle of orthogonality*.

2.2 Rayleigh quotient

The Rayleigh quotient is used to compute eigenvalues from the eigenvectors. Assume that the eigenvector $\mathbf{q}_j$ of the matrix $\mathbf{R}$ is given or estimated and then the goal is to find its associated eigenvalue λ_j. The following quotient named *Rayleigh quotient* is used for this purpose:

$$\lambda_j = \frac{\mathbf{q}^T \mathbf{R} \mathbf{q}}{\mathbf{q}^T \mathbf{q}} \tag{8}$$

3 The proposed GbSA-PCA

The proposed GbSA for the PCA, which is called here 'GbSA-PCA' is composed of several components. The pseudo-code of the GbSA is depicted in Table 1. It is seen that the GbSA is composed of two main components: *SpiralChaoticMove* and the *LocalSearch*. The *SpiralChaoticMove* has the role of searching around the current solution denoted by *SG*. When the *SpiralChaoticMove* finds an improved solution better than the *GS*, it updates the *SG* with the improved solution, and the variable *Flag* is set to *true*. When *Flag* is *true*, the *LocalSearch* component of the GbSA is activated to search locally around the updated solution GS.

The pseudo-codes of the *SpiralChaoticMove* and the *LocalSearch* are shown in Tables 2 and 3, respectively. The *SpiralChaoticMove* is iterated maximally for *Max*Re *p* number of times. However, as shown in Table 2, whenever it finds a solution better than the current solution, the *SpiralChaoticMove* is terminated and the control of the algorithm is transferred to the GbSA. The *SpiralChaoticMove* searches the space around the current best solution using a spiral movement enhanced by a chaotic variable generated by *NextChaos*(). The chaotic sequence is generated by the logistic map:

$$x_{n+1} = \lambda x_n \left(1 - x_n\right) \quad n = 0, 1, 2, ... \tag{9}$$

Here, $\lambda = 4$ and $x_0 = 0.19$. It should be mentioned that the first 2,000 iterations of the logistic map are discarded because we do not want to include in the sequence the transient motion that leads to the chaotic attractor.

Table 1 The pseudo-code of the proposed GbSA

```
Procedure GbSA
SG ← GenerateInitialSolution
SG ← LocalSearch(SG)
While (termination condition is not met) do
    Flag ← False
    SpiralChaoticMove(SG, Flag)
    If (Flag) then
        SG ← LocalSearch(SG)
    Endif
Endwhile
Return SG
Endprocedure
```

Table 2 The pseudo-code of the *SpiralChaoticMove* used in the GbSA

```
Procedure SpiralChaoticMove
// input:
    S is the current best solution with L components such that $S_i$ denotes the ith component of solution S.
// output:
    SNext is the output, which is found first that is better than the given solution S.
    Flag is set to true to indicate that a better solution has been found.
// parameters:
    Each $\theta_i$ is initialised by (–1 + 2 NextChaos()).
    $\Delta\theta$ is a parameter. Here, 0.01.
    r is 0.001.
    $\Delta r$ is set by the value NextChaos() in each procedure call.
    MaxRe p is the maximum iteration that the SpiralChaoticMove searches. Here, 100.
$\theta \leftarrow -\pi$
While rep < MaxRe p
    Repeat for i = 1 to L
        $SNext_i \leftarrow S_i + NextChaos()\, r \cos(\theta_i)$
    Endrepeat
    If ($f(SNext) \geq f(S)$) then
        Flag ← true
        Goto Endprocedure;
    Endif
    Repeat for i = 1 to L
        $SNext_i \leftarrow S_i - NextChaos()\, r \cos(\theta_i)$
    Endrepeat
    If ($f(SNext) \geq f(S)$) then
        Flag ← true
        Goto Endprocedure;
    Endif
    $r \leftarrow r + \Delta r$
    Repeat for i = 1 to L
        $\theta_i \leftarrow \theta_i + \Delta\theta$
    Endrepeat
    Repeat i = 1 to L
    If($\theta_i > \pi$) then
        $\theta \leftarrow -\pi$
    Endif
    Endrepeat
    $rep \leftarrow rep + 1$
Endwhile
Endprocedure
```

Table 3 The pseudo-code of the local search used in the GbSA

```
Procedure LocalSearch
// input:
    $L$ is the number of components of candidate solutions.
    $S$ is the current solution with $L$ components such that $S_i$ denotes the component $i^{th}$ of solution $S$.
// output:
    $SNext$ is the output of the local search
// parameters:
    $\Delta S$ is the step size which is set by function $NextChaos()$.
    $\alpha$ is a dynamic parameter .
    $KMax$ denotes the maximum iteration that the local search has to search around a
    component to find a better solution. Here, 1,500.
    repeat for $i = 1$ to $L$
    $\alpha \leftarrow 1$
    $k \leftarrow 0$
    while $k < kMax$
        $SL_i \leftarrow S_i - \alpha \cdot \Delta S \cdot NextChaos()$
        $SU_i \leftarrow S_i + \alpha \cdot \Delta S \cdot NextChaos()$
        If $f(SL) < f(S)$ and $f(SU) < f(S)$ then
            Goto Endrepeat
        Endif
        If $f(SU) > f(S)$ then
            $S_i \leftarrow SU_i$
            $SL_i \leftarrow SU_i$
            $\alpha \leftarrow \alpha + 0.01 \times NextChaos()$
            $k \leftarrow k + 1$
        Else
        If $f(SL) > f(S)$ then
            $S_i \leftarrow SL_i$
            $SU_i \leftarrow SL_i$
            $\alpha \leftarrow \alpha + 0.01 \times NextChaos()$
            $k \leftarrow k + 1$
        Else
        $\alpha \leftarrow \alpha + 0.05 \times NextChaos()$
        $k \leftarrow k + 1$
        Endif
        Endwhile
    $SL_i \leftarrow S_i$
    $SR_i \leftarrow S_i$
Endrepeat
$SNext \leftarrow S$
Endprocedure
```

The *LocalSearch* component of the GbSA shown in Table 3 finds a locally-optimum solution or it exceeds the maximum number of iterations denoted by *kMax*. It is mentioned that a locally-optimum solution *S* is better than its two immediate neighbours *SU* and *SL*.

The *SpiralChaoticMove* in the GbSA is the mechanism which is used for exploring the search space to find the promising area where may include the optimal solution. In contrast, the *LocalSearch* is the component of the GbSA which is used for exploiting the promising area to find within this area the peak (valley) of the fitness (error) function. In summary, exploration is conducted by the *SpiralChaoticMove* while exploitation is guided by the *LocalSearch*. In the GAs, the exploration is implemented by crossover and the exploitation is executed by mutation. Both exploration and exploitation mechanisms are necessary for any metaheuristic to obtain a solution as close as possible to the optimal solution(s). For example, without the exploitation mechanism, the metaheuristic may not be able to obtain accurate solutions whereas without the exploration mechanism, the metaheuristic may get easily trapped into a local optimum.

The advantage of using the *LocalSeach* component used in this paper to some other exploitation mechanisms such as mutation is that the proposed local search never let the algorithm lose the best solution. Moreover, the *SpiralChaoticMove* also never loses the best solution found so far. In addition, it uses chaos to reduce the chance to get trapped into a local optimum. Due to the chaotic process, the GbSA does not return to the same solution and thus diversity of the found solutions is kept high. Keeping diversity high is especially important in dealing with multimodal problems in which there exist several local optimums in their search spaces.

The termination condition for the GbSA of Table 1 is implemented in two parts:

1 The GbSA terminates when the iteration count exceeds a prespecified maximum iteration. Here, 50,000.

2 The second part for termination checks the error difference between two successive iterations of the GbSA. If this difference is less than the value of 1e-10, then the parameter *inbreak* is increased by one. Moreover, Δr and ΔS are reset to the value of one. The GbSA exits when *inbreak* exceeds a predetermined value, here 100.

The solution that the GbSA-PCA seeks is the *p* dominant principal vectors of the data matrix **D**, which in fact are the *p* dominant eigenvectors of the correlation matrix **R**. The initial solution of the GbSA-PCA is composed of *p* vectors where each component of the vectors is drawn randomly from a uniform distribution in the interval [–1, 1]. This initialisation phase is implemented by the *GenerateInitialSolution* component of the GbSA in Table 1.

The error or the objective value that the GbSA-PCA intends to minimise is denoted by *f*(.) in Tables 1 to 3, which is defined in the following.

Consider the current solution $SG = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_p]$ contains the first *p* dominant eigenvectors produced by the GbSA-PCA. The estimated eigenvectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_p$ are normalised to have the unit norm. If $p = 1$, then the error of the solution $f(SG)$ denoted by *e* is calculated by

$$e = \sum_{i=1}^{n} \mathbf{e}_i^T \mathbf{e}_i \tag{10}$$

where $\mathbf{e}_i = \mathbf{x}_i - (\mathbf{q}_1^T \mathbf{x})\mathbf{q}_1$. For cases where $p > 1$, we use the following general formula for obtaining $f(SG)$:

$$e = \sum_{j=1}^{p} \sum_{i=1}^{n} \mathbf{e}_{i,j}^T \mathbf{e}_{i,j} \tag{11}$$

where $\mathbf{e}_{i,j} = \mathbf{x}_i - \sum_{k=1}^{j} (\mathbf{q}_k^T \mathbf{x})\mathbf{q}_k$.

In summary, the GbSA-PCA finds a solution $SG = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_p]$ which minimises the approximation error *e* defined in equation (11). The value of *p* is user-defined and it can be as great as the number of rows in the data matrix **D**.

After the GbSA terminates, the solution that it delivers is the approximate eigenvectors. As a result, we may use the *Rayleigh quotient* to obtain the eigenvalues from the estimated eigenvectors.

4 Experimental results

Two synthetic data matrixes are used to test the capability of the proposed GbSA-PCA in finding and estimating the dominant principal vectors. Moreover, two well-known datasets Iris and E.coli (UC Irvine Machine Learning Repository, 2011) are employed for testing the PCA-estimation capability of the GbSA-PCA. The Iris dataset contains 150 feature vectors where each vector contains four attributes. Therefore, the data matrix becomes of size 4-by-150. The E.coli dataset contains 336 feature vectors where each vector is composed of seven numerical attributes (the attribute 'name' is excluded). As a result, its relevant data matrix becomes of size 7-by-336. The GbSA-PCA is run for five times for each data matrix. It should be mentioned that all the experiments are conducted on a notebook with a Pentium 4 CPU running Microsoft Windows 7 operating system. The Microsoft Visual Studio 2005 is used to write the codes of the algorithms using the C# language.

For the first experiment, the following data matrix

$$\mathbf{D}_1 = \begin{bmatrix} 8 & -2 & 2 \\ -2 & 6 & -4 \\ 2 & -4 & 6 \end{bmatrix} \tag{12}$$

which contains three column data vectors, is used to test the proposed GbSA-PCA. The GbSA-PCA is run for five times. The eigenvalues and the errors are shown in Table 4. The correlation matrix of $\mathbf{D}_1$ contains two nonzero eigenvalues and one zero eigenvalue. We mention again that for correlation matrixes, eigenvalues are nonnegative and real numbers. As Table 4 shows, the first two eigenvalues are positive values and are in descending order of dominance. The biggest eigenvalue comes first, and then the second biggest eigenvalue appears. However, the GbSA-PCA finds the third eigenvalue such that its value is bigger than the second eigenvalue. Moreover, the range of eigenvalues obtained for the third eigenvalue is very wide. These two observations reflect this fact that the third eigenvalue must be zero. The estimated eigenvectors associated to the two biggest eigenvalues are shown in Table 5. Moreover, the error of the GbSA-PCA as shown in the last column of Table 4 is nearly zero. This error is the total approximation error defined in equation (11).

The second experiment uses the matrix

$$\mathbf{D}_2 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 2 & 0 & 3 \\ 5 & 1 & 11 \end{bmatrix} \tag{13}$$

which contains three 4-dimensional data vectors. The four eigenvalues and the approximation errors are shown in Table 6. This matrix has two nonzero eigenvalues and two zero eigenvalues. Table 6 shows that the first two eigenvalues are in descending order and they are the two dominant eigenvalues of $\mathbf{D}_2$. The first two dominant associated eigenvectors are shown in Table 7. The errors in the last column of Table 6 confirm that the GbSA-PCA performs well in approximating the eigenvalues and their associated eigenvectors.

The third experiment is performed on the Iris dataset. The four eigenvalues along with the approximation errors are shown in Table 8. For all the five runs, the errors are near to zero. Moreover, all the eigenvalues are in descending order. In summary, the GbSA-PCA performs quite well for the Iris dataset.

The last experiment with the GbSA-PCA is conducted for the E.coli dataset. The eigenvalues and the errors are reported in Table 9. In three out of the five runs of the GbSA-PCA, the eigenvalues are obtained in descending order. In runs three and four, however, two eigenvalues are not in the descending order. Their associated errors are quite higher than the rest of the runs of the GbSA-PCA revealing that the GbSA-PCA falls into local optimums in the two runs. Nevertheless, ignoring the order of eigenvalues, the values of the obtained eigenvalues are almost correct in the two mentioned runs. Therefore, the GbSA-PCA is capable of finding the eigenvalues and their associated eigenvectors, but the order of the eigenvalues may not be in the correct order if the GbSA-PCA gets trapped into a local optimum.

Table 4 The eigenvalues and the errors of the GbSA-PCA for the data matrix of equation (12)

Run	*EigenValue 1*	*EigenValue 2*	*EigenValue 3*	*Error*
1	129.543	27.7899	45.6304	1.49E-08
2	129.543	27.7899	55.9243	1.42E-08
3	129.543	27.7899	104.081	1.21E-06
4	129.543	27.7899	106.208	5.53E-07
5	129.543	27.79	124.911	4.41E-08
Average	129.5434	27.78992	87.35081	3.68E-07

Table 5 The first two dominant eigenvectors of the data matrix of equation (12)

	Eigenvector 1			*EigenVector 2*		
1	–0.4847	0.6506	–0.5846	–0.8532	–0.2047	0.47971
2	–0.4848	0.6506	–0.5846	–0.8532	–0.2047	0.47973
3	0.48475	–0.6506	0.58457	0.85327	0.20483	–0.4796
4	–0.4846	0.65066	–0.5847	–0.8534	–0.2046	0.47946
5	0.48437	–0.6507	0.5848	0.85363	0.20418	–0.4792
Average	–0.48464	0.65063	–0.58464	–0.85334	–0.20459	0.479531

Table 6 The eigenvalues and the errors of the GbSA-PCA for the data matrix of equation (13)

Run	*EigenValue 1*	*EigenValue 2*	*EigenValue 3*	*EigenValue 4*	*Error*
1	57.216018	1.4506942	47.662622	27.715023	3.46E-06
2	57.216023	1.4506437	5.3099672	40.241666	1.80E-08
3	57.216024	1.4506424	0.098681	10.811065	8.92E-10
4	57.216023	1.4506475	0.8145076	0.04833	1.67E-07
5	57.216022	1.4508919	13.5596	14.157156	9.27E-07
Average	57.21602	1.450704	13.48908	18.59465	9.15E-07

Table 7 The first two dominant eigenvectors of the Data matrix of equation (13) obtained by the proposed GbSA-PCA

	EigenVector 1				*EigenVector 2*			
1	0.08981	–0.17959	0.27650	0.93980	–0.37708	0.74869	–0.44836	0.31024
2	–0.08997	0.17998	–0.27658	–0.93969	–0.37488	0.74962	–0.44790	0.31133
3	0.08993	–0.17988	0.27656	0.93972	0.37519	–0.74967	0.44770	–0.31112
4	–0.08981	0.17986	–0.27654	–0.93974	0.37497	–0.74971	0.44796	–0.31091
5	0.09014	–0.17987	0.27652	0.93971	–0.37352	0.75225	–0.44639	0.30878
Average	0.0899	–0.1798	0.2765	0.9397	0.3751	–0.7500	0.4477	–0.3105

Table 8 The eigenvalues and the errors of the GbSA-PCA for the Iris dataset

Run	*EigenValue 1*	*EigenValue 2*	*EigenValue 3*	*EigenValue 4*	*Error*
1	629.50125	36.094143	11.700491	3.528929	1.86E-05
2	629.50125	36.094302	11.700067	3.528779	8.96E-07
3	629.50127	36.094319	11.700058	3.5287837	2.35E-06
4	629.50123	36.09429	11.700068	3.5332167	6.83E-05
5	629.50125	36.094291	11.700899	3.5295393	3.66E-05
Average	629.5012	36.09427	11.70032	3.52985	2.54E-05

Table 9 The eigenvalues and the errors of the GbSA-PCA for the E.coli dataset

Run	*EigenValue 1*	*EigenValue 2*	*EigenValue 3*	*EigenValue 4*	*EigenValue 5*	*EigenValue 6*	*EigenValue 7*	*Error*
1	30.038377	14.229841	4.6563648	4.5355224	2.8884514	1.6545324	0.2208473	0.000533
2	30.033374	14.2344	4.7621996	4.411665	2.9053236	1.6720871	0.2339738	0.003483
3	29.180866	15.045864	4.4573337	4.8994119	2.1665973	2.4781261	0.2297066	0.027593
4	29.887474	4.9213711	14.029333	4.8808329	2.7190809	1.9160736	0.2593248	0.052839
5	30.036456	14.23418	4.8145134	4.407388	2.8590766	1.6534558	0.2207977	0.000614
Average after sorting:	29.83531	14.35472	4.810772	4.538548	2.770012	1.812549	0.23293	0.017012

Figure 1 Average of five runs of the error versus iteration obtained by the GbSA-PCA

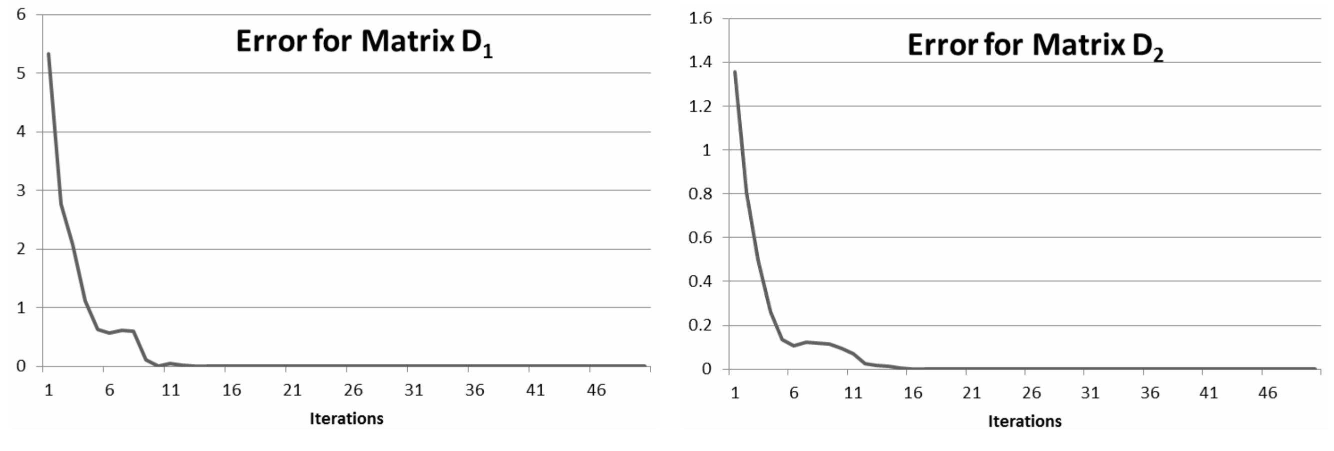
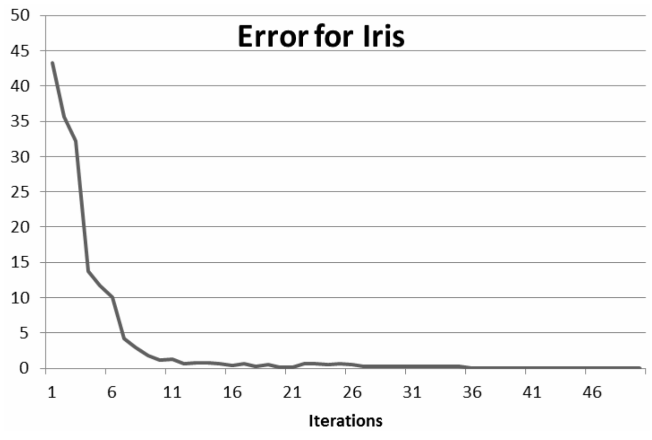
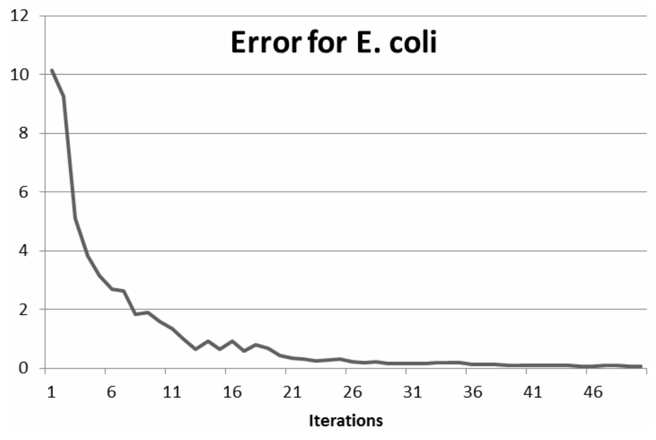


Figure 1 Average of five runs of the error versus iteration obtained by the GbSA-PCA (continued)





For the four experiments mentioned above, the error of the PCA estimation versus iteration averaged over five runs of the GbSA-PCA is shown in Figure 1. It is observed that the error of the GbSA-PCA drops fast at the first few iterations. Then, the error decreases gradually toward zero as the iteration number increases.

The running times of experiments performed by the GbSA-PCA are reported in Table 10. It is seen that as the number of data vectors increases the running time of the GbSA-PCA also increases. The reason is mostly related to the time to calculate the error in equation (11). In fact, the error calculation takes more time as the number of data vectors increases.

Table 10 The GbSA-PCA's running times (in seconds)

Data matrix or dataset	D_1	D_2	Iris	E.coli
Average in five runs	1.08	2.67	81.38	653.83

5 Conclusions

A new metaheuristic for continuous optimisation is introduced, which is used for PCA. The proposed metaheuristic is called the 'GbSA-PCA'. The GbSA itself mimics partly the spiral galaxies and how their arms fill the outer space. Here, the GbSA explores the search space of the PCA by spiral movement enhanced by a chaotic sequence. Experiments reveal that the GbSA-PCA is successful in approximating the PCA. However, it is possible to define a simpler error for the PCA to reduce the running time of the proposed algorithm. Moreover, the GbSA-PCA may be employed for some real applications where the PCA is involved in their relevant problem solving.

References

Barros, A.S. and Rutledge, D.N. (1998) 'Genetic algorithm applied to the selection of principal components', *Chemometrics and Intelligent Laboratory Systems*, Vol. 40, No. 1, pp.65–81.

Diamantaras, K.I. and Papadimitriou, T. (2009) 'Applying PCA neural models for the blind separation of signals', *Neurocomputing,* Vol. 73, Nos. 1–3, pp 3–9.

Ding, S., Zhang, P., Din, E., Yin, S., Naik, A., Deng, P. and Gui, W. (2010) 'On the application of PCA technique to fault diagnosis', *Tsinghua Science and Technology*, Vol. 15, No. 2, pp.138–144.

Dorigo, M. and Stutzle, T. (2004) *Ant Colony Optimization*, MIT Press, Cambridge, MA, USA.

Gonzalez, R.C. and Woods, R.E. (2002) *Digital Image Processing*, Prentice-Hall, New Delhi.

Hiden, H.G., Willis, M.J., Tham, M.T. and Montague, G.A. (1999) 'Non-linear principal components analysis using genetic programming', *Computers and Chemical Engineering*, Vol. 23, No. 3, pp.413–425.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Huang, D., Yi, Z. and Pu, X. (2008) 'A new local PCA-SOM algorithm', *Neurocomputing*, Vol. 71, Nos. 16–18, pp.3544–3552.

Kennedy, J. and Eberhart, R. (1995) 'Particle swarm optimization', in *Proc. of the IEEE Int. Conf. on Neural Networks*, Piscataway, NJ, pp.1942–1948.

Kung, S.Y. and Diamantaras, K.I. (1990) 'A neural network learning algorithm for adaptive principal component analysis (APEX)', *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp.861–864.

Lopez-Rubio, E., Munoz-Perez, J. and Gomez-Ruiz, J.A. (2004) 'A principal components analysis self-organizing map', *Neural Networks*, Vol. 17, No. 2, pp.261–270.

Sanger, T.D. (1989) 'Optimal unsupervised learning in a single-layer linear feedforward neural network', *Neural Networks*, Vol. 2, No. 6, pp.459–473.

Shah-Hosseini, H. (2007) 'Problem solving by intelligent water drops', *IEEE Congress on Evolutionary Computation*, Swissotel The Stamford, Singapore, pp.3226–3231.

Shah-Hosseini, H. and Safabakhsh, R. (2001) 'TAPCA: time adaptive self-organizing maps for adaptive principal components analysis', *Proc. IEEE Int. Conf. Image Processing*, pp.509–512.

UC Irvine Machine Learning Repository (2011) Available at http://archive.ics.uci.edu/ml.

Xian, B., Li, T., Sun, G. and Cao, T. (2004) 'A combination of principal component analysis, genetic algorithm, and Tabu search for 3d molecular similarity', *Journal of Molecular Structure*, Vol. 674, Nos. 1–3, pp.87–97.

Zhao, Q. and Lu, H. (2007) 'PCA-based web page watermarking', *Pattern Recognition*, Vol. 40, No. 4, pp.1334–1341.