Natural Computing
Setting the galaxy-based search algorithm into context

Kiyoon Kim and Lukas Schäfer

November 15, 2018

1 Introduction

Principal component analysis (PCA) is a mathematical procedure from the fields of statistics and linear algebra, originally introduced by Pearson in 1901 [Pearson, 1901] aiming to reduce a dataset of observations into a potentially lower-dimensional subspace while maintaining as much information as possible. This technique is frequently used as preprocessing of high-dimensional datasets or to reduce the dimensionality of such for human-interpretable visualisation. PCA has a vast variety of applications with astonishing results, e.g. it was successfully used to recreate a map of Europe based only on the genes of its population which correlated to their countries [Novembre et al., 2008] or to detect body features based on scans of human bodies [Freifeld and Black, 2012].

As nowadays, datasets of large size need to be dealt with, it can be of interest to approximate algorithms as PCA. For this purpose, Shah-Hosseini introduced the galaxy-based search algorithm (GbSA), a metaheuristic optimisation algorithm inspired by spiral movement of galaxies [Shah-Hosseini, 2011]. The paper defines PCA as a continuous minimisation problem in order to formulate an error function. The minimisation of this error leads to the the optimal solution for the principal component analysis. The search algorithm itself incorporates the exploration and exploitation idea frequently found in metaheuristic optimisation (MHO) algorithms. It uses circular movements from the currently best solution with random elements, referred to as "chaos", to find better solutions which are thereafter improved in a fine-grained local search.

Putting aside the unnecessarily dramatic motivation of "chaos enhanced imitation of spiral galaxies", it seems reasonable to aim for efficient approximations of a popular problem as PCA. However, principal component analysis in the end remains an eigenvalue problem for which highly optimised implementations already exist. The formulation of PCA as a minimisation problem to make use of optimisation techniques is interesting and seems promising, but it remains to be seen whether the algorithm can compete with modern implementations of this particular problem capable of finding the exact eigenvalues and -vectors in a fairly short amount of time.

2 The galaxy-based search algorithm

2.1 PCA as a minimisation problem

In order to use the algorithm of GbSA for PCA, the problem needs to be formulated as an optimisation problem. In PCA, we aim to reduce the dimensionality of a dataset. Let $\mathbf{X}$ be a $m \times n$ matrix, containing n m-dimensional sample vectors. First, we need to compute the mean of these samples

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1}$$

where x_i corresponds to the i-th column vector of $\mathbf{X}$. After subtracting this vector from each column of $\mathbf{X}$, each row of the matrix will have zero mean, i.e. its components will sum up to zero.

The m-by-m correlation matrix $\mathbf{R}$, also referred to as the scatter matrix, can be computed as follows

$$\mathbf{R} = \sum_{i=1}^{n} x_i \cdot x_i^T \tag{2}$$

where x_i again refers to the i-th column of $\mathbf{X}$ [1].

We compute $\mathbf{R}$ because we are interested in its eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$, sorted in descending order so that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_m$, and the corresponding unit eigenvectors $q_1, q_2, ..., q_m$. Due to the creation of $\mathbf{R}$, the matrix closely relates to the covariance matrix

[1] Note that this equation differs from the original paper of Shah-Hosseini. There is no reason to divide the sum by n as long as the mean is computed as shown in equation (1) which already includes the normalising factor $\frac{1}{n}$.

of $\mathbf{X}$. Its eigenvectors are all linearly-independent and orthogonal to each other, i.e. they construct a m-dimensional space, and most importantly point in the direction of the largest variance of the dataset $\mathbf{X}$ given the previous criteria. In order to transform $\mathbf{X}$ to p dimensions while maintaining maximal variance of the data, one would use the first p eigenvectors $q_1, q_2, ..., q_p$ to recreate each sample vector x_i of $\mathbf{X}$ as precisely as possible:

$$\hat{x}_{i,p} = \sum_{j=1}^{p} (q_j^T \cdot x_i) \cdot q_j \tag{3}$$

These estimates are a measure of quality for a PCA approximation used in the following error function

$$e = \sum_{j=1}^{p} \sum_{i=1}^{n} (x_i - \hat{x}_{i,j})^T \cdot (x_i - \hat{x}_{i,j}) \tag{4}$$

where each term resembles the reconstruction error of sample vector x_i using the p largest eigenvectors of $\mathbf{R}$. This error will be minimised by the exact eigenvectors. It should be noted that in general the exact vectors x_i can only be constructed by using all m eigenvectors. Therefore, it is not to be expected that the error e will be minimised at a value of 0 as it will presumably be impossible to reach a perfect reconstruction using less than m eigenvectors [2].

2.2 Algorithm outline

The galaxy-based search algorithm aims to find the eigenvectors $q_1, q_2, ..., q_m$ which minimise the error stated in equation (4). A solution of GbSA therefore is a set of m m-dimensional vectors as approximations of these eigenvectors of $\mathbf{R}$.

The algorithm first starts by initialising a solution where each component of the m vectors is randomly sampled from a uniform distribution over $[-1, 1]$. Besides, the so called chaotic sequence, which generates values in the interval $[0, 1]$ and is used as such seemingly random values, is initialised with $c_0 = 0.19$. The sequence is defined as follows

$$c_{n+1} = \lambda \; c_n \; (1 - c_n) \tag{5}$$

where the paper suggests $\lambda = 4$. At the beginning of the algorithm, it discards the first 2000 values. In the following, we will annotate the next chaos sequence value with c_{n+1}, i.e. a new value of the sequence is computed and used.

[2]Such a perfect reconstruction using $k < m$ eigenvectors will only be possible if there are linear dependencies among the m features of $\mathbf{X}$ so that they only span a space of at most dimension k.

Following this initialisation, a *LocalSearch* is executed aiming to improve on the randomly initialised solution. After that a circular search based on the current best solution called *SpiralChaoticMove* for exploration and the *LocalSearch* to improve upon new solutions found in the first search are used until a termination criterion is met. GbSA is terminated whenever either $50,000$ iterations are reached or when the last 100 iterations did not lead to any significant improvements of the error. Usually an error progression is seen as insignificant whenever it does not exceed 10^{-10}. Those three termination parameters can be varied.

2.2.1 LocalSearch

The *LocalSearch* of GbSA updates each eigenvector of the current solution by doing random steps. Therefore, two adjacent solutions SU_i and SL_i will be computed for each eigenvector of the current best solution S_i as follows

$$SL_i = S_i - \alpha \cdot \Delta S \cdot v \tag{6}$$

$$SU_i = S_i + \alpha \cdot \Delta S \cdot v \tag{7}$$

where α and ΔS are step-sizes and v [3] is a vector of the m next chaotic sequence values. α is set to 1 at the beginning of each *LocalSearch* execution while ΔS will be set to the next value of the chaos sequence after each call of *LocalSearch*. If neither SL nor SU lead to a smaller error than S then the *LocalSearch* is terminated early and it returns the current S. However, if one of those is of higher quality then the respective eigenvector S_i will be set to the one of the most promising solution and α will be increased by $0.01 \cdot c_{n+1}$. This will be repeated for each eigenvector for at most $kMax$ iterations, which is by default set to 1500.

2.2.2 SpiralChaoticMove

SpiralChoaticMove is a search in the promising area surrounding our current solution S using spiral movements. Therefore a value $\theta_i \in [-\pi, \pi]$ for each eigenvector is computed as follows:

$$\theta_i = (-1 + 2 \cdot c_{n+1}) \cdot \pi \text{ [4]} \tag{8}$$

In each iteration of this search, we compute a new eigenvector $SNext_i$ based on S_i of the current solu-

[3]In the original paper, a single chaotic sequence value is used instead of a m-dimensional vector. For more information regarding this change, see section 4.

[4]Note that the π does not appear in defintion of the original paper. The reasoning behind this change will be outlined in section 4.

tion so that

$$SNext_i = S_i + r \cdot \cos(\theta_i) \cdot v \tag{9}$$

where $v \in \mathbb{R}^m$ contains the next m values of the chaos sequence [5]. After repeating this for all m eigenvectors, $SNext$ is returned if it has a lower error than S, otherwise the same process is repeated again. If the second iteration does not improve S either, than r, which is initially set to 0.001 at the beginning of each *SpiralChaoticMove* call, is increased by Δr and each θ_i is updated by $\Delta\theta$. If a θ_i value exceeds π, then it will be set to $-\pi$ again to remain in its original range.

Δr is set to the next chaos sequence value after each *SpiralChaoticMove* execution, just as ΔS for the *LocalSearch* and $\Delta\theta$ is a constant parameter usually set to 0.01.

3 Presented evaluation results

Shah-Hosseini empirically evaluated the GbSA-PCA algorithm for four different datasets. The first two are a chosen symmetric matrix $D_1 \in \mathbb{Z}^{3\times3}$ and $D_2 \in \mathbb{Z}^{4\times3}$:

$$D_1 = \begin{bmatrix} 8 & -2 & 2 \\ -2 & 6 & -4 \\ 2 & -4 & 6 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 2 & 0 & 3 \\ 5 & 1 & 11 \end{bmatrix}$$

Besides those, an evaluation is also conducted for the Iris [Fisher, 1936] and Ecoli [Horton and Nakai, 1996] datasets which contain 150 4-dimensional and 336 7-dimensional sample vectors respectively. For each dataset, the GbSA algorithm is run five times. The paper states the obtained eigenvalues and errors for each such run of the datasets, as well as the dominant eigenvectors, error development over iterations and the average runtime. However, it should be criticised that no baseline performance is included in the evaluation. While the results seem to indicate that the algorithm is working as intended, the evaluation is hardly credible without any comparison.

The results also indicate that GbSA computes good approximations for the dominating eigenvalues and -vectors but there is considerable variance to be observed in the case of zero eigenvalues. This can be seen for the third eigenvalue of D_1 as well as the last two eigenvalues of D_2. Shah-Hosseini does not further elaborated why the algorithm includes such a variance in this case. However, it can be noticed that a zero eigenvalue implies linear dependency. Therefore, in these cases the exact data matrix $\mathbf{X}$ can be reconstructed using only the eigenvectors corresponding to non-zero eigenvalues of $\mathbf{R}$. So, all the remaining eigenvectors corresponding to the zero eigenvalues are irrelevant for the optimisation leading to arbitrary values for those vectors.

The reported runtimes of GbSA included in the evaluation imply a potential scalability issue given that the it took over 650s to terminate on the Ecoli dataset.

Additionally, the error values reported for each dataset are claimed to be approximation errors as defined in equation (4). First of all, the error is dependent on the choice of p which is not specified in the paper. Given that the table for each dataset includes the exact amount of eigenvalues of the scatter matrices, it could be assumed that the entire set of eigenvectors is used for recreation. However, each error, independent of p, will always include the term

$$\sum_{i=1}^{n} (x_i - \hat{x}_{i,1})^T \cdot (x_i - \hat{x}_{i,1})$$

of the sum where $p = 1$. This term for the first reported eigenvector $\begin{bmatrix} -0.4847 & 0.6506 & -0.5846 \end{bmatrix}^T$ of Run 1 for D_1 leads to a error of ≈ 41.58 which is much larger than the reported error of $1.49 \cdot 10^{-08}$. Given that each term is a squared value, the error can only rise for more considered p values. Therefore, we are uncertain which error function is reported in the tables of the GbSA-PCA evaluation, but it is surely not the error from equation (4) as stated in the paper.

4 Reproduction

For our implementation of the GbSA algorithm, we used the Python programming language (Python Software Foundation, `https://www.python.org`) with the NumPy [Oliphant, 2006] library for efficient linear algebra computation.

Generally, we adapted the algorithm as close to the outlined implementation of Shah-Hosseini as we could. However, we noticed some minor and a few major changes which were necessary for the algorithm to operate as we intended.

In the *LocalSearch*, it is unspecified how the solutions of SL and SU will be initialised. Given that

[5] Just as in *LocalSearch*, the original paper only used a single chaotic sequence value instead of a m-dimensional vector. In section 4 we outline our reasoning for this change.

the search is supposed to "exploit the promising area" around S, we assume that those solutions will be initialised to be S and then updated as specified in equation (7).

Furthermore we noticed that the solution components SL_i and SU_i were updated with a scalar value of the chaos sequence (weighted with step sizes). However, it seems to make more sense for us to update each eigenvector of the solutions SL and SU by using a vector of chaos sequence values rather than a scalar, so that each component of the eigenvector can be updated differently. This allows for more complex exploitation around S [6].

The same change also applies to the update of $SNext_i$ of the *SpiralChaoticMove* which originally only included a single chaos sequence value rather than a vector of m sequence values.

Besides, *SpiralChaoticMove* contains a minor uninitialised value with *rep* which is not mentioned before its usage in the while condition. We assume that the variable is just supposed to be a counter and we therefore initialise it to 0 at the beginning of the function.

The arguably biggest modification we made to the paper pseudo code is the initialisation of θ in *SpiralChaoticMove*. The initialisation we refer to in equation (8) is a modified version from the original paper where π is not included. Omitting the π leads to θ_i being in the interval $[-1, 1]$. However, θ_i is used in the $SNext_i$ computation where a cosine function is applied to it. Given that the update is supposed to explore the area around S in circular movements a cosine applied to a value in $[-\pi, \pi]$ seems more expedient and aligns properly with the condition at the end of the function where θ_i is set to $-\pi$ whenever the value exceeds π. This change seems essential for the algorithm to perform well, as without it, GbSA performed significantly less reliably. It could be that the first line of the pseudo code $\theta \leftarrow -\pi$ was intended to express a similar step but it conflicts with the initialisation mentioned above in the paper.

In our evaluation, we compare our paper-based GbSA algorithm to an implementation of PCA computing the exact eigenvectors of the scatter matrix and another baseline algorithm of the field of metaheuristic optimisation (MHO).

[6] It could also be that Shah-Hosseini refers to single values of the eigenvectors as components of the solutions. In this case the scalar update of a scalar component would make more sense, but would still lead to basically the same result without the efficiency benefits of our implementation which makes use of vectorisation techniques.

5 Proposed baseline

We implemented particle swarm optimisation (PSO) [Eberhart and Kennedy, 1995] as a comparable MHO algorithm for PCA approximation. For this algorithm, we used typical parameters without careful selection: $\alpha_1 = \alpha_2 = 2.1$, inertia $\omega = 0.7$, number of particles $N = 50$ and the maximal number of iterations $T = 10,000$. Each particle position was represented by a m^2-dimensional vector representing the m eigenvectors to be stacked on top of each other. The particle values were initially uniform randomly sampled from the interval $[-2.5, 2.5]$ and all velocities were sampled likewise from the interval $[-0,005, 0,005]$. The only unusual addition we introduced was an early termination criterion whenever the global best did not change for 20 iterations.

The particle positions and velocities are updated as usual, expressed by the following equations

$$v_i = \omega v_i + \alpha_1 r_1 \cdot (p_i - x_i) + \alpha_2 r_2 \cdot (g - x_i) \quad (10)$$

$$x_i = x_i + v_i \quad (11)$$

where v_i and x_i describe the velocity and position of the ith particle, r_1 and r_2 are uniformly random values drawn from $[0, 1]$ for each update, p_i resembles the personal best position of particle i and g represents the global best position found by any particle so far.

We have chosen PSO as the MHO baseline to GbSA because it has a similar structure. It also executes a local search to improve upon previously found solutions of good quality by moving towards and around the personal best position of a particle. At the same time, it moves particles towards their global best motivating exploitation and drive towards the global optima. This tradeoff of exploration in order to reach a global optimum and escape local bests and exploitation in order to improve upon previous performance is essential for most MHO algorithms. This consideration is executed similarly for GbSA and PSO.

One major difference between those two algorithms is the fact that PSO uses a single update step incorporating local and global search competing with each other while in GbSA those concepts have their separate search functions executed after each other. The arguably biggest difference between the two MHO algorithms is that PSO is population-based, i.e. it maintains a set of possible solutions which are updated and optimised in parallel while exchanging information using the global best position. In GbSA only the single best solution found so far is preserved and there is no knowledge about previous or different solutions present at the same time. This difference

allows us to evaluate whether a population-based approach might be suitable for PCA approximation.

6 Evaluation

We evaluated GbSA (as described in section 4) along with our PSO implementation and the exact PCA computation to see how the proposed algorithm performs compared to our baselines. Evaluation was conducted on the smaller D_1 and D_2 datasets as well as the Iris dataset [Fisher, 1936] from the original paper. This allows us to directly compare our evaluation results to the ones presented in the original paper and enables an assessment regarding the scalability of all methods as so that we can analyse whether the algorithms perform well on smaller as well as a larger dataset such as Iris.

All experiments were run on a Ubuntu 16.04 LTS server with a Intel(R) Core(TM) i7-6950X CPU, clocked at 3.00GHz, and 64GB of 2133MHz DDR4 memory using Python 3.5.2 and NumPy 1.15.4.

Similarly to the evaluation of Shah-Hosseini, we conducted five runs for GbSA and PSO for each of the datasets for a more representative result given the random components of both algorithms. The left side plots of Figure 1 show the progression of GbSA and PSO using the averaged error decay over all five runs of the algorithms over 100 iterations for all three datasets. It can immediately be seen that PSO converges considerably faster in all cases and shows more fluent progression while GbSA seems to make more sudden improvements. The right illustrations indicate the averaged error development of GbSA over 1000 iterations. Here it becomes clear that the algorithm takes many iterations without any visible error decay in order to converge.

This is also resembled in the running times of the algorithms which can be seen in table 1. While the durations for GbSA and PSO are averaged over five runs, we only executed the exact PCA computation once due to its deterministic nature.

Algorithm	D_1	D_2	Iris
GbSA	45.53	143.59	5121.58
PSO	0.72	4.90	164.24
Exact	0.00023	0.00012	0.00054

Table 1: (Averaged) running times (in seconds) for GbSA, PSO and the exact PCA computation for all three datasets.

The table clearly indicates that PSO scales considerable better for large datasets than GbSA. However,

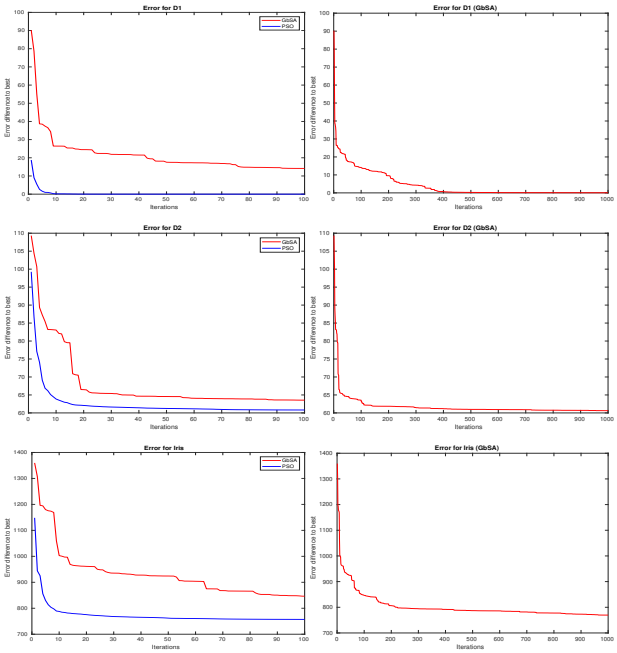

Figure 1: Error progression of both GbSA and PSO for 100 iterations (left), and GbSA only for 1000 iterations (right)

none of the MHO algorithms come close to the exact eigenvector computation executed in Numpy using geev LAPACK routines (LAPACK - Linear Algebra PACKage, `http://www.netlib.org/lapack`) to compute eigenvalues and eigenvectors of square matrices (the scatter matrix of which we compute eigenvectors is by definition always square). This exact calculation is about $10,000,000$ times faster than the GbSA algorithm for the Iris dataset. We already indicated this potential problem at the end of section 1 given that such routines are highly-optimised algorithms for these specific computations.

Table 2, found in the appendix, shows more detailed information regarding the performance of all algorithms on the D_1 dataset including running times, computed eigenvalues, errors (as defined in equation (4) using up to all eigenvectors $p = m$ for recreation) and the error difference of these algorithms to the exact PCA computation (denoted as "Error diff"). Each run was executed separately with newly initialised values and reset parameters ensuring comparable conditions.

All eigenvalues reported are computed by the Rayleigh quotient

$$\lambda_i = \frac{q_i^T \; R \; q_i}{q_i^T \; q_i} \tag{12}$$

using the eigenvectors found by the algorithms, where q_i is the eigenvector of the scatter matrix $\mathbf{R}$ corre-

sponding to λ_i.

While running times for PSO and especially GbSA were hardly impressive, it can be seen that all non-zero eigenvalues were found with high precision leading to very small error differences to the exact solution. PSO was able to find the exact eigenvalues in all five runs, while GbSA found those in four runs with minimal deviation in the second run.

Tables 3 and 4, which can be found in the appendix, show comparable information regarding the evaluation of D_2 and the Iris dataset. For D_2, we can see that PSO and GbSA were able to approximate the first eigenvalue with high precision but already for the second eigenvalue, considerable variance can be found leading to a larger error difference. GbSA performs more consistent on this dataset than the PSO baseline leading to a smaller average error difference to the exact computation error.

Looking at table 4 for the Iris dataset, we can clearly see that both metaheuristic optimisation algorithms perform poorly for more complex datasets. It is to be expected that with rising complexity and dimensionality of the input data the approximation of PCA becomes more challenging, but both algorithms were only able to consistently approximate the first eigenvalue. The second eigenvalue was sometimes approximated with reasonable precision, more often by PSO than GbSA, but they both failed to approximate the third and fourth eigenvalues. This leads to a high error difference and in the end a consistently better performance of PSO on this dataset.

7 Results

Generally, it can be stated that the performance of GbSA and PSO beyond simple input data just as D_1 and D_2 is lacking considerably. The exact eigenvalues are not reliably and consistently approximated well which makes these PCA approximations not very useful.

However, even with more precise approximations neither PSO nor in particular GbSA would be usable for real-world PCA approximation due to their inability to scale to large datasets. Even for Iris, GbSA took consistently longer than 1.5 hours. While Iris is the largest dataset considered in our evaluation, it should be kept in mind that it contains 150 samples of 4 dimensions. This is by no means a large dataset with respect to modern machine learning and data science standards. Nowadays, research and real-world applications deal with data of more than hundreds of thousands of samples with far higher dimensionality. Computing PCA on such a dataset would be impossible using any of the two MHO algorithms considered in this evaluation.

Both these algorithms try to approximate the exact PCA computation. As already indicated in section 1, the only use of such an estimation is to obtain estimations faster and more efficient than the exact calculation so that at least these can be computed where exact computation is infeasible. This is entirely missed when looking at the exact PCA computation which was not just faster on small datasets but seem to scale significantly better than GbSA and even PSO.

It is also disappointing that GbSA performs mostly worse than a simple PSO implementation given that the GbSA algorithm is considerably more complicated and we used the exact parameters proposed in the paper, already used for these datasets and PCA application. On the other hand, we simply used typical parameters for PSO without any parameter optimisation.

It seems, that in our evaluation one of the main problems of GbSA is its scalability. However, even with improved scaling the algorithm seems to make less stable and slower progress than the simple PSO algorithm disregarding the more computational effort in each GbSA iteration, looking at figure 1.

Lastly, we should address the extensive discrepancy of GbSA performance in our evaluation and the original evaluation of Shah-Hosseini. While we already criticised the absence of any baseline for comparison in the original paper, it is apparent that his stated runtime values vary largely with our findings. Also, the eigenvalues of Iris were approximated with higher consistency and especially the error development reaches a very low error after only $10, 15$ and 27 iterations for D_1, D_2 and Iris. As already elaborated in section 3, we are still uncertain which error is actually reported by the author.

8 Conclusions

We presented the galaxy-based search algorithm for PCA approximation and evaluated the MHO algorithm in comparison to our proposed PSO baseline as well as an exact solution to PCA. Our evaluation results confirmed that both metaheuristic approximations scale considerably worse than the exact computation, which makes them hardly useful. Additionally, we found a considerable discrepancy in our evaluation to the claimed results of the original paper and proposed corrections of the algorithm. Future work should try to improve the scalability of such MHO algorithms to make for feasible approximations.

References

[Eberhart and Kennedy, 1995] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.

[Fisher, 1936] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

[Freifeld and Black, 2012] Freifeld, O. and Black, M. J. (2012). Lie bodies: A manifold representation of 3d human shape. In *European Conference on Computer Vision*, pages 1–14. Springer.

[Horton and Nakai, 1996] Horton, P. and Nakai, K. (1996). A probabilistic classification system for predicting the cellular localization sites of proteins. In *Ismb*, volume 4, pages 109–115.

[Novembre et al., 2008] Novembre, J., Johnson, T., Bryc, K., Kutalik, Z., Boyko, A. R., Auton, A., Indap, A., King, K. S., Bergmann, S., Nelson, M. R., et al. (2008). Genes mirror geography within europe. *Nature*, 456(7218):98.

[Oliphant, 2006] Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Shah-Hosseini, 2011] Shah-Hosseini, H. (2011). Principal components analysis by the galaxy-based search algorithm: a novel metaheuristic for continuous optimisation. *Int. J. Comput. Sci. Eng.*, 6(1/2):132–140.

A Appendix - Evaluation tables

Algorithm	Run	Runtime (s)	Eigenvalue1	Eigenvalue2	Eigenvalue3	Error	Error diff
Exact	/	0.00023	129.54342	27.78991	0.00000	185.12324	/
GbSA	1	18.36497	129.54342	27.78991	5.87655	185.12325	3.42509E-06
GbSA	2	97.39782	129.54341	27.78993	21.34846	185.12327	2.26916E-05
GbSA	3	38.06691	129.54342	27.78991	28.79637	185.12324	6.18133E-08
GbSA	4	41.54043	129.54342	27.78991	7.81696	185.12324	3.39577E-09
GbSA	5	32.28971	129.54342	27.78991	7.14390	185.12324	2.73158E-08
PSO	1	0.62021	129.54342	27.78991	8.31267	185.12324	-2.84217E-14
PSO	2	0.81320	129.54342	27.78991	11.36033	185.12324	0.00000E+00
PSO	3	0.61629	129.54342	27.78991	5.35483	185.12324	0.00000E+00
PSO	4	0.75415	129.54342	27.78991	4.67150	185.12324	-2.84217E-14
PSO	5	0.79515	129.54342	27.78991	48.89745	185.12324	-2.84217E-14

Table 2: D_1 evaluation results for GbSA, PSO and the exact PCA calculation.

Algorithm	Run	Runtime (s)	Eigenvalue1	Eigenvalue2	Eigenvalue3	Eigenvalue4	Error	Error diff
Exact	/	0.00012	57.21602	1.45064	0.00000	0.00000	60.11731	/
GbSA	1	119.02664	57.21207	0.83193	0.59292	3.36334	60.75723	0.63992
GbSA	2	177.31323	57.21564	1.38202	0.22476	27.68202	60.23938	0.12207
GbSA	3	145.11669	57.21602	1.45020	0.00005	8.61863	60.11831	0.00100
GbSA	4	132.02770	57.21601	1.44394	0.02134	22.64916	60.12906	0.01175
GbSA	5	144.48816	57.21409	1.13954	0.26568	2.66958	60.43630	0.31899
PSO	1	5.09605	57.20504	1.40575	0.12292	3.99875	60.35720	0.23989
PSO	2	4.35881	57.21573	0.53057	1.41868	0.43100	61.36535	1.24804
PSO	3	3.88815	57.21332	1.33794	0.24121	3.12530	60.26181	0.14450
PSO	4	5.00990	57.13563	1.61556	0.06261	4.52381	60.41008	0.29277
PSO	5	6.12782	57.14846	1.27326	0.51551	2.08029	60.80482	0.68751

Table 3: D_2 evaluation results for GbSA, PSO and the exact PCA calculation.

Algorithm	Run	Runtime (s)	Eigenvalue1	Eigenvalue2	Eigenvalue3	Eigenvalue4	Error	Error diff
Exact	/	0.00054	629.50127	36.09429	11.70006	3.52877	750.90513	/
GbSA	1	4483.88745	629.50038	14.39912	35.21438	90.21195	773.86505	22.95992
GbSA	2	5956.49259	629.47252	16.51689	34.07919	432.04850	774.64263	23.73750
GbSA	3	4873.65144	627.91609	36.90890	13.59590	190.25128	756.34385	5.43872
GbSA	4	5206.28345	629.44995	33.19018	10.57089	34.08156	758.26460	7.35947
GbSA	5	5087.56060	629.26200	34.30459	9.10701	362.38499	759.42687	8.52174
PSO	1	118.10489	629.07437	21.91155	26.35250	37.97740	766.67734	15.77221
PSO	2	176.15215	629.38158	35.87833	4.37164	211.23968	759.27522	8.37009
PSO	3	214.86395	628.48292	34.55929	12.47700	22.59804	756.01556	5.11043
PSO	4	162.05806	629.32511	36.34220	7.91614	56.90591	756.38737	5.48224
PSO	5	150.02870	628.93795	31.70392	19.27262	564.46425	759.14880	8.24367

Table 4: Iris evaluation results for GbSA, PSO and the exact PCA calculation.