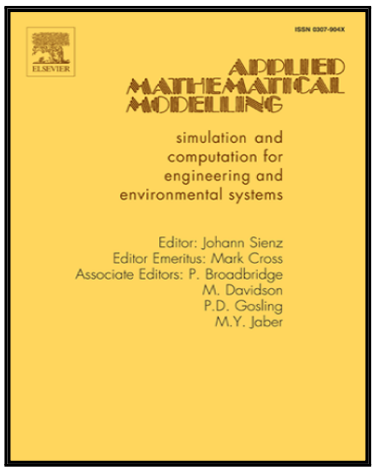

Numerical determination for solving the symmetric eigenvector problem using genetic algorithm

F.J. Navarro-González , P. Compañ , R. Satorre , Y. Villacampa

PII: S0307-904X(15)00824-0
DOI: 10.1016/j.apm.2015.12.015
Reference: APM 10932

To appear in: *Applied Mathematical Modelling*

Received date: 5 January 2015
Revised date: 11 November 2015
Accepted date: 4 December 2015

Please cite this article as: F.J. Navarro-González , P. Compañ , R. Satorre , Y. Villacampa , Numerical determination for solving the symmetric eigenvector problem using genetic algorithm, *Applied Mathematical Modelling* (2015), doi: 10.1016/j.apm.2015.12.015

Highlights

The methodology obtains the eigenvectors of a symmetric or Hermitian matrix using a genetic algorithm.

Also the target vector space can be either real or complex.

The method searches the eigenvectors and calculates the eigenvalues afterwards.

The method obtains precise results in the real and complex cases.

Numerical determination for solving the symmetric eigenvector problem using genetic algorithm

Navarro-González, F.J[1].; Compañ, P.[2]; Satorre, R.[2]; Villacampa, Y.[1]

[1]*Departamento de Matemática Aplicada, Universidad de Alicante, Alicante. Spain*

[2]*Departamento de Ciencia de la Computación e Inteligencia Artificial. Universidad de Alicante, Alicante. Spain*

ABSTRACT

The eigenvalues and eigenvectors of a matrix have many applications in engineering and science. For example they are important in studying and solving structural problems, in the treatment of signal or image processing, in the study of quantum mechanics and in certain physical problems. It is therefore essential to analyze methodologies to obtain the eigenvectors and eigenvalues of symmetric and Hermitian matrices.In this paper the authors present a methodology for obtaining the eigenvectors and eigenvalues of a symmetric or hermitian matrix using a genetic algorithm. Unlike other methodologies, the process is centred in searching the eigenvectors and calculating the eigenvalues afterwards. In the search of the eigenvectors a genetic-based algorithm is used. Genetic algorithms are indicated when the search space is extended, unknown or with an intricate geometry. Also, the target vector space can be either real or complex, allowing in this way a wider field of application for the proposed method. The algorithm is tested comparing the results with those obtained by other methods or with the values previously known. So, seven applications are included: a real symmetric matrix corresponding to a vibrating system, a complex hermitian matrix and an important application of the diagonalization problem (Coope matrix) corresponding to quantum mechanics examples, a physical problem in which data are analysed to reduce the number of variables, a comparison with the power method and the studies of a degenerate and an ill-conditioned matrix.

Keywords: eigenvector, eigenvalue, genetic algorithm, symmetric matrices, Hermitian matrices, computational methods

Corresponding author: villacampa@ua.es;francisco.navarro@ua.es

1. Introduction.

The problem of determining the set of eigenvectors and eigenvalues of a matrix is known as the complete eigenvalue problem. This matter is presented in several problems, as solving differential equations, studying the stability and behaviour of mechanical structures and determining the allowed states of quantum systems.

Given a matrix a, A, representing a linear transformation on a vector space $V(K)$ defined over a field K, the problem results in the determination of the vectors, eigenvectors *u*, and scalars, eigenvalues *ʎ*, that solve the equation:

$$A \cdot u = \lambda u \tag{1.1}$$

A classic approach is to calculate the eigenvalues as the solution of:

$$\det(A - \lambda \cdot I) = 0 \tag{1.2}$$

That is, solving a polynomial equation (characteristic polynomial) $P(\lambda) = 0$. Once the eigenvalues are known, the eigenvector corresponding to each eigenvalue λ_i is calculated solving the linear system:

$$(A - \lambda_i \cdot I) \cdot u^{(i)} = 0 \tag{1.3}$$

Leverrier and Krylov methods can be used to solve the characteristic polynomial (Stoer, 2002,[1]; Endre, 2003, [2]; Demidovich, 1993, [3]; Volkov, 1987, [4]). However, the calculation of the numeric solution has instability problems because of the dependence of the zeroes estimation from the polynomial coefficients. Moreover, there are numerical methods for diagonalizing a symmetric matrix (Parlettt,1981, [5]; Wilkinson,1965 [6]; Cullun, 1985 [7]). Sometimes not all the eigenvalues are calculated, limiting the efforts to some of them.

The eigenvalue problem can also be converted in an optimization problem, where an adequate function can be minimized or maximized computing its gradient and hessian, and obtaining the eigenvectors of the matrix. In this case the non-deterministic and stochastic methods can be of interest. These methods, that have been applied in physics, economy and other fields, can be found in several papers (Kirkpatrick, 1983 [8]; Van Laarhoven, 1987 [9]; Goldberg, 1989 [10]).

Some authors (Subhajit, 2011 [11]) have used genetic algorithms to calculate some of the eigenvalues by means of the Rayleigh quotient.

Other methods that don't need to solve the characteristic polynomial are the power method and its variants, (Demidovich, 1993 [3]; Volkov, 1987 [4]). Let the eigenvalues $\lambda_1, \lambda_2, .., \lambda_n$ ordered by their magnitudes,

$$|\lambda_1| \geq |\lambda_2| \geq .. \geq |\lambda_n| \tag{1.4}$$

When considering the repeated action of the matrix over an arbitrary vector (expressed using the eigenvectors basis)

$$A \cdot y = A \cdot \left(y^1 \cdot u_1 + y^2 \cdot u_2 + .. + y^n \cdot u_n \right) = y^1 \cdot \lambda \cdot u_1 + y^2 \cdot \lambda_2 \cdot u_2 + .. + y^n \cdot \lambda_n \cdot u_n \tag{1.5}$$

$$A^N \cdot y = \lambda_1^N \cdot \left[y^1 \cdot u_1 + y^2 \cdot \left(\frac{\lambda_2}{\lambda_1} \right)^N \cdot u_2 + .. + y^n \cdot \left(\frac{\lambda_n}{\lambda_1} \right)^N \cdot u_n \right] \quad (1.6)$$

The convergence to the first eigenvector depends on the relation (λ_2/λ_1). There are variations of the method to accelerate the convergence velocity: the shifted power method, Aitken's method, (Demidovich, 1993 [3]; Volkov, 1987 [4]), or the Jacobi method and others if the matrix is symmetric. When the first eigenvector is calculated, a process of deflation removes it from the search space.

Other possibility that can be used is the QR method that makes orthogonal transforms according to the spectral theorem for symmetric matrices (Golub, G.H,1996 [12]; Endre Süli,2003 [13]). So, the QR decomposition method factors any matrix A as A=QR, where Q is an orthogonal matrix and R is a non-singular upper matrix. This is the base of the QR eigenvalue algorithm used for symmetric and Hermitian matrices. A variation of this method, known as Shifted QR makes a previous modification of the matrix A, $A - \alpha I = QR$ to speed up the convergence of the algorithm.

2. Numeric methodology

In the present paper an implementation of a genetic algorithm is presented for calculating the first eigenvector and eigenvalue of a symmetric or hermitian matrix. To determine the rest of eigenvectors, the matrix is modified by a rotation that transforms the first vector of the canonical basis into the calculated eigenvector. The eigenvectors of the transformed matrix have the first component equal to zero.

Once all the eigenvectors have been calculated, the change of basis can be inverted, obtaining the components of the eigenvectors in the canonical basis.

2.1. *Search of the first eigenvector.*

Given a matrix A, obtaining the eigenvectors is equivalent to solve the equation:

$$A \cdot u_i = \lambda_i \cdot u_i \quad (2.1)$$

For simplicity, it can be assumed the normalized eigenvectors, $\|u_i\| = 1$. Vectors u_i and its images $z = A \cdot u_i$ are collinear, and so that it meets:

$$u_i^T \cdot z = u_i^T \cdot A \cdot u_i = \lambda_i \cdot u_i^T \cdot u_i = \lambda_i \cdot \|u_i\|^2 = \lambda_i \quad (2.2)$$

Given any vector of module 1 close to the eigenvector $x = u_i + e$, the image can be decomposed into two parts one collinear and one perpendicular:

$$z = A \cdot x = A \cdot (u_i + e) = \lambda_i \cdot u_i + A \cdot e = \lambda_i \cdot [x - e] + A \cdot e = \lambda_i \cdot x + [A - \lambda_i \cdot I] \cdot e =$$
$$= \lambda_i \cdot x + \varepsilon \cdot x + z_\perp = (\lambda_i + \varepsilon) \cdot x + z_\perp$$

So, if the vectors are normalized, $\|u\| = 1$ any unitary vector close to the eigenvector, $x = u_i + e$ has an image which has parallel and orthogonal components:

$$z = A \cdot x = (\lambda_i + \varepsilon) \cdot x + z_\perp \tag{2.3}$$

One way to measure the quality of x as eigenvector is considered the perpendicular component module:

$$\frac{\|z_\perp\|^2}{\|z\|^2} = \frac{1}{\|z\|^2} \cdot \left\| z - \frac{(z^T \cdot x)}{\|x\|} \cdot \frac{x}{\|x\|} \right\|^2 = \frac{\left\| z - (z^T \cdot x) \cdot x \right\|^2}{\|z\|^2} =$$
$$= 1 + \frac{(z^T \cdot x)^2}{\|z\|^2} - 2 \cdot \frac{(z^T \cdot x) \cdot (z^T \cdot x)}{\|z\|^2} = 1 - \frac{(z^T \cdot x)^2}{\|z\|^2} = 1 - \frac{\|z\|^2 \cdot \|x\|^2 \cdot \cos^2 \alpha}{\|z\|^2} =$$
$$= 1 - \cos^2 \alpha = \sin^2 \alpha \tag{2.4}$$

, where α is the angle between the vectors x and z.

Following these arguments, the next function can measure the proximity to the eigenvectors:

$$\Phi_I(x) = \frac{\left\| A \cdot x - (x^T \cdot A \cdot x) \cdot x \right\|}{\|A \cdot x\|} = \sin(\alpha) \tag{2.5}$$

In the above figure 1, the function (2.5) is represented for a bi-dimensional symmetric matrix. Also, the vectors $z_\perp$ can be seen:

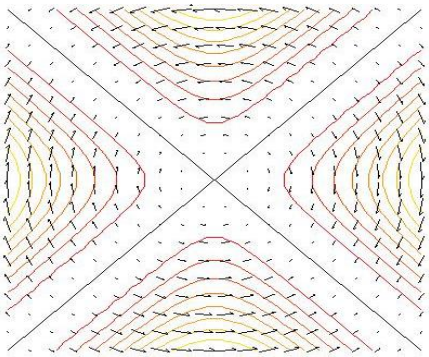

Figure 1- Perpendicular error and level curves of error function $\Phi_I(x)$

But this is not the only possibility, because other functions can also be considered:

$$\Phi_{II}(x)=\left\|\frac{z}{\|z\|}-\frac{x}{\|x\|}\right\|^2=\left\|\frac{z}{\|z\|}-x\right\|^2=\left\|\frac{z}{\|z\|}\right\|^2+\|x\|^2-2\cdot\frac{\left(z^T\cdot x\right)}{\|z\|}=2-2\cdot\frac{\left(z^T\cdot x\right)}{\|z\|}= \tag{2.6}$$
$$=2-2\cdot\cos\alpha=2\cdot\left[1-\cos\alpha\right]$$

The behaviour of these functions is different when the angle is near zero:

$$\Phi_I(x)=\sin\alpha\approx\alpha+o(3) \tag{2.7}$$

$$\Phi_{II}(x)=2\cdot\left[1-\cos\alpha\right]\approx\alpha^2+o(4) \tag{2.8}$$

The functions (2.7) and (2.8) can be used as fitness functions in the genetic algorithm. Once the eigenvector u_i is determined, the corresponding eigenvalue can be obtained considering that:

$$\left[A\cdot u_i\right]^k=\left[\lambda_i+a_k\right]\cdot u_i^k\rightarrow\frac{\left[A\cdot u_i\right]^k}{u_i^k}=\lambda_i+a_k\rightarrow\frac{1}{n}\cdot\sum_{k=1}^{n}\frac{\left[A\cdot u_i\right]^k}{u_i^k}=\lambda_i+\frac{1}{n}\cdot\sum_{k=1}^{n}a_k\approx\lambda_i \tag{2.9}$$

,because the components of the error a_k are usually small and distributed around zero.

However, problems could appear when vectors with $u_i^k<<1$ cause an over-consideration of the particular quotient, sometimes very far from the searched eigenvalue. To avoid this, a weighted average (using the squares of the components u_i as weights), can be considered:

$$\frac{1}{\sum_{k=1}^{n}\left(u_i^k\right)^2}\cdot\sum_{k=1}^{n}\frac{\left[A\cdot u_i\right]^k}{u_i^k}\cdot\left(u_i^k\right)^2=\frac{1}{1}\cdot\sum_{k=1}^{n}\left[A\cdot u_i\right]^k\cdot u_i^k=u_i^T\cdot A\cdot u_i=\lambda_i+\sum_{k=1}^{n}a_k\approx\lambda_i \tag{2.10}$$

,because the vectors are unitary.

2.2. ***Dimensional reduction of the problem. The next eigenvectors.***

Initially, the case of a matrix with real eigenvectors is studied. After, the complex case is considered.

2.2.1. *Dimensional reduction for real matrices.*

When the first eigenvector u_1 has been obtained, the vector $e_1=\begin{pmatrix}1\\0_{nx1}\end{pmatrix}$ of the canonical basis is rotated taking it to the direction of the vector u_1.

Regardless of indices, the eigenvector *u* has the components: $u = \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix}$.

where υ_1 is a scalar and υ_R a vector of dimension *d-1*. The symmetric matrix can be decomposed in similar blocks

$$A = \begin{pmatrix} m & p^T \\ p & M \end{pmatrix} \Rightarrow \begin{pmatrix} m & p^T \\ p & M \end{pmatrix} \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix} = \lambda \cdot \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix} \tag{2.11}$$

It is necessary to rotate in the plane defined by the vectors $e_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $u = \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix}$.

Denoting by $\otimes$ the Kronecker product, the rotation matrix has the form:

$$Q = \begin{pmatrix} \upsilon_1 & -\upsilon_R^T \\ \upsilon_R & I - \dfrac{1}{(1+\upsilon_1)} \cdot \upsilon_R \otimes \upsilon_R^T \end{pmatrix} \tag{2.12}$$

The matrix in this new basis is, *A'*:

$$A' = Q^T \cdot A \cdot Q \tag{2.13}$$

The next eigenvectors have a zero in the first component, that is, the dimension of the search space is n-1. When the eigenvectors are found, the change of basis must be inverted:

When further eigenvector $u_2|_{B_1} = \begin{pmatrix} 0 \\ \upsilon_R^2 \end{pmatrix}_{B_1}$ has been obtained, it is necessary to transform

it to put it in the original base, B_0. For this:

$$u_2|_{B_0} = Q_1 \cdot \begin{pmatrix} 0 \\ \upsilon_R^{(2)} \end{pmatrix}_{B_1} \tag{2.14}$$

,where Q_1 is the matrix corresponding to the rotation.

The expression of the matrix A in the new basis A' has the form:

$$A|_{B_1} = Q_1^T \cdot A|_{B_0} \cdot Q_1 = \begin{pmatrix} \lambda_1 & 0 \\ 0 & A_{red} \end{pmatrix} \tag{2.15}$$

In the same way it is obtained for the k-th eigenvector:

$$u_k|_{B_0} = Q_1 \cdots Q_{k-3} \cdot Q_{k-2} \cdot u_k|_{B_{k-1}} \tag{2.16}$$

2.2.2. *Dimensional reduction for complex matrices.*

In the case of a hermitian matrix the only difference are the vectors that take part of the rotation.

Now, the rotation takes the vector e_1 to the vector $u_1 = \begin{pmatrix} |\upsilon_1| \\ \frac{\upsilon_1^{\dagger} \cdot \upsilon_R}{|\upsilon_1|} \end{pmatrix}$, where $\begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix}$ is the first obtained eigenvector, and $u^{\dagger}$ is the complex conjugate of the vector u.

The complex eigenvector must satisfy the equation:

$$A = \begin{pmatrix} m & p^{\dagger} \\ p & M \end{pmatrix} \Rightarrow \begin{pmatrix} m & p^{\dagger} \\ p & M \end{pmatrix} \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix} = \lambda \cdot \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix} \tag{2.17}$$

The norm of the vector is:

$$\|u\|^2 = u^{\dagger} \cdot u = 1 \tag{2.18}$$

Imposing the additional condition that the first component is real;

$$w = \frac{1}{\upsilon_1} \cdot \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix} \rightarrow \|w\|^2 = \frac{1}{|\upsilon_1|^2} \cdot \|u\|^2 = \frac{1}{|\upsilon_1|^2} \tag{2.19}$$

This normalized vector has the form:

$$v = \frac{w}{\|w\|} = \frac{|\upsilon_1|}{\upsilon_1} \cdot \begin{pmatrix} \upsilon_1 \\ \upsilon_R \end{pmatrix} = \begin{pmatrix} |\upsilon_1| \\ \frac{|\upsilon_1| \cdot \upsilon_R}{\upsilon_1} \end{pmatrix} = \begin{pmatrix} |\upsilon_1| \\ \frac{\upsilon_1^{\dagger} \cdot \upsilon_R}{|\upsilon_1|} \end{pmatrix} \tag{2.20}$$

As in the real case (2.12), the matrix Q that changes between the old and the new basis can be calculated:

$$Q = \begin{pmatrix} |\upsilon_1| & -\frac{\upsilon_1 \cdot \upsilon_R^{\dagger}}{|\upsilon_1|} \\ \frac{\upsilon_1^{\dagger} \cdot \upsilon_R}{|\upsilon_1|} & I - \frac{1}{(1+|\upsilon_1|)} \cdot \frac{\upsilon_1^{\dagger} \cdot \upsilon_R}{|\upsilon_1|} \otimes \frac{\upsilon_1 \cdot \upsilon_R^{\dagger}}{|\upsilon_1|} \end{pmatrix} \tag{2.21}$$

,and the matrix in the new basis:

$$A' = Q^T \cdot A \cdot Q \tag{2.22}$$

3. Genetic algorithms

There are a lot of problems that can be solved searching the solution in a space of states. Several strategies try to generate new states and test them: uninformed search, greedy search, A*-search, gradient descent algorithms, simulated annealing, genetic algorithms, (Russell, 2004 [14]).

There are problems in which the solution consists not only in an element of the search space the intermediate steps are also of interest (robot navigation problem or a sliding tile puzzle). On the other side, some problems need only the solution element to be considered as solved (natural language processing, electronic circuit design and the n-queens problem are examples of this kind of problems). In the second case, local searching algorithms (genetic algorithms are into this kind) are of interest.

Genetic algorithms (GA) are non-deterministic optimization methods (Holland, 1975 [15]). These methods are based on the theory of evolution by natural selection, where biological diversity and species adaptation to natural environments are originated by biological processes as reproduction (with genetic information interchange), mutations and individual selection. Genetic algorithms (first used in 70's) try to mimic these natural processes, creating an initial population of individuals with characteristics that determine its fitness to the problem solution. The evolutive process consists in the creation of new individuals using the characteristics of the best adapted of each generation.

The existence of a fitness function allows using GA as an alternative way of searching in spaces, characterizing each solution candidate as a point in that space.

Genetic algorithms combine the effects of two processes: the survival of the fittest and a random interchange of characteristics between individuals of the population of candidate solutions. Such combination allows using GA as a searching algorithm for optimization problems (Golberg, 1989 [16]). A very interesting property of GA is its highly potential for parallelizability.

In a general description of how a GA works, it starts with a randomly generated initial population, corresponding to potential solutions of the considered problem. On every individual there is defined a fitness function that gives an indication of how far the individual is from a solution of the problem (in the biological simile this is equivalent to the capacity of the individual obtaining natural resources). Using the operations crossover and mutation, a new generation of individuals is obtained. Sometimes, some of these new individuals have better fitness values, that is, are better solutions

candidates. The iteration over time causes that considering the best candidate, the algorithm converges to the solution.

The algorithm can be resumed as:

```
g=0
Generate initial population, P(g);
evaluate P(g);
repeat
        g:=g+1;
        generate P(g) from P(g-1); // crossover and mutation
        evaluate P(g);
until convergence
```

4. The proposed genetic method

Sometimes the best option to solve a problem is using hybrid techniques, taking advantages of all the methods and reducing the problems inherent to each of them. For example, the A*-search algorithm used in autonomous robot navigation, vehicles or video games characters, (Schwab, 2004 [17]), uses too much memory. To avoid this problem, techniques as the iterative deepening A* (IDA*) are developed, combining the A*-search and an iterative deepening search (Russell, 2004 [14]).

Other hybrid techniques combine informed strategies (like A*) with irrevocable searches (like gradient descent).

In this paper the proposed algorithm combines the philosophy of genetic algorithms with modified crossover and mutation operators that improve the convergence velocity.

4.1. **Vector representation and fitness function.**

Usually, a candidate point $x_i = (x_{i1}, x_{i2}, \ldots x_{in})$ is represented using an array of bits, called chromosome. Each component is codified with b bits, so the entire point is represented by $n \cdot b$ bits.

In the proposed method a chromosome is codified as an array of doubles $(x_1, x_2, \ldots x_n)$. The considered vectors are normalized:

$$\sum_k x_K^2 = 1 \qquad (4.1)$$

The initial population is generated creating individuals with random components obtained from a constant distribution in the interval [-1,1], normalizing each vector

afterwards. The population is uniformly distributed over the n-dimensional hypersphere with radius 1.

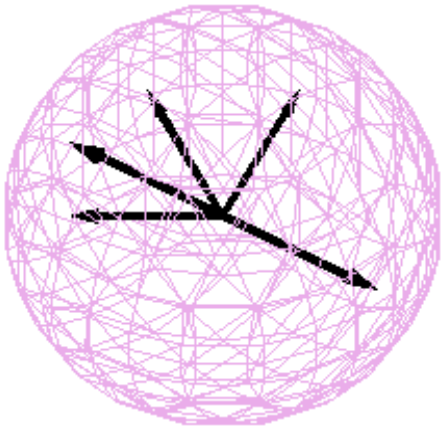

Figure 2. Example of initial population

4.2. **Selection of individuals**

Given a population, the process of determining the individuals that will take part in the crossover operation is called selection. Under some circumstances (survival of the fittest individual) the algorithm obtains improvements in the best fitness.

The functions defined in (2.7) and (2.8) can be used as fitness functions in the genetic algorithm. However, there are not significant differences between the convergence rates of the respective derived algorithms.

In the considered algorithm the selection is completely deterministic because just the NP individuals with the best fit are selected (NP is the size of the population).

4.3. **Crossover operator**

Crossover operator interchanges the genetic material between the previously selected individuals of the population. The combination of two vectors results in a new vector obtained from a probability distribution as shown in figure 3.

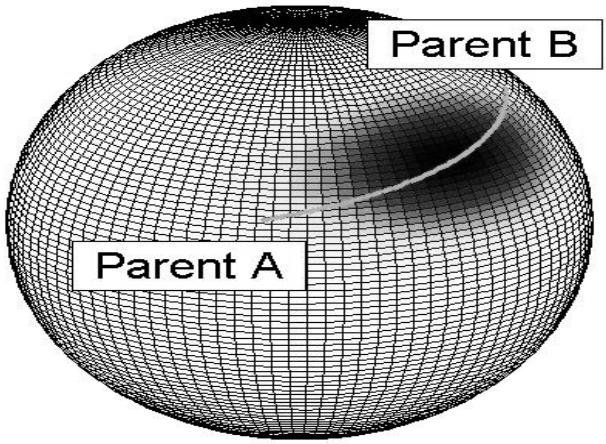

Figure 3 Influence area for the crossover operator.

For each survivor a couple is randomly selected from the rest of survivors. These pairs are combining using the crossover operation. In such operation, giving the pair of vectors (x1, x2, …xd) and (y1, y2, …yd), the descendant is constructed using the expression:

$$z_k = \frac{x_k + y_k}{2} \cdot [1 + \pi_k] \qquad \pi : N^d\left(0, F(fitness(x), fitness(y))\right) \tag{4.2}$$

That is, the new vector is normally distributed around the mean vector with the variance being a function F of the respective fitness.

4.4. **Mutation**

The mutation operator modifies some of the components of the vector. With a small probability of mutation the individual is very similar to its parents.

This operator maintains a minimum degree of diversity in the components of the candidate vectors. If some of these components have converged for the entire population the crossover operator cannot change it.

The modification a component has the expression:

$$x_i' = x_i + \Delta x \quad ; \Delta x : N\left(0, F\left(fitness\,(x)\right)\right) \tag{4.3}$$

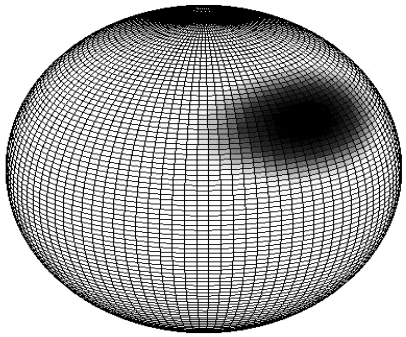

Figure 4 Influence area of mutation perturbation

Figure 4 shows the probability distribution for the mutated vector around the initial vector.

5. Examples

The proposed algorithm is tested in seven problems, comparing the results with those obtained by other methods or with the values previously known.

Firstly the methodology has been tested in a symmetric matrix that represents a system of masses connected by springs. Furthermore two Hermitian matrices corresponding to quantum mechanics problems have been selected.

Secondly the Principal Component Analysis (PCA) has been applied by the diagonalization of the data covariance matrix comparing the results.

Finally it has been selected three examples in order to compare the methodology with the power method and the QR algorithm, including the studies of a degenerate and an ill-conditioned matrix.

In the examples the mutation probability has been set to 0.005 (0.5 %), and the population size is 100.

5.1. Example 1.Vibration problem

Eigenvalue/Eigenvector analysis is useful for a wide variety of differential equations, and it can be used in the study of vibration problems. Considering a system of masses connected by springs, it can be split in a number of pieces with mass m. Each one of these pieces is under uniform tension, T. Each mass, has a displacement from the equilibrium position, xi. The equations containing the dynamics of each of this spring pieces have the form:

$$m \cdot \ddot{x}_i = \alpha \cdot x_{i-1} - 2 \cdot \alpha \cdot x_i + \alpha \cdot x_{i+1} \tag{5.1}$$

With an adequate selection of units $m = 1, \alpha = 1$, the system can be written as:

$$\ddot{x}_i = A \cdot x \tag{5.2}$$

$$A(i,j) = \begin{cases} -2, & i = j \\ 1, & |i - j| = 1 \\ 0 & otherwise \end{cases} \tag{5.3}$$

In table 1 the eigenvalues obtained by the algorithm and those provided by Matlab can be compared.

	Results	
Eigenvalue nº	**Genetic Algorithm**	**Matlab**
1	-3.918985947090077	-3.918985947228995
2	-3.682507065763702	-3.682507065662362
3	-3.309721467871853	-3.309721467890570
4	-2.830830026020166	-2.830830026003773
5	-2.284629676550326	-2.284629676546571
6	-1.715370323452336	-1.715370323453430
7	-1.169169974012325	-1.169169973996227
8	-0.690278532110185	-0.6902785321094292
9	-0.317492934347117	-0.3174929343376384
10	-0.081014052781915	-0.08101405277100521

Table 1 Eigenvalue in an interconnected spring's system

Figure 5 shows average of the error over 100 executions of the algorithm for the best individual in each generation for the first three eigenvectors:

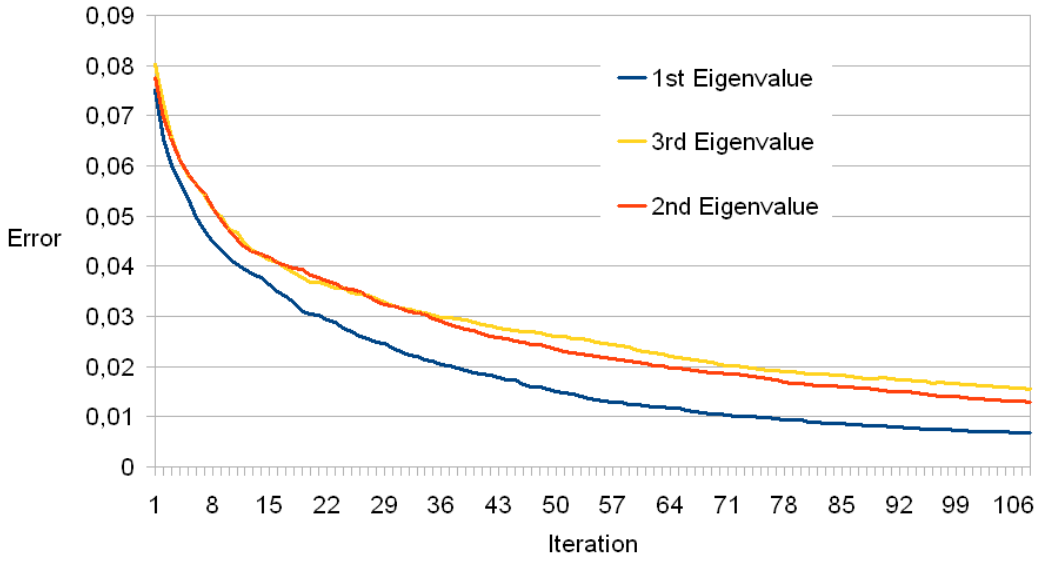

Figure 5 Evolution of the error for the three first eigenvectors

5.2. Example 2. Hermitian matrix.

To check the behaviour of the algorithm in the case of a Hermitian matrix, a quantum mechanics example is considered, comparing the obtained values and the exact numeric solutions.

Given a quantum system with total angular momentum of 5/2, a basis for the state space is formed by the eigenvectors of the J_Z operator, represented as $\left\{ \left| 5/2 \quad m \right\rangle_{J_Z} \right\}_{m=-5/2}^{m=5/2}$.

The quantum operator J_Y can be written in that basis as:

$$J_Y \sim \begin{pmatrix} 0 & -i\cdot\sqrt{5} & 0 & 0 & 0 & 0 \\ i\cdot\sqrt{5} & 0 & -i\cdot\sqrt{8} & 0 & 0 & 0 \\ 0 & i\cdot\sqrt{8} & 0 & -i\cdot\sqrt{9} & 0 & 0 \\ 0 & 0 & i\cdot\sqrt{9} & 0 & -i\cdot\sqrt{8} & 0 \\ 0 & 0 & 0 & i\cdot\sqrt{8} & 0 & -i\cdot\sqrt{5} \\ 0 & 0 & 0 & 0 & i\cdot\sqrt{5} & 0 \end{pmatrix} \qquad (5.4)$$

Its eigenvalues are well known, being proportional to $\{-5,-3,-1,+1,+3,+5\}$.

The eigenvectors have been obtained after 7000 generations. Table 2 shows the exact eigenvalues and the values obtained by the genetic algorithm. The relative error is also

calculated.

Exact eigenvalue	Calculated eigenvalue	% Error
5	4.99999999978128	$4.4 \cdot 10^{-9}$
3	3.0000000000209	$7.0 \cdot 10^{-10}$
1	0.998640586530327	$1.4\,10^{-3}$
-1	-0.998640586493665	$1.4\,10^{-3}$
-3	-3.0000000000619	$2.1 \cdot 10^{-9}$
-5	-4.99999999977693	$4.5 \cdot 10^{-9}$

Table 2 Exact value, approximate solution and error

The following Figure 6 shows the evolution for the error in each generation:

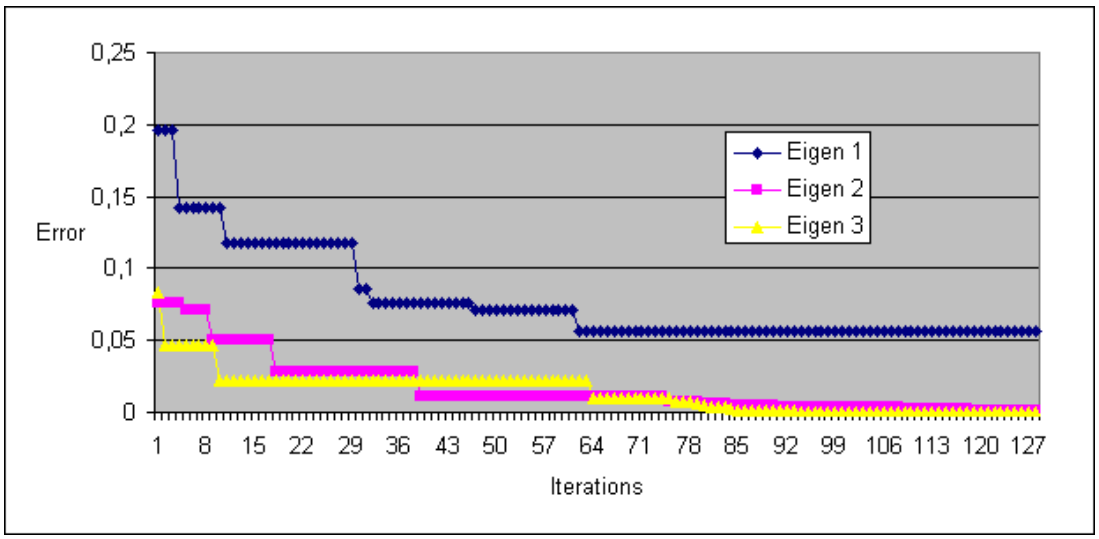

Figure 6. Error evolution for the first eigenvectors

5.3. Example 3**.** Coope Matrix

An important application of the diagonalization problem is quantum mechanics. Under some conditions, solving Schrödinger equation is equivalent to the calculation of the eigenvalues and eigenvectors of the following matrix (Coope matrix):

$$H(i,\ j) = \begin{cases} 2i - 1, i = j \\ 1, \text{i} \neq \text{j} \end{cases}$$

There are previous works (Subhajit, 2011 [11]) where the problem is solved for the first eigenvalues using a genetic algorithm. For the methodology introduced in the present paper, a study for the relationship between the matrix size and the temporal cost of the program is done. With the obtained data the algorithm performance is estimated (table 4).

Matrix dimensión	Time(seconds)
5	0,48243333
8	1,71365
10	5,13338667
12	13,9806667
15	31,9464667
17	63,6046333

Table 3. Computational time for the algorithm

In the next figure, the data from table 3 is represented, obtaining the relationship between the dimension of the matrix and the computing time.

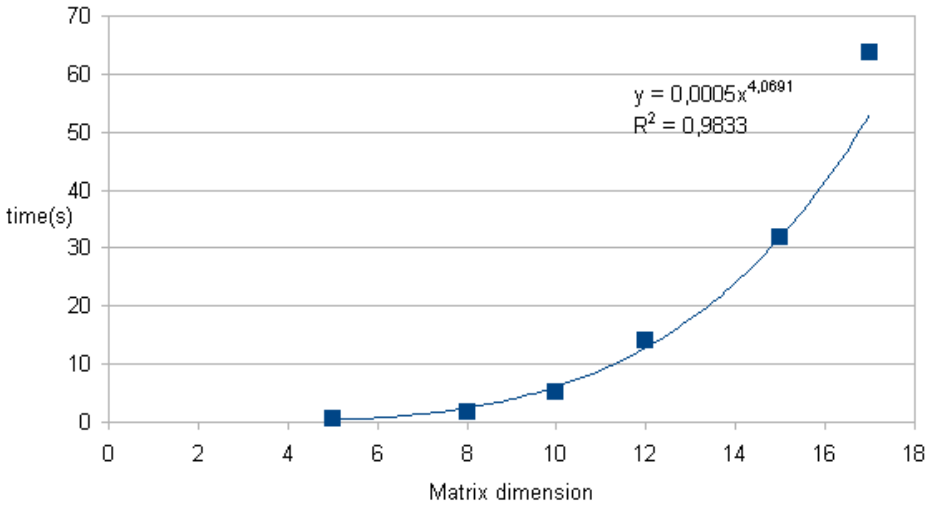

Figure 7 Adjust of computational efficiency of the algorithm

Another study is to consider the reduced problems, calculating only five eigenvectors for each matrix. The resulting times are:

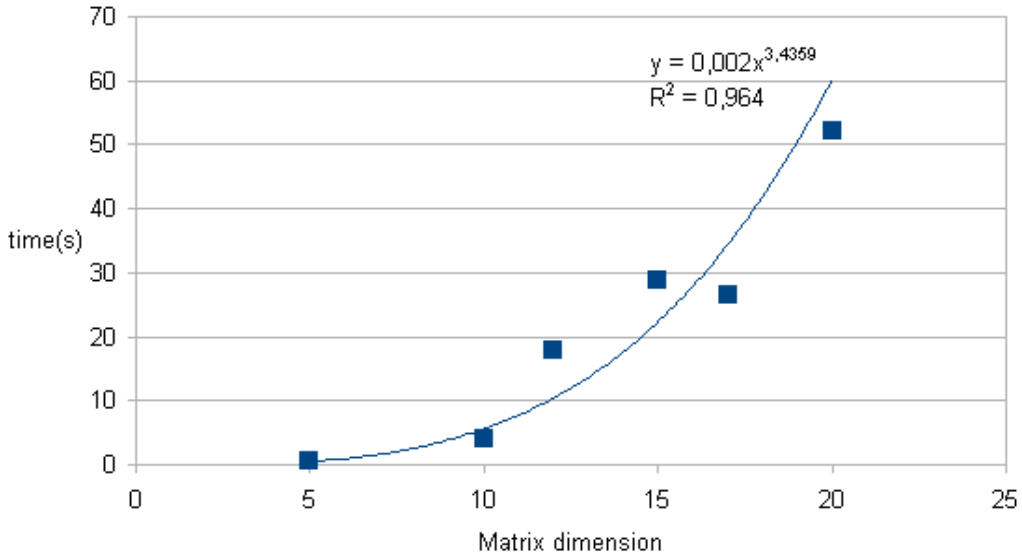

Figure 8 Adjust of computational efficiency of the algorithm for the reduced problem (only five eigenvectors).

5.4. Example 4. Principal component analysis

Principal component analysis (PCA) is a statistical method used to convert a set of observations of D variables into a set of D' (D'≤D) uncorrelated orthogonal variables called principal components. Obtaining these components is equivalent to the diagonalization of the covariance matrix of the data.

These components are obtained in a form that is ordered using the variance (including the most part of the variability of the data).

PCA is used in several fields, by example in signal processing, mechanical engineering, linear algebra, meteorological science, structural dynamics, medical diagnostic, neuroscience, etc.

The next example is a physical problem in which data are analyzed to reduce the number of variables in the data. The dataset is available in the UCI Machine Learning Repository ([18]).

The description of the dataset given by the authors is: "The data are Monte Carlo program generated ([19]) to simulate registration of high energy gamma particles in a

ground-based atmospheric Cherenkov gamma telescope using the imaging technique. Cherenkov gamma telescope observes high energy gamma rays, taking advantage of the radiation emitted by charged particles produced inside the electromagnetic showers initiated by the gammas, and developing in the atmosphere. This Cherenkov radiation (of visible to UV wavelengths) leaks through the atmosphere and gets recorded in the detector, allowing reconstruction of the shower parameters. The available information consists of pulses left by the incoming Cherenkov photons on the photomultiplier tubes, arranged in a plane, the camera. Depending on the energy of the primary gamma, a total of few hundreds to some 10000 Cherenkov photons get collected, in patterns (called the shower image), allowing to discriminate statistically those caused by primary gammas (signal) from the images of hadronic showers initiated by cosmic rays in the upper atmosphere (background).

Typically, the image of a shower after some pre-processing is an elongated cluster. Its long axis is oriented towards the camera centre if the shower axis is parallel to the telescope's optical axis, i.e. if the telescope axis is directed towards a point source. A principal component analysis is performed in the camera plane, which results in a correlation axis and defines an ellipse. If the depositions were distributed as a bivariate Gaussian, this would be an equidensity ellipse. The characteristic parameters of this ellipse (often called Hillas parameters) are among the image parameters that can be used for discrimination. The energy depositions are typically asymmetric along the major axis, and this asymmetry can also be used in discrimination. There are, in addition, further discriminating characteristics, like the extent of the cluster in the image plane, or the total sum of depositions."

The dataset is formed by 10 variables:

1. fLength: continuous. Major axis of ellipse [mm]
2. fWidth: continuous. Minor axis of ellipse [mm]
3. fSize: continuous. 10-log of sum of content of all pixels [in #phot]
4. fConc: continuous. Ratio of sum of two highest pixels over fSize [ratio]
5. fConc1: continuous. Ratio of highest pixel over fSize [ratio]
6. fAsym: continuous. Distance from highest pixel to centre, projected onto major axis [mm]
7. fM3Long: continuous. 3rd root of third moment along major axis [mm]

8. fM3Trans: continuous. 3rd root of third moment along minor axis [mm]
9. fAlpha: continuous. Angle of major axis with vector to origin [deg]
10. fDist: continuous. Distance from origin to centre of ellipse [mm]

A principal component analysis performed by the free computational software R ([20]) has obtained seven vectors as the main contributors to the variance (accounting for the 0.99999). In the proposed methodology the eigenvectors for the covariance matrix have been calculated. The results are similar in both cases as can be seen with detail in Annex I.

The next table shows a short comparison between both results, where the proportion of variance of each component can be seen:

	Comp. 1	Comp. 2	Comp. 3	Comp. 4	Comp. 5	Comp. 6	Comp. 7
Standard deviation	81,1159	62,0795	44,9042	36,42471	24,70197	20,82304	10,8319
Proportion of variance	0,4405	0,258	0,135	0,08882	0,04085	0,02903	0,00785
Cumulative proportion	0,4405	0,6985	0,8335	0,92226	0,96311	0,99214	0,99999
Error module	0,00163	0,00347	0,00181	0,01393	0,00995	0,00150	0,01625

Table 4 Proportion of variance of each component

The errors between the R-software and the genetic method eigenvectors are displayed in the Error module row. These errors have been calculated as:

$$error_i = \sqrt{\sum_i \left(V^i_{genetic} - V^i_R\right)^2}$$

5.5. Example 5. Comparison with the power method

Given the matrix A, the Matlab implementation of the power method has been used to obtain the eigenvalue with the greatest absolute value and the corresponding eigenvector. The results are compared with those obtained using the new method.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

Using Matlab, the obtained eigenvector is v= [1, 0.8198, 0.8198, 1], and the corresponding eigenvalue must verify: $\lambda * v = A * v$=[9.0990, 7.4594, 7.4594, 9.0990]. According with the proposed methodology, the obtained eigenvector is:

v= [1, 0.8207, 0.8197, 1] and the resulting value for λ is 9.09902, so the results are similar.

5.6. Example 6. Degenerate matrix

For the degenerate matrix B, the results obtained with the QR algorithm in Matlab and the genetic algorithm are equal. That is, a triple eigenvalue λ=1 and a single one, λ=5

$$B = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

5.7. Example 7. And ill conditioned matrix

Given the stochastic matrix M, which has high-magnitude elements, the results obtained with Matlab using 32 digits and arithmetic precision (considering this as the exact solution) are shown in Table 5 The results obtained with the proposed methodology are coincident with Matlab (arithmetic precision), as can be checked in the same table.

$$M = \begin{pmatrix} 10^{40} & 10^{29} & 10^{19} \\ 10^{29} & 10^{20} & 10^{9} \\ 10^{19} & 10^{9} & 10^{1} \end{pmatrix}$$

	32Dig	Precision arithmetic	Genetic algorithm
λ_1	-3,96678784561050e+23	9.981818	9.98182
λ_2	-8,10000976406272e+19	$9{,}9000*10^{19}$	9.9e+19
λ_3	1,00000000000000e+40	$1{,}000000*10^{40}$	1e+40

Table 5 Comparison of eigenvalues

6. Conclusions

In this paper the authors have developed a non-deterministic method, based on a genetic algorithm, to obtain the solution to the complete eigenvector problem. The main result is the set of eigenvectors, and from them it is possible to estimate the corresponding eigenvalues.

The method can be used with symmetric and hermitian matrices that are common in scientific and engineering problems.

The selected examples show that the methodology obtains precise results in the real and complex cases, comparing them with the values obtained by other tools or theoretically.

In the third example, the velocity of convergence decreases with the matrix dimension. The reason of this is that the search is random in a neighbourhood of the parent vectors. This volume increases with the dimension as L^n where L is the radius of the zone affected by the normal distribution in the operations of crossover and mutation.

The future lines of research include designing new fitness functions and improvements in the crossover and mutation operators to increase the convergence velocity. Another study will be the comparisons of the proposed methodology with the classical genetic algorithm approach of binary coding, trying to understand the causes of the better behaviour of the proposed method.

References

[1] J Stoer, R Bulirsch. Introduction to numerical analysis. 2002

[2] Süli, Endre and David Mayers. Introduction to numerical analysis.2003.

[3] B.P. Demidovich, I.A. Maron. Cálculo Numérico Fundamental. Ed. Paraninfo, 1993

[4] E.A. Volkov “Métodos Numéricos”, 1987.

[5] B.N. Parlett. Te Symmetric Eingenvalue problema, Prentice Hall. Englewood Cliff, NJ. 1981.

[6] J.H. Wilkinson. The Algebraic Eigenvalue, Clarendon, Oxford, 1965.

[7] J.K. Cullum, R.A. Willonghby. Lanczos Algorithm for Large Symetric Eigenvalue Computations. Birkhauss, Boston, 1985.

[8] S. Kirkpatrick; C.D. Gelett; M.P. Vecchi. Optimization by simulated annealing. Science 67(1983) 671.

[9] P.J.M Van Laarhoven,; E.H.L. Aarts. Simuklated Annealing: theory and Applications. Kluwer. Dordrechet. 1987.

[10] Goldberg, David E. Genetic Algorithm in Search Optimization and Machine Learning. Addison Wesley.Reading, MA. 1989.

[11] Subhajit Nandy; Rahul Sharma; S.P. Bhattacharyya, Solving symmetric eingenvalue problem via genetic algorithms: Serial versus parallel implementation. Applied Soft Computing. 11 (2011), 3946-3961.

[12] Golub, G.H and Van Loan, C.F. Matrix Computations. Third Edition The John Hopkins University Press, (Baltimore and London). 1996.

[13] An Introduction to the Numerical Analisys. Endre Süli and David F. Mayers. Cambrigde University Press, 2003

[14] Stuart Russell and Peter Norvig. Inteligencia Artificial un enfoque moderno. Prentice Hall, 2004).

[15]Holland J H 1975 Adaptation in natural and artificial systems (Ann Arbor, MI: University of Michigan Press)

[16] Goldberg, David E. Genetic Algorithm in Search Optimization and Machine Learning. Addison Wesley.Reading, MA. 1989

[17] Brian Schwab, Charles River . AI Game Engine Programming. Media Inc. 2004.

[18] Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository (MAGIC Gamma Telescope Data Set)[http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[19] Corsika, described in D. Heck et al., CORSIKA, A Monte Carlo code to simulate extensive air showers, Forschungszentrum Karlsruhe FZKA 6019 (1998)).

[20] The R Project for Statistical Computing (http://www.r-project.org/).

ANNEX. I

R –Principal Components results:

PC1=(0,3278123051 ; 0,1138100005 ; 0,0030607669 ; -0,0009271091 ; -0,0005218824 ; -0,3807000111 ; -0,0716522956 ; 0,0033740577 ; -0,0535411190 ; 0,8524267069)

PC2=(0,1344063317 ; 0,0538837224 ; -0,0006593720 ; 0,0002582053 ; 0,0001527100 ; -0,6938974103 ; -0,5626201689 ; 0,0010323557 ; 0,1183514088 ; -0,4086413654)

PC3=(-0,0655078481 ; 0,0189938175 ; -0,0010593176 ; 0,0005103997 ; 0,0002946359 ; 0,5400768346 ; -0,8131211084 ; 0,0124715329 ; 0,0531065391 ; 0,1988007027)

PC4=(0,863988253 ; 0,319766485 ; 0,006930287 ; -0,002605551 ; -0,001512759 ; 0,285130223 ; 0,075884464 ; 0,011894809 ; 0,092513865 ; -0,235496475)

PC5=(0,0945511967 ; -0,0197548493 ; 0,0026518525 ; -0,0015389987 ; -0,0009193406 ; -0,0054838792 ; -0,1022108038 ; -0,0097559004 ; -0,9842422099 ; -0,1065577049)

PC6=(1,985520e-02 ; -2,325923e-02 ; -1,334415e-04 ; 3,464377e-05 ; 2,215016e-05 ; 8,839391e-03 ; -9,938329e-03 ; -9,993549e-01 ; 1,289978e-02 ; 3,348567e-03)

PC7=(-0,338166662 ; 0,938307839 ; 0,010882323 ; -0,003062871 ; -0,001813869 ; -0,021981496 ; 0,029176149 ; -0,029750433 ; -0,053260554 ; -0,005865354)

Genetic algorithm results:

PC1=(0,328496 ; 0,113855 ; 0,00248505 ; -0,00131838 ; -0,00106787 ; -0,380012 ; -0,0710994 ; 0,00326351 ; -0,0527718 ; 0,85256)

PC2=(0,135853 ; 0,0542528 ; -0,00112985 ; 0,000615244 ; -0,00289937 ; -0,693855 ; -0,562397 ; 0,00103028 ; 0,118527 ; -0,408431)

PC3=(-0,0649912 ; 0,0206752 ; -0,00104675 ; 0,000684572 ; 0,000435467 ; 0,540202 ; -0,81311 ; 0,0123226 ; 0,0531574 ; 0,198504)

PC4=(0,863418 ; 0,318954 ; 0,0118596 ; 0,00968647 ; -0,000239244 ; 0,285831 ; 0,0779287 ; 0,0112382 ; 0,0958263 ; -0,235503)

PC5=(0,0961946 ; -0,0137564 ; -0,00469309 ; 0,000866045 ; -0,000865649 ; -0,00493433 ; -0,101872 ; -0,0098289 ; -0,98419 ; -0,10679)

PC6=(0,0192692 ; -0,023674 ; 0,0011457 ; -5,56369e-05 ; -0,00010642 ; 0,00864459 ; -0,00988361 ; -0,999359 ; 0,0128022 ; 0,00338023)

PC7=(-0,337772 ; 0,938708 ; -0,00393605 ; 0,000388873 ; -6,7101e-05 ; -0,0226902 ; 0,0308158 ; -0,0298931 ; -0,0482828 ; -0,0056205)