Learning Linear Operators by Genetic Algorithms

JEAN FABER[1], RICARDO N. THESS[1], GILSON A. GIRALDI[1]

[1]LNCC–National Laboratory for Scientific Computing -
Av. Getulio Vargas, 333, 25651-070 Petropolis, RJ, Brazil
{faber,rnthess,gilson}@lncc.br

Abstract. In this paper we consider the situation where we do not know a linear operator L but instead have only a set S of example functional points of the form (x, y) such that $L(x) = y$. This problem can be analysed from the viewpoint of numerical linear algebra or learning algorithms. The later is the focus of this work. Firstly, we present a method found in the literature to learn quantum (unitary) operators. We analyse the convergence of the learning algorithm and show its limitations. Next, we propose a new method based on genetic algorithms (GAs). We discuss the results obtained by the GA learning technique and compare the method proposed with traditional approaches in the field of numerical solution of linear systems.

1 Introduction

The problem of finding a linear function $f : \Re^n \rightarrow \Re$ through a sample set $S = \{(x, y)\,; \quad y = f(x)\}$ is very common in fields like signal processing [10], economy and social sciences.

In practical applications, the challenge in general is to find an algorithm which takes S as input and returns as output a function $\tilde{f}$ which approximates f.

From the point of view of Artificial Intelligence methods it can be considered as an *inductive learning* problem, in which we want to learn a rule (the function f) from the data set S [19].

Inductive learning may be *supervised* or *unsupervised*. In the former, it is assumed that at each instant of time we known in advance the response of the system (the function f, in our case) [19, 7]. In the later, learning without supervision, we do not know the system response. Therefore, we can not explicitly use the errors of the learning algorithm in order to improve its behavior [19].

A common approach in this field is to minimize an error function E. The optimization can be done by Random searches (Metropolis, Genetic algorithms), gradient search, second order search, among others [19].

A special class of problems arrises if the target function is a linear operator $L : V \rightarrow V$, where V is a real or complex vector space of dimension $\dim(V) = n$.

Recently, the problem of estimating an unitary operator $\tilde{L}$ was formulated in the context of learning algorithms by Dan Ventura [20]. The central idea follows the basic methods in neural networks [2]: (1) A *guess operator* is chosen; (2) A *test* to compare the obtained output with the desired one ; (3) A rule to adapt parameters if need.

A special place for unitary operators is Quantum Computation and Quantum Information [14, 16, 15]. In these fields, the computation is viewed as effected by the evolution of a physical system, which is given by unitary operators, according to the Laws of Quantum Mechanics [14].

In this paper we focus on learning algorithms to estimate a linear (unitary or non-unitary) operator L.

Firstly, we analyze the solution proposed in [20] for the unitary case. We focus on the convergence of that learning algorithm and show its limitations.

Next, we propose a new solution by using genetic algorithms. We discuss the advantages and difficulties of our method. The main advantage is its generality: it can be used to learn unitary and non-unitary linear operators. We compare our results with the leaning method proposed by Dan Ventura [20] and with traditional iterative methods [6, 17].

The paper is organized as follows. Next, we put our basic problem in terms of error minimization and gives an interesting and useful geometric interpretation 2. Section 3 shows the Dan Ventura 's learning method for unitary operators [20]. We also discuss some properties and limitations of this approach in section 3.2. Section 4 gives a briefly introduction to genetic algorithms. Our implementation is presented on section 5. In the experimental results (section 6) we analyze the efficiency of our GA algorithm to learn operators defined on vector spaces of dimensions 2, 3, 4 and 6. We present a case which Dan Ventura 's method can not solve but our GA learning method does. We discuss these results in section 7 and compare the computational efficiency of the GA method with traditional numerical approaches (described on Appendix A). The final considerations are given on section 8.

2 Problem Analysis

The problem we are in face is to find a linear operator $L : V \rightarrow \Re$ given a sample set $S = \{(v_i, u_i) \in V \times V; \quad L(v_i) = u_i; \quad i = 1, ..., K\}$.

If $x_i, y_i \in \Re^n$ are the matrix representation of v_i, u_i, respectively, in some basis [8], then an equivalent problem is to find A such that $y_i = Ax_i$ (A is a matrix representation of the operator L [8]).

It might be convenient to rewrite this problem as an optimization one. Specifically, let us consider the objective function:

$$Objective\,(A) = \frac{1}{n}\sum_{i=1}^{K} \|Ax_i - y_i\|^2 . \tag{1}$$

Then, the matrix we are looking for is the solution of the following problem:

$$\underset{A \in \Re^{n\times n}}{min} \quad \frac{1}{n} J\,(A)$$

where:

$$J\,(A) = \sum_{i=1}^{K} \|Ax_i - y_i\|^2 . \tag{2}$$

To minimize the above *Objective* function , we have to look for solutions of the equation:

$$\nabla J\,(A) = 0. \tag{3}$$

It is interesting to observe that these solutions are the stationary points of the system:

$$\frac{dA}{dt} = -\nabla J\,(A)\,. \tag{4}$$

An interesting result comes from the analysis of this problem when A is real. Let us consider the symmetric matrix:

$$H\,(J) = \begin{bmatrix} \frac{\partial^2 J}{\partial a_{11}^2} & \frac{\partial^2 J}{\partial a_{12}\partial a_{11}} & \cdots & \frac{\partial^2 J}{\partial a_{nn}\partial a_{11}} \\ \frac{\partial J}{\partial a_{11}\partial a_{12}} & \frac{\partial^2 J}{\partial a_{12}^2} & \cdots & \frac{\partial J}{\partial a_{nn}\partial a_{12}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial J}{\partial a_{11}\partial a_{nn}} & \frac{\partial J}{\partial a_{12}\partial a_{nn}} & \cdots & \frac{\partial^2 J}{\partial a_{nn}^2} \end{bmatrix}; \tag{5}$$

called the Hessian of J (or the Jacobian of the field $F = \nabla J\,(A)$). Once $H\,(J)$ is real and symmetric its eigenvalues are all real.

If A^* is a solution of equation (3) and if all eigenvalues of $-H\,(J)$ are non-null and negative in A^*, then, by Hartman's Theorem [18], the system in expression (4) has an attractor in A^*; that is, the solution of the initial value problem:

$$\frac{dA}{dt} = -\nabla J\,(A)\,; \quad A\,(0) = A_0, \tag{6}$$

is exactly A^* if A_0 belongs to a neighborhood of A^*.

It is a dynamical interpretation of the very known fact that if A^* is a solution of equation (3) and if the eigenvalues of the Hessian matrix $H\,(J)$ are non-null and positive, then $J\,(A^*)$ is a local minimum and there is a basin of attaction for any gradient-based minimization method. The solution of equation (6) is a continuous version of a steepest descent method starting from A_0 [3].

When $K = n$ in S and the vectors $\{x_i; \quad i = 1, 2, ..., n\}$ are linearly independent (LI), then the optimization problem has only one solution (see Property 2 in Appendix A). The same is true if $K > n$ but there is a subset $\tilde{S} \subset S$ with this property.

For any other case ($K < n; K \geq n$ but without a linearly dependent set $\{x_i; \quad i = 1, 2, ..., n\}$), there will be infinite solutions.

The above discussion is worthwhile to give us a geometric interpretation of what is going when using GAs.

Let us consider that A^* is the global minimum (only one solution) and that the size of its basin of attaction is r_δ. As we shall see later, the basic idea behind GAs is to search for the solution by evolving a set of *candidate* through genetic inspired operations. Thus, if p_1, p_2 (Figure 1) are two such points, then the key idea is to design these operations in such a way that the sons of p_1, p_2 will be better than their parents; that is, closer the optimum. Thus, unlike steepest descent methods, in which we have one point following a solution of (6) we would have a set of candidates searching for it. However, we have to pay a price due to storage requirements and computational complexity. Later we shall discuss the related trade-offs.

To simplify the presentation that follows, we are going to use the following nomenclature: We say that we have an *underconstrained problem if* $K < n$; a *constrained/overconstrained problem* if we $K = n$ or $K > n$, respectively (n is the dimension of the vector space).

3 Learning Quantum Operators

This section describes the learning method found in [20] in the context of quantum operators. To let this paper self-contained we introduce some background next.

The field of quantum computation is a promising area posing challenges in fields of quantum physics, computer science and information theory [14, 16, 15].

Quantum computation and Quantum Information encompass processing and transmission of data stored in quantum states (see [15] and references therein). The process can be viewed as effected by the evolution of a quantum system which can be mathematically described by [14, 16]:

$$F : \mathcal{H} \rightarrow \mathcal{H}; \quad F \mid \chi\rangle = \mid \psi\rangle, \tag{7}$$

where $\mathcal{H}$ is a complex, finite dimensional Hilbert space (an inner product vector space which is complete with respect

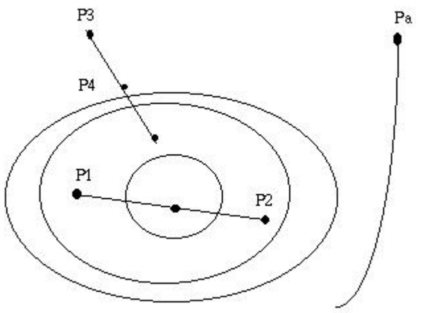


Figure 1: Genetic algorithms can take two parents, P_3 and P_4 for instance and generate a point P_{off} closer than the optimum despite the fact that they may be out of the attraction basin. Steepest descent fails if point P_a does not bellongs to the attraction region.

to the norm defined by the inner product [4]), F is a linear and unitary operator, and the pair $(\mid \chi\rangle, \mid \psi\rangle)$ represents the initial and final state, respectively, of the physical system. The notation " $\mid \chi\rangle$" ,for a vector, is the *Dirac notation* which is standard in quantum mechanics.

The inner product will be also represented in *Dirac notation*, as follows:

$$(\mid \chi\rangle, \mid \psi\rangle) \in \not{H} \times \not{H} \quad \underset{Inner \quad product}{\rightarrow} \quad \langle \chi \mid \psi\rangle \in C,$$

where C is the set of complex numbers and the function $\langle \cdot \mid \cdot\rangle$ follows the usual properties of inner products in complex vector spaces [4, 14].

In this context, the Dual to the vector $\mid \chi\rangle$ is denoted by $\langle \chi \mid$. This functional is defined as follows:

$$\langle \chi \mid (\mid \psi\rangle) = \langle \chi \mid \psi\rangle, \qquad \forall \mid \psi\rangle \in \not{H}.$$

It can be shown that, if $[z_0, z_1, ..., z_{n-1}]^T$ is the matrix representation of $\mid \chi\rangle$ with respect to some orthonormal basis, then $\left[z_0^*, z_1^*, ..., z_{n-1}^*\right]$ is the matrix representation of the Dual $\langle \chi \mid$ with respect to the corresponding Dual basis.

If $[z_0, z_1, ..., z_{n-1}]^T$, $[w_0, w_1, ..., w_{n-1}]^T$ are the matrix representations of $\mid \chi\rangle$ and $\mid \psi\rangle$ with respect to the same basis of $\not{H}$ then the (standard) inner product of $\mid \chi\rangle$ and $\mid \psi\rangle$ can be calculated by:

$$\langle \chi \mid \psi\rangle = \sum_{i=0}^{n-1} z_i^* w_i = \left[z_0^*, z_1^*, ..., z_{n-1}^*\right] \begin{bmatrix} w_0 \\ \cdot \\ \cdot \\ \cdot \\ w_{n-1} \end{bmatrix}.$$

3.1 Learning Algorithm

If we do not know an operator $F : V \rightarrow V$ but insteady we have a set of functional points $S = \{(\mid \chi_i\rangle, \mid \psi_i\rangle)\,; \quad F \mid \chi_i\rangle = \mid \psi_i\rangle, \quad i = 0, 1, ..., n\}$, where $\dim(V) = n$, also called the *learning sequence*, we can hypothesize a function G such that $\|G \mid \chi_i\rangle - \mid \psi_i\rangle\| \cong 0$ (as usual, $\|\mid v\rangle\| = \sqrt{\langle v \mid v\rangle}$ is the norm induced by the inner product).

The method proposed in [20] to find G is based on a supervised learning algorithm.

Let χ_k^i denotes the component k of the matrix representation of the vector $\mid \chi_i\rangle$ (equivalently for ψ_j^i and $\tilde{\psi}_j^i$). The method can be summarized as follows:

```
Initialization: Random Unitary Operator $G^0$ and $S$
For $i = 0, ...., n$
....Calculate $G^i \mid \chi_i\rangle = \mid\tilde{\psi}_i\rangle$,
...Update $G$ as
......$g_{jk}^{i+1} = g_{jk}^i + \delta\left(\psi_j^i - \tilde{\psi}_j^i\right)\chi_k^i$,
```

This algorithm is classified as supervised because at each interaction we known in advance the desired response.

As an example, we will consider the case studied in [20], where the set S and the operator G^0 are given by:

$$S = \left\{ \begin{array}{c} \left(\frac{1}{\sqrt{5}}\begin{bmatrix} 2 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{10}}\begin{bmatrix} 3 \\ 1 \end{bmatrix}\right) \\ \left(\frac{1}{\sqrt{20}}\begin{bmatrix} -2 \\ 4 \end{bmatrix}, \frac{1}{\sqrt{40}}\begin{bmatrix} 2 \\ -6 \end{bmatrix}\right) \end{array} \right\}, \tag{8}$$

$$G^0 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

Applying the first step of the algorithm we get:

$$G^0 \mid \chi_0\rangle = \mid\tilde{\psi}_0\rangle = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \tag{9}$$

$$\frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

If $\delta = 1$, the update rule gives:

$$g_{00}^1 = g_{00}^0 + \left(\psi_0^0 - \tilde{\psi}_0^0\right)\chi_0^0 = \tag{10}$$

$$0 + \left(\frac{3}{\sqrt{10}} - \frac{1}{\sqrt{5}}\right)\frac{2}{\sqrt{5}} \approx 0.449.$$

Calculating the rest of matrix entries in a similar way we obtain the first update of G :

$$G^1 \approx \begin{bmatrix} 0.449 & 1.224 \\ 0.083 & 0.541 \end{bmatrix}. \tag{11}$$

By repeating the process by taking the second pair $\left(\mid \chi\rangle^1, \mid \psi\rangle^1\right)$ of S we find the desired result:

$$G^2 \approx \begin{bmatrix} 0.707 & 0.707 \\ 0.707 & -0.707 \end{bmatrix} \approx \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{12}$$

which is the very known Hadamard Transform [14].

Despite of the success of Ventura's learning rule for this example, the way the algorithm works is not obvious. Besides, we need to characterizes the conditions to assure that the target will be found.

Next, we discuss the theory behind the algorithm and prove basic properties. That is the first contribution of this paper.

3.2 Convergence Analysis

Firstly, we must observe that, if S is a set of real vectors, like in the above example, the updating rule can be rewritten as:

$$G^{i+1} = G^i + \delta \left(\mid \psi_i\rangle - \mid\tilde{\psi_i}\rangle\right) \langle \chi_i \mid, \tag{13}$$

where we are using the fact that ψ^i_j is the component j of the matrix representation of vector $\mid \psi_i\rangle$. The same for χ^i_k and the Dual $\langle \chi_i \mid$ (we are considering real vectors).

Thus, from the above equation, the application of G^{i+1} to the state $\mid \chi_i\rangle$ gives:

$$G^{i+1} \mid \chi_i\rangle = G^i \mid \chi_i\rangle + \delta \left(\mid \psi\rangle^i - \mid\tilde{\psi}\rangle^i\right) \langle \chi_i \mid \chi_i\rangle. \tag{14}$$

In Quantum Mechanics, every state $\mid \chi_i\rangle$ is normalized. Therefore, we have $\langle \chi_i \mid \chi_i\rangle = 1$. Thus, equation (14) becomes:

$$G^{i+1} \mid \chi_i\rangle = G^i \mid \chi_i\rangle + \delta \left(\mid \psi_i\rangle - \mid\tilde{\psi_i}\rangle\right). \tag{15}$$

But, from Ventura's algorithm we have $G^i \mid \chi_i\rangle = \mid\tilde{\psi_i}\rangle$. If we set $\delta = 1$, we find that expression (14) becomes:

$$G^{i+1} \mid \chi_i\rangle = \mid \psi_i\rangle. \tag{16}$$

But, by hypothesis, we know that there is an operator G such that $G \mid \chi_i\rangle = \mid \psi_i\rangle$. This fact, added to the last equation, produces:

$$G^{i+1} \mid \chi_i\rangle = \mid \psi_i\rangle, \tag{17}$$

$$G \mid \chi_i\rangle = \mid \psi_i\rangle. \tag{18}$$

By subtraction these equations we find:

$$\left(G - G^{i+1}\right) \mid \chi_i\rangle = 0 \Longrightarrow \mid \chi_i\rangle \in Kernel\left(G - G^{i+1}\right). \tag{19}$$

From this result, it comes out the following constraints:

$$\left(G - G^1\right) \mid \chi_0\rangle = 0, \tag{20}$$

$$\left(G - G^2\right) \mid \chi_1\rangle = 0, \tag{21}$$

obtained by taking $i = 0$ and $i = 1$, respectively, in expression 19.

Now, we shall return to equation (13) and remember that we have set $\delta = 1$. Equation (13) becomes:

$$G^2 = G^1 + \left(\mid \psi_1\rangle - \mid\tilde{\psi_1}\rangle\right) \langle \chi_1 \mid. \tag{22}$$

which can be rewritten in the form:

$$G^2 - G^1 = \left(\mid \psi_1\rangle - \mid\tilde{\psi_1}\rangle\right) \langle \chi_1 \mid. \tag{23}$$

Applying to $\mid \chi_0\rangle$ both sides of this equation we find:

$$\left(G^2 - G^1\right) \mid \chi_0\rangle = \left(\mid \psi_1\rangle - \mid\tilde{\psi_1}\rangle\right) \langle \chi_1 \mid \chi_0\rangle. \tag{24}$$

If $\mid \chi\rangle^0$ and $\mid \chi\rangle^1$ are orthogonal then $\langle \chi_1 \mid \chi_0\rangle = 0$ and equation (24) becomes:

$$\left(G^2 - G^1\right) \mid \chi_0\rangle = 0. \tag{25}$$

Thus, from this result and the equation (20) we get:

$$\begin{aligned} \left(G - G^1\right) &\mid \chi_0\rangle = 0, \\ \left(G^2 - G^1\right) &\mid \chi_0\rangle = 0. \end{aligned}$$

Finally, by subtracting the second expression from the other we found that:

$$\left(G - G^2\right) \mid \chi_0\rangle = 0 \Longrightarrow \mid \chi_0\rangle \in Kernel\left(G - G^2\right). \tag{26}$$

Hence, from expression (19) and (26) it follows immediately that $G = G^2$. We shall remind that we have supposed that the vectors $\mid \chi\rangle^0$, $\mid \chi\rangle^1$ are unitary and orthogonal. Therefore, we have proved the following property:

Property 1: *If* $\{| \chi_0\rangle, | \chi_1\rangle\}$ *is a set of real vectors comprising an orthonormal basis of a bidimensional Hilber space* $\bar{H}$ *and* $\delta = 1$ *, then the algorithm of section 3.1 gives the unknown operator* G *after two interactions.*

This result can be generalized for dimension $n > 2$.

Now, it is clear why the algorithm gives the exact solution for the example of section 3.1. It is just a matter of observing that:

$$\left(| \chi\rangle^1\right)^T | \chi_0\rangle = \frac{1}{\sqrt{20}} \left[\begin{array}{cc} -2 & 4 \end{array} \right] \frac{1}{\sqrt{5}} \left[\begin{array}{c} 2 \\ 1 \end{array} \right] = 0, \quad (27)$$

and to notice that $\|| \chi_0\rangle\| = \|| \chi_1\rangle\| = 1$. Therefore, we have the conditions above stated and the property is verified.

But, what happens if $\langle\chi_1 \mid \chi\rangle^0 \neq 0$? This is discussed in the following example. We changed the set S but kept the initial guess G^0 used above:

$$S = \left\{ \begin{array}{c} \left(\frac{1}{\sqrt{5}} \left[\begin{array}{c} 1 \\ 2 \end{array} \right], \frac{1}{\sqrt{10}} \left[\begin{array}{c} 3 \\ -1 \end{array} \right] \right) \\ \left(\frac{1}{\sqrt{2}} \left[\begin{array}{c} 1 \\ 1 \end{array} \right], \frac{1}{\sqrt{4}} \left[\begin{array}{c} 2 \\ 0 \end{array} \right] \right) \end{array} \right\}, \quad (28)$$

$$G^0 = \left[\begin{array}{cc} 0 & 1 \\ -1 & 0 \end{array} \right].$$

The correct result is the Hadamard gate again (matrix (12)). However, set $\left\{| \chi\rangle^0, | \chi\rangle^1\right\}$ is not an orthonormal basis. The result obtained after two interactions is:

$$G^2 = \left[\begin{array}{cc} 0.2461879 & 1.168025 \\ -0.405649 & 0.405649 \end{array} \right], \quad (29)$$

which is far from the target.

This unsuccessful test shows that the conditions to assure correctness (Property 1, above) are restrictive even for the Quantum Mechanics (the operator to be learned and the vectors are unitary ones).

In this paper, we look for a more general supervised learning algorithm, which could overcome these limitations. We try to find out such method in the context of Genetic Algorithms. Next section gives a briefly introduction to this area.

4 Evolutionary Computation and GAs

In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering problems. The idea in all these systems was to evolve a population of candidate solutions for a given problem, using operators inspired by natural genetic and natural selection.

Since then, three main areas evolved: evolution strategies, evolutionary programming, and genetic algorithms. Nowadays, they form the backbone of the field of evolutionary computation [13, 1].

Genetic Algorithms (GAs) were invented by John Holland in the 1960s [9]. Holland's original goal was to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanism of natural adaptation might be imported into computer systems. In Holland's work, GAs are presented as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method for moving from one population of *chromosomes* to a new one by using a kind of *natural selection* together with the genetics-inspired operators of crossover and mutation. Each chromosome consists of *genes* (bits in computer representation), each gene being an instance of a particular *allele* (O or 1).

Traditionally, these crossover and mutations are implemented as follows [9, 13].

Crossover: Two parent chromosomes are taken to produce two child chromosomes. Both parent chromosomes are split into left and a right subchromosomes. The split position (*crossover point*) is the same for both parents. Then each child gets the left subchromosome of one parent and the right subchromosome of the other parent. For example, if the parent chromosomes are 011 10010 and 100 11110 and the crossover point is between bits 3 and 4 (where bits are numbered from left to right starting at 1), then the children are 01111110 and 100 10010.

Mutation: : When a chromosome is chosen for mutation, a random choice is made of some of the genes of the chromosome, and these genes are modified. The corresponding bits are *flipped* from 0 to 1 or from 1 to 0.

Next we present the basic definitions of the genetics-inspired operators for a real-coded GA, that is, for the case where the alleles are real parameters [21]. The algorithm proposed in [21] is a generalization of the $0 - 1$ case. It is useful as an introduction for our GA implementation (section 5).

4.1 Real-Coded Genetic Algorithm

The context of our interest is Genetic Optimization Algorithms. Simply stated, they are search algorithms based on the mechanics of natural selection and natural genetics and are used to search large, non-linear search spaces where expert knowledge is lacking or difficult to encode and where traditional optimization techniques fall short [5].

To design a standard genetic optimization algorithm, the following elements are needed:

(1) A method for choosing the initial population;

(2) A "scaling" function that converts the objective

function into a nonnegative fitness function;

(3) A selection function that computes the "target sampling rate" for each individual. The target sampling rate of an individual is the desired expected number of children for that individual.

(4) A sampling algorithm that uses the target sampling rates to choose which individuals are allowed to reproduce.

(5) Reproduction operators that produce new individuals from old ones.

(6) A method for choosing the sequence in which reproduction operators will be applied

For instance, in [21] each population member is represented by a chromosome which is the parameter vector $x = (x_1, x_2, ..., x_m) \in \Re^m$, and genes are the real parameters.

When alleles are allowed to be real parameters, some care should be taken to define these operators.

Mutations can be implemented as a perturbation of the chromosome. In [21] authors chosen to make mutations only in coordinate directions instead of to make in $\Re^m$ due to the difficult to perform global mutations compatible with the schemata theorem (it is a fundamental result for GAs [9, 5]).

Besides, the crossover in $\Re^m$ may also have problems. Figure 2 illustrates the difficult. The ellipses in the figure represent contour lines of the objective function. A local minimum is at the center of the inner ellipse. Points (x_1, y_1) and (x_2, y_2) are both relatively good points in that their function value is not too much above the local minimum. However, if we implement a traditional-like crossover (section 4) we may get points that are worse than either parents.

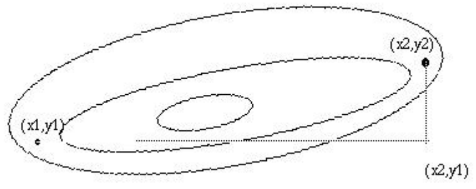


Figure 2: Crossover can generate points out of the attraction region.

To get around this problem, in [21] was propose another form of reproduction operator that was called linear crossover. From the two parent points p_1 and p_2 three new points are generated, namely:

$$\frac{1}{2}(p_1 + p_2); \qquad \frac{3}{2}p_1 - \frac{1}{2}p_2; \qquad -\frac{1}{2}p_1 + \frac{3}{2}p_2.$$

The best two of these three points are selected.

Inspired on the above analysis we propose the algorithm of section 5 to learn a linear operator from a set S of example functional points. It is the main contribution of this paper.

5 GA for Learning Operators

In our implementation, each population member (chromosome) is a matrix $A \in \Re^{n \times n}$, and alleles are real parameters (matrix entries). Up to now, the alleles are restricted to $[-1, 1]$ despite of the fact that more general situations can be implemented.

We consider two different strategies to generate the initial population: (a) All chromosomes are randomly generated; (2) The first member receives a seed. Then other ones are randomly chosen.

Once a population is generated, a fitness value is calculated for each member. The fitness function is defined by:

$$fitness\,(A) = \exp\left(-error\,(A)\right); \quad A \in \Re^{n \times n}, \quad (30)$$

where the error function is defined as follows. Let the learning sequence $S = \{(|\,\chi_i\rangle, |\,\psi_i\rangle)\,; \quad i = 1, ..., m\}$, then:

$$error\,(A) = \frac{1}{n \cdot m}\sum_{i=1}^{m} \left\| A\,|\,\chi_i\rangle - |\,\psi_i\rangle \right\|_1, \quad (31)$$

where $\|x\|_1$ denotes the 1-norm of a $x = (x_1, ..., x_n)$, defined by: $\|x\|_1 = |x_1| + ... + |x_n|$.

Once the fitness is calculated for each member, the population is sorted into ascending order of the fitness values. Then, the GA loop starts. Before enter the loop description, some parameters must be specified.

Elitism: It might be convenient just to retain some number of the best individuals of each population (members with best fitness). The other ones will be generated through mutation and/or crossover. This kind of selection method was first introduced by Kenneth De Jong [11, 13] and can improve the GA performance.

Selection Pressure: The degree to which highly fit individuals are allowed many offsprings [13]. For instance, for a selection pressure of 0.6 and a population with size N, we will get only the $0.6 * N$ best chromosomes to apply genetic operators.

Mutation Number: Maximum number of alleles that can undergo mutation. Like in [21], we do not choose to make mutations (implemented as perturbations also) in $\Re^{n \times n}$. We randomly choose some matrix entries to be perturbed.

Termination Condition: Maximum number of generations.

Mutation and Crossover Probabilities: Pm and Pc, respectively.

The crossover is defined as follows. Given two parents $A = [a_{ij}]$ and $B = [b_{ij}]$, randomly chose one of them. Next, take its ij component value and put it on c_{ij}. Once a soon is completed, the next offspring is created in the same way. Thus, we finally gave:

$$A, B \underset{Crossover}{\rightarrow} C_1, C_2$$

The mutation is implemented as a perturbation of the alleles. Thus, given a member A, the mutation operator works as follows:

$$A \underset{Mutation}{\rightarrow} A + \Delta$$

where Δ is a perturbation matrix. The mutation number establishes the quantity of non-null entries for Δ. They are defined according to the mutation probability and a pre-defined Perturbation Size, that is, a range $[a, b] \in \Re$, such that $a \leq \Delta_{ij} \leq b$.

Once the above parameters are pre-defined and the input data (set S) is given, the GA algorithm proceed. In the following pseudo-code block, $P(t)$ represents the population at the interaction time t and N is its size. $Ngen$ is the maximum number of generations allowed, the procedure $Evaluate\_Sort$ calculates the fitness of each individual and sort the chromosomes into ascending order of the fitness values. The integer Ne defines de elite members, the parameter $Ps \in [0, 1]$ defines the selection pressure and Nm the number of matrix entries that may undergo mutations.

```
Procedure Learning-GA
begin
........$t \leftarrow 0$;
........initialize $P(t)$;
........while($t < Ngen$) do
        begin
................$t \leftarrow t + 1$;
................$Evaluate\_Sort(P(t-1))$;
................Store in $P(t)$ the $Ne$ best members of $P(t-1)$;
................Complete $P(t)$ by crossover and mutation over the best $Ps * N$ members of $P(t-1)$;
......end
end
```

Some specific details of the genetic operators implementation shall be explained for completeness.

When applying a crossover; firstly, two members of $P(t-1)$ are randomly chosen. A random number between 0 and 100 is selected. If the crossover probability iguals or exceeds this number, then the genetic operator is applied. When crossover does not happens, the offsprings become a copy of the parents.

Now, mutation is applied. Again, a random number is generated and its value compared with the mutation probability to decide if mutation happens. If applied to a individual A, we randomly chosen Nm elements of A and apply a perturbation $a_{ij} \leftarrow a_{ij} + \Delta_{ij}$ for each one. In our implementation crossover and mutation are independent events in the sense that Pm and Pc do not depend on each other.

Finally, if we have complex entries in S, thus a complex operator L to learn, we can at first, apply the some core of the GA algorithm stated but now we have to consider that chromosomes are complex matrices that will undergo crossover and mutations in complex space. We must emphasize that further analysis should be made to assure the correctness of such scheme.

6 Experimental Results

The first point we focus in this section is the behavior of the algorithm against operator dimension.

Thus, we apply the method to learn two, three, four and six dimensional linear operators. This section reports the results obtained. Parameters values are reported on Table 1. They were described on section 5.

We observe that only the associated probabilities remain unchanged during all over the experiments.

Matrix	$Ngen$	N	Pc	Pm	Ps	Ne	Nm
2×2	100	200	0.85	0.95	0.30	30	1
2×2	200	200	0.85	0.95	0.30	30	1
3×3	300	200	0.85	0.95	0.30	30	2
3×3	155	300	0.85	0.95	0.30	30	2
4×4	100	100	0.85	0.95	0.30	30	2
6×6	3500	200	0.85	0.95	0.50	10	1
6×6	2200	200	0.85	0.95	0.50	10	1

Table 1: Parameters used in this section.

To improve convergence, we take the following rule: if the best member 's error is smaller than a user defined δ, than the upper bound for the perturbation is changed. This is an application dependent rule and will be specified case by case.

Besides, we added a constraint that the matrix elements belongs to the range $[-1, 1]$. This is a user defined range that should be set according to some prior knowledge about the desired operator.

As a performance measure of the genetic algorithm we collected the error of the best fitted member found within the maximum number of generations, over 25 runs. The error is calculated according to expression (31).

We checked the elapsed time per one run when using a Pentium-III 866MHz, 524 RAM, running Borland Delphi 5.

6.1 Operators in 2D

In this section we analyze the behavior of the GA for the same examples of section 3.1. It is reproduced bellow.

The set S and is given by:

$$S = \left\{ \begin{array}{l} \left(\frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right) \\ \left(\frac{1}{\sqrt{20}} \begin{bmatrix} -2 \\ 4 \end{bmatrix}, \frac{1}{\sqrt{40}} \begin{bmatrix} 2 \\ -6 \end{bmatrix} \right) \end{array} \right\}. \qquad (32)$$

The result over 25 runs was always the correct one (Hadamard Transform in expression 12). The set of parameters is given in the first line of Table 1 (above). The perturbation size is given by $[0.001, 0.1]$.

Figure 3 shows the error evolution over 25 runs. We collect the best population member (smallest error) for each run and take the mean value, for each generation, over the 25 runs. It suggests that the algorithm learns very fast and stabilizes itself during its execution. Indeed, this behavior was observed for all experiments we did.

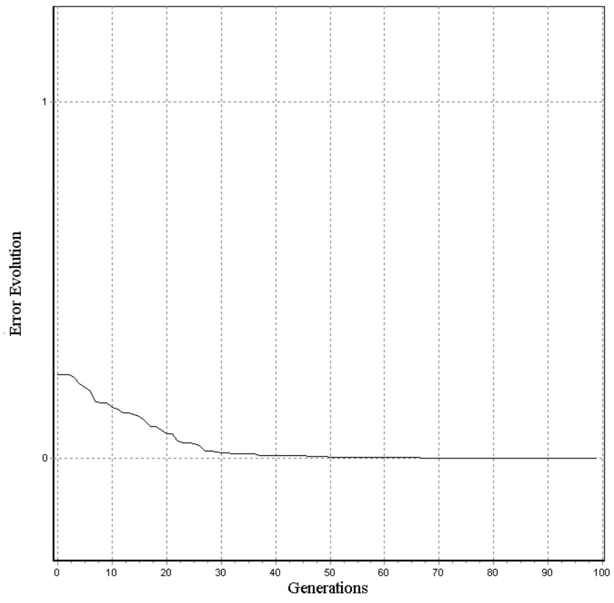


Figure 3: Error evolution over 25 runs for Ventura's example.

The next 2D example is given by the same set S used in 28:

$$S = \left\{ \begin{array}{l} \left(\frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ -1 \end{bmatrix} \right) \\ \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{4}} \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right) \end{array} \right\}. \qquad (33)$$

As we already explained on section 3.2, Ventura's algorithm [20] was not able to learn the operator because the set S do not agree with the Property 1.

Our GA algorithm was able to deal with this case.

The second line of Table 1 shows the parameters used. We decided to keep all parameters of Example 1 unchanged but the number of generations ($Ngen$) had to be increased to achieve the correct result. This point out that our GA method may be sensitive to Property 1, despite that it learns correctly. Further analysis should be made to reinforce (or not) this observation.

Figure 4 pictures the mean error evolution. It decays fast but take some time to become null.

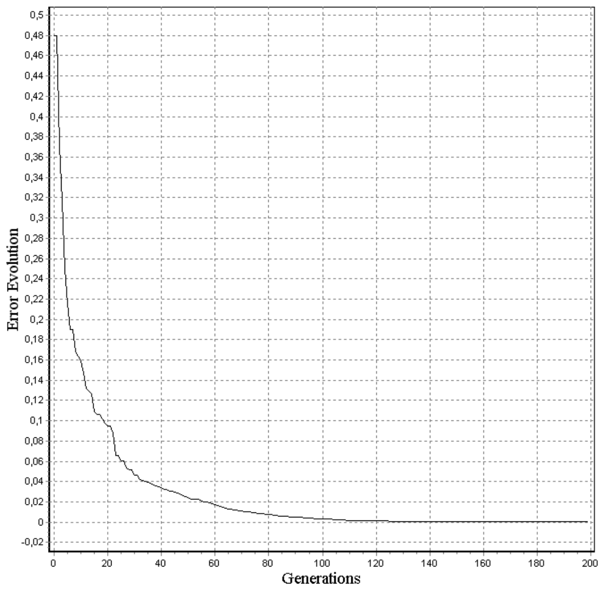


Figure 4: Error evolution over 25 runs when the set S do not agree with Property 1. Picture shows that the error decays fastly.

It is important to emphasize the fact that GAs, by their random nature, can result in different outcomes. For instance, if we pictures the evolution of the best population member for two runs, it is possible that the results are different.. However, it is expected that the result always is correct. Fortunately, Figures 3,4 show that we have achieved this goal.

6.2 Three-Dimensional Matrix

In this case, we explore not only a higher dimension problem but also the possibility of learning in face of an underconstrained three-dimensional problem; that is, we have only two input vector pars $\{(x_1, y_1), (x_2, y_2)\}$ to learn a three-dimensional operator.

The number of solutions gets larger (it is infinite). However, the prior information is incomplete. We expect some trade-off between these elements.

Before analyzing algorithm efficiency, we will consider a case for which there is only one solution. Than, the analysis of the underconstrained case will be more effective
.

The set S and the desired matrix are given by:

$$S = \left\{ \begin{array}{c} \left(\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix}, \begin{bmatrix} 4.10 \\ 0.72 \\ 2.30 \end{bmatrix} \right); \\ \left(\begin{bmatrix} 0.70 \\ 0.50 \\ 0.90 \end{bmatrix}, \begin{bmatrix} 1.43 \\ 0.30 \\ 1.01 \end{bmatrix} \right); \\ \left(\begin{bmatrix} 0.10 \\ 0.40 \\ 0.20 \end{bmatrix}, \begin{bmatrix} 0.40 \\ 0.11 \\ 0.18 \end{bmatrix} \right) \end{array} \right\}, \qquad (34)$$

and

$$A = \begin{bmatrix} 0.60 & 0.40 & 0.90 \\ 0.26 & 0.20 & 0.02 \\ 0.80 & 0.00 & 0.05 \end{bmatrix}, \qquad (35)$$

respectively.

The parameters used are those ones given by the third line of Table 1 plus a perturbation size defined by the range $[0.1, 0.2]$.

We observe that we had to increase $Ngen$ and Nm to get the algorithm learns correctly. It is possible the set of parameter values used is not the optimum one. However, remember that we aim to show stability of parameters against problem dimension. That is way we keep the parameters unchanged as much as possible.

Figure 5 shows the mean error evolution over 25 runs.

Let us consider now the learning process when we take a subset of $\tilde{S}$ of S. In this example, we choose the first two vector pairs of S. Figure 6 shows the error evolution.

The set of parameters is given by the fourth line of Table 1 plus the same perturbation size of the constrained case ($[0.1, 0.2]$).

If compared with the constrained case, we observe that we had to increase the population size but the number of generations is smaller than that one for the constrained test. As we expect, there is a trade-off between the increase of candidate solutions and the fact that we are less able to properly evolve the populations due to the lack of prior information.

6.3 Four-Dimensional Matrix

The set S and the matrix A are respectively given by:

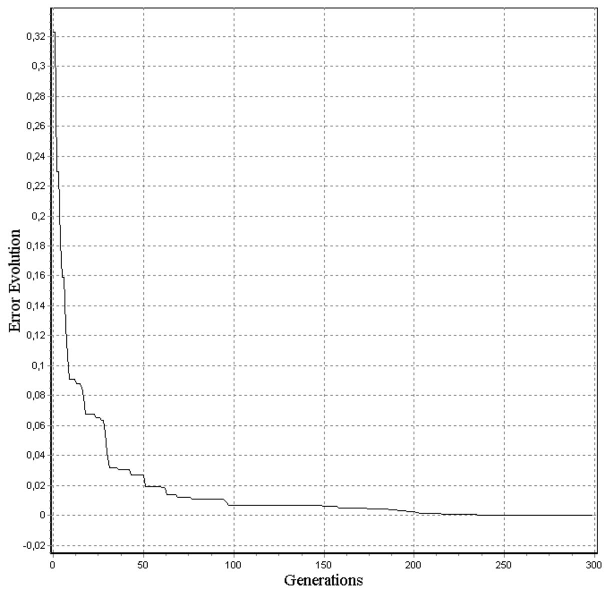


Figure 5: Error evolution over 25 runs for constrained 3D case. Like in the previous examples, this figure shows a fast decay for the error.

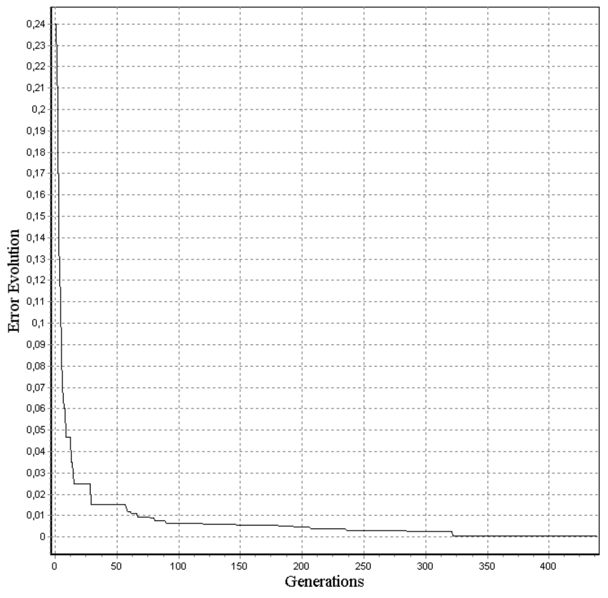


Figure 6: Error evolution over 25 runs for unconstrained 3D case. Like in the previous examples, this figure shows a fast decay for the error.

$$S = \left\{ \begin{array}{c} \left(\begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \\ 1 \\ 0 \end{bmatrix} \right); \left(\begin{bmatrix} 0 \\ 5 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right); \\ \left(\begin{bmatrix} 9 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \\ 9 \\ 0 \end{bmatrix} \right); \left(\begin{bmatrix} 2 \\ 4 \\ 6 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ 6 \\ 2 \\ 0 \end{bmatrix} \right) \end{array} \right\}; \tag{36}$$

and:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Matrix A is a permutation matrix. It permutes the input vector entries, as we can verify through the set S above. Thus, our GA algorithm has an 4×4 unitary operator to learn.

The parameters used are reported on Table 1 with a perturbation size defined by the range $[0.02, 0.05]$. This is the best result we get. Indeed, the algorithm learns correctly and the number of generations and the population size are smaller than for any other case studied. Figure 7 shows mean error evolution. It follows the same pattern of previous examples

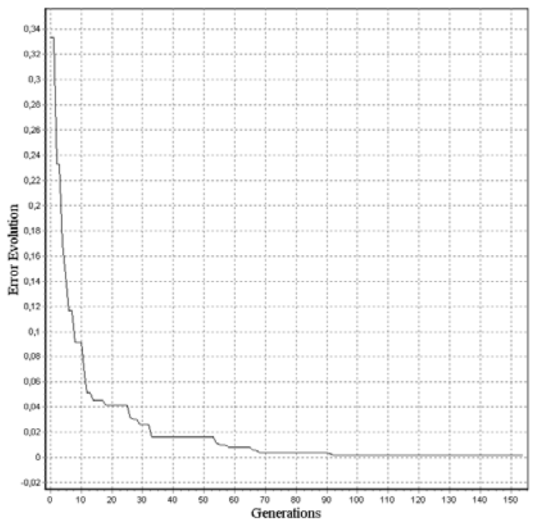


Figure 7: Error evolution over 25 runs during the learning of a 4×4 matrix.

6.4 Six Dimensional Case

The following example shows a challenge for our GA learning method. When the dimension increases parameters choice becomes a difficult task.

The set S and the target matrix are the next ones:

$$S = \left\{ \begin{array}{cc} \left(\begin{bmatrix} 1.0 \\ -1.0 \\ 1.0 \\ -1.0 \\ 1.0 \\ -1.0 \end{bmatrix}, \begin{bmatrix} 2.0 \\ -0.3 \\ -1.7 \\ -1.0 \\ 1.6 \\ -0.5 \end{bmatrix} \right); & \left(\begin{bmatrix} 0.5 \\ 0.6 \\ 0.7 \\ 0.8 \\ 0.9 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 1.27 \\ 2.8 \\ -0.54 \\ 1.04 \\ 1.76 \\ 2.27 \end{bmatrix} \right); \\ \left(\begin{bmatrix} -0.9 \\ -0.7 \\ -0.5 \\ -0.3 \\ -0.1 \\ -0.6 \end{bmatrix}, \begin{bmatrix} -0.3 \\ -1.66 \\ 0.81 \\ -0.89 \\ -1.04 \\ -1.55 \end{bmatrix} \right); & \left(\begin{bmatrix} 2.0 \\ 1.5 \\ 1.2 \\ -3.0 \\ -1.4 \\ 1.8 \end{bmatrix}, \begin{bmatrix} -3.24 \\ -0.72 \\ -3.60 \\ 1.52 \\ -1.14 \\ 0.01 \end{bmatrix} \right); \\ \left(\begin{bmatrix} 0.2 \\ 1.0 \\ 0.6 \\ 0.8 \\ 0.7 \\ 0.1 \end{bmatrix}, \begin{bmatrix} 1.45 \\ 2.81 \\ 0.24 \\ 0.88 \\ 1.24 \\ 1.75 \end{bmatrix} \right); & \left(\begin{bmatrix} -0.2 \\ 0.8 \\ -0.4 \\ 0.6 \\ -0.9 \\ 0.7 \end{bmatrix}, \begin{bmatrix} -1.34 \\ 0.23 \\ 0.96 \\ 0.80 \\ -0.88 \\ 0.41 \end{bmatrix} \right); \\ \left(\begin{bmatrix} -2.0 \\ 3.0 \\ 5.0 \\ 1.2 \\ -0.3 \\ 6.0 \end{bmatrix}, \begin{bmatrix} -4.40 \\ 8.76 \\ 3.38 \\ 10.99 \\ 1.08 \\ 6.97 \end{bmatrix} \right); & \left(\begin{bmatrix} 4.0 \\ -1.0 \\ 5.0 \\ -2.0 \\ -5.0 \\ -7.0 \end{bmatrix}, \begin{bmatrix} 3.90 \\ -5.80 \\ 5.20 \\ -3.80 \\ 0.20 \\ -10.70 \end{bmatrix} \right) \end{array} \right\};$$

$$A = \begin{bmatrix} 0.4 & 0.2 & 0.3 & 0.5 & 1.0 & -1.0 \\ 0.1 & 1.0 & 0.7 & 0.8 & 1.0 & 0.3 \\ -1.0 & -0.2 & 0.6 & 0.9 & -1.0 & -0.4 \\ -0.3 & 0.2 & 0.8 & 0.3 & -0.1 & 0.9 \\ 0.8 & -0.2 & 0.4 & 0.6 & 0.8 & 0.0 \\ 0.5 & 0.3 & 0.1 & 0.7 & 0.9 & 1.0 \end{bmatrix}.$$

This is an overconstrained problem as the number of functional points in the set $S = \{(x_i, y_i); i = 1, ..., 8\}$ is bigger than the space dimension.

We did two experiments: Firstly, we take the first six ones (constrained case). Next, we take all pair in S (overconstrained case).

Unfortunately, we were not able to find out an optimum parameters set. For both testes, the algorithm were not able to learn correctly.

The solutions obtained for the first and second testes are, respectively, given by:

$$A_1 = \begin{bmatrix} 0.4 & 0.2 & 0.3 & 0.5 & 1.0 & -1.0 \\ 0.1 & 1.0 & 0.7 & 0.8 & 1.0 & 0.3 \\ \mathbf{-0.7} & \mathbf{0.4} & \mathbf{0.1} & \mathbf{0.8} & \mathbf{-0.9} & \mathbf{-1.0} \\ -0.3 & 0.2 & 0.8 & 0.3 & -0.1 & 0.9 \\ 0.8 & -0.2 & 0.4 & 0.6 & 0.8 & 0.0 \\ 0.5 & \mathbf{0.4} & 0.1 & 0.7 & \mathbf{0.8} & \mathbf{0.8} \end{bmatrix}.;$$

$$A_2 = \begin{bmatrix} 0.4 & 0.2 & 0.3 & 0.5 & 1.0 & -1.0 \\ \mathbf{0.4} & \mathbf{0.5} & 0.7 & \mathbf{1.0} & \mathbf{0.8} & \mathbf{0.6} \\ -1.0 & \mathbf{0.0} & 0.6 & 0.9 & \mathbf{-0.9} & \mathbf{-0.5} \\ -0.3 & 0.2 & 0.8 & 0.3 & -0.1 & 0.9 \\ \mathbf{0.7} & -0.2 & \mathbf{0.5} & 0.6 & 0.8 & 0.0 \\ 0.5 & 0.3 & 0.1 & 0.7 & 0.9 & 1.0 \end{bmatrix}.$$

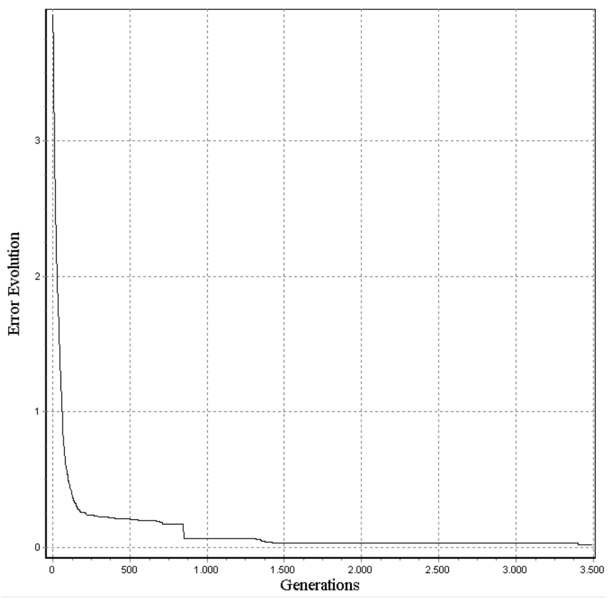
where we have written in bold the matrix elements that are different from the desired one.

The result is not far away from the desired one. The errors for A_1 and A_2 are 0.0230 and 0.0454, respectively.

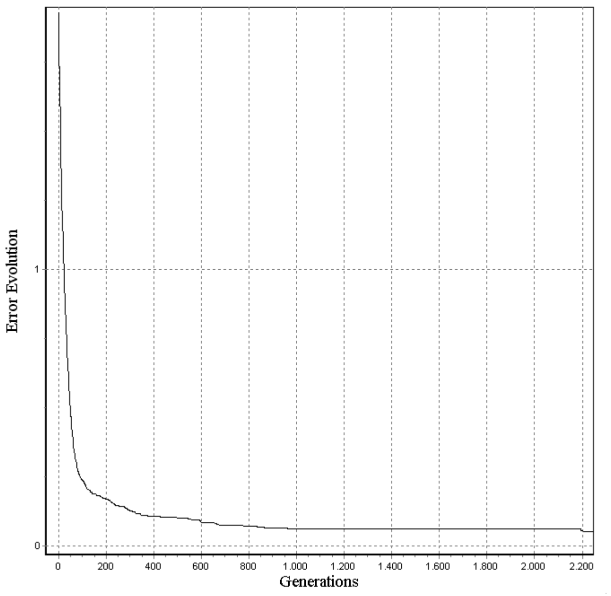
The convergence is faster for the constrained case, an unexpected result if considering that we have less prior information than for the overconstrained one. Figures 8.a and 8.b may be useful to understand this behavior. They picture the mean error evolution.

Again, it is emphasized a feature of our GA learning method. We observe that the error decays fastly but becomes almost unchanged during a long time later. The factors behind this problem may be the reasons the method fails in this case. We will discuss this point later.

The sixth and seventh lines of Table Table1 give the parameters used. Besides, we take a perturbation size given by the range $[0.1, 0.2]$.



(a)



(b)

Figure 8: Error evolution over 25 runs during the learning of a 6×6 matrix. (a) Constrained problem. (b)Overconstrained case.

7 Discussion

Table 1 shows that the associated probabilities remained unchanged for all experiments. It is a desired property because it reports to some kind of generality.

The number of generations seems to increase when space dimension gets higher. The increase rate must be controlled if we change the population size properly. However, such procedure could be a serious limitation of the algorithm for large linear systems.

Moreover, the clock time for one run is very acceptable (≤ 0.04 seconds). The behavior for underconstrained problems is also an advantage of the method, if compared with traditional ones. In this case, matrix methods (Appendix A) can not be applied without extra machinery because the solution is not unique [6].

The comparison with Dan Ventura 's learning method (section 3.1) shows that our method overcomes the limitation of the later: we do not need any of the hypothesis stated in the Property 1 of section 3.2.

However, when using our GA method, we pay a price due to storage requirements and computational complexity.

Dan Ventura's algorithm as well as numerical methods, GMRES (Appendix A), have a computational cost asymptotically limited by $O\left(n * n^2\right)$ while our GA method needs $O\left(Ngen * N * n^2\right)$ float point operations.

On the other hand, for traditional numerical matrix methods and Ventura's algorithm, we observe a storage requirements of $O\left(n^2\right)$. Thus, the disadvantage of our method becomes clear.

However, if compared with iterative methods, our algorithm is in general less sensitive to roudoff errors [6]. This is due to unlikely numerical methods, that try to follow a path linking the initial position to the optimum (for steepest descent methods, the integral solution of the problem (6)), our GA algorithm searches the solution through a set of candidates.

To improve the convergence we believe that we need better evolutionary (crossover/mutation) strategies. The behavior pictured on the Figures of section 6 for the error evolution indicates that our implementation for the genetic operators is nice to get closer the solution but not to complete the learning process. Further analysis should be made to improve these operators.

Besides, there is an important point that we must consider.

The problem we are in face is essentially a liner one for which traditional methods (Appendix A) are efficient to deal. Thus, would it be a nice idea to use GA?

The following text extracted from [12] can help the discussion about this:

”*The key point in deciding whether or not use genetic algorithms for a particular problem centers around the question: what is the space to be searched? If that space is well-understood and contains structure that can be exploited by special-purpose search techniques, the use of genetic algorithms is generally computational less efficient.*[12]”

Up to now, we shall say that our results have verified this observation.

However, in practice, traditional methods may have problems due to the sensitivity of the linear system solution against roudoff errors [6] and can not solve the underconstrained case without extra techniques. If an effective GA representation of that space can be developed then we can improve our results and our research will become more worthwhile. These are further directions for this research.

8 Conclusions

This paper reports our researches for genetic algorithms in the context of learning operators. This work was inspired in learning methods applied to quantum (unitary) operators [20].

Our GA learning method overcomes the problems found in [20]. However, we have to pay a price in terms of computational complexity and storage requirements.

We observe that our method have problems when the dimension increases because parameters choice becomes a difficult task We believe that we need a more effective GA representation of that space/problem to achieve the target efficiently. Up to now, our method is barely nice if compared with the traditional numerical ones (Appendix A).

9 Appendix A: Iterative Methods

As we already said, the proposed problem can be viewed as solution of a linear system. The simple example of section 3.1 is useful again.

We are looking for a matrix:

$$L = \left[\begin{array}{cc} a_{00} & a_{01} \\ a_{10} & a_{11} \end{array} \right], \tag{37}$$

knowing the set S, that is:

$$L \mid \chi_i \rangle = \mid \psi_i \rangle, \quad i = 0, 1. \tag{38}$$

By substituting the pairs of S (expression (8)) in this equation we find the following linear system:

$$\frac{2}{\sqrt{5}} a_{00} + \frac{1}{\sqrt{5}} a_{01} + 0 a_{10} + 0 a_{11} = \frac{3}{\sqrt{10}}, \tag{39}$$

$$0 a_{00} + 0 a_{01} + \frac{2}{\sqrt{5}} a_{10} + \frac{1}{\sqrt{5}} a_{11} = \frac{1}{\sqrt{10}}, \tag{40}$$

$$-\frac{2}{\sqrt{20}} a_{00} + \frac{4}{\sqrt{20}} a_{01} + 0 a_{10} + 0 a_{11} = \frac{2}{\sqrt{40}}, \tag{41}$$

$$0a_{00}+0a_{01}-\frac{2}{\sqrt{20}}a_{10}+\frac{4}{\sqrt{20}}a_{11}=-\frac{6}{\sqrt{40}}, \quad (42)$$

which can be represented in the following matrix form:

$$Ax=b, \quad (43)$$

where:

$$A=\begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 & 0 \\ 0 & 0 & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ -\frac{2}{\sqrt{20}} & \frac{4}{\sqrt{20}} & 0 & 0 \\ 0 & 0 & -\frac{2}{\sqrt{20}} & \frac{4}{\sqrt{20}} \end{bmatrix}; \quad (44)$$

$$x=\begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix};b=\begin{bmatrix} \frac{3}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} \\ \frac{2}{\sqrt{40}} \\ -\frac{6}{\sqrt{40}} \end{bmatrix}. \quad (45)$$

Once the input vectors $\{\mid \chi_0\rangle, \mid \chi_1\rangle\}$ are LI, the system (38) has only one solution. This property can be easily demonstrated as follows. We are going to consider the general case because it is not worthwhile to be restricted to the bi-dimensional one.

Property 2*:* If $S=\{(\mid \chi_i\rangle, \mid \psi_i\rangle)\,; \quad i=1,...,n\}$ is such that the set of input vectors $\{\mid \chi_i\rangle; \quad i=1,...,n\}$ *is an orthonormal basis of a n-dimensional Hilber space* $\cancel{H}$*,* then *the linear system given by the equations* $L\mid \chi_i\rangle=\mid \psi_i\rangle, \quad i=1,2,...,n$, has only one solution for arbitrary output vectors $\mid \psi_i\rangle$.

Demonstration. Let us rewrite the linear system in the equivalent form:

$$L\left(\mid \chi_1\rangle, \mid \chi_2\rangle, ..., \mid \chi_n\rangle\right)=\left(\mid \psi_1\rangle, \mid \psi_2\rangle, ..., \mid \psi_n\rangle\right).$$

Thus, taking the transpose for both sides of this equation we have:

$$\left(\mid \chi_1\rangle, \mid \chi_2\rangle, ..., \mid \chi_n\rangle\right)^T L^T=\left(\mid \psi_1\rangle, \mid \psi_2\rangle, ..., \mid \psi_n\rangle\right)^T.$$

This is the linear system to be solved, considering that the unknown variables are the elements of the matrix L. From the standard results in linear algebra ([6]), if $\{\mid \chi_i\rangle; \quad i=1,...,n\}$ is LI, the system has only one solution $\boxdot$.

For a more general set S, we can return to section and rewrite this problem as an optimization one given by:

$$\min_{A\in\Re^{n\times n}} \quad J(A), \quad (46)$$

where the functional $J(A)$ is defined by expression (2):

$$J(A)=\sum_{i=1}^{K}\|Ax_i-y_i\|^2=\sum_{i=1}^{K}(Ax_i-y_i)^T\cdot(Ax_i-y_i). \quad (47)$$

This is a least squares problem [6]. By expanding expression (47) it is straightforward to show that the problem (46) is equivalent to the following one:

$$\min_{A\in\Re^{n\times n}} \quad \sum_{i=1}^{K}\left(x_i^TA^TAx_i-x_i^TA^Ty_i-y_i^TAx_i\right),$$

which in turn can be rewritten using the tensor product [10]:

$$\min_{A\in\Re^{n\times n}} \quad \sum_{i=1}^{K}\left(x_i^T\otimes x_i^T[A^TA]-2x_i^T\otimes y_i^T[A^T]\right).$$

where $[\cdot]$ means the vector whose elements are the coefficients of the matrix between the brackets sorted according to matrix rows.

Thus, like in expression (3), we have to find out a matrix A, such that

$$\nabla\left[\sum_{i=1}^{K}\left(x_i^T\otimes x_i^T[A^TA]-2x_i^T\otimes y_i^T[A^T]\right)\right]=0. \quad (48)$$

The so obtained linear system can be solved by direct or iterative methods [6]. It will be equivalent to the linear system (43) if S is given by (8).

By Direct Methods we mean linear equation solvers that require the factorization of the matrix A. Gaussian elimination, LU decomposition, are very known examples of that class of methods. The computational cost of these methods is $O\left(n^3\right)$ where n is the dimension of the vector space.

However, we must observe that A in expression (43) has some sparsity as $n^2/2$ entries are zero. It is simple to conclude that this property is true for $n\geq 2$. The same is true for the linear system corresponding to the problem (48).

It is important to use methods which takes in account this property. That is way we should consider iterative methods.

These methods generate a sequence of approximate solutions $\left\{x^k\right\}$ and do not involve factorization of matrix A [6]. The advantage of these methods is to preserve the sparsity.

The following theorem is central in this field. It's demonstration is simple and is included for completeness (see [6] for more details).

Theorem: Suppose $b\in\Re^n$ and $A=M-N\in\Re^{n\times n}$ a nonsingular matrix. If M is nonsingular and the spectral radius of $M^{-1}N$ satisfies $\rho\left(M^{-1}N\right) < 1$, then the iterates x^k defined by $Mx^{k+1} = Nx^k + b$ converge to $x = A^{-1}b$ for any starting vector x^0.

Demonstration: Let $e^k = x^k - x$ denotes the error in the k-th iteration.. The fact that $Mx = Nx + b$ together with the recursive relation implies that :

$$\begin{aligned} Mx^{k+1} &= Nx^k + b \\ Mx &= Nx + b \end{aligned}$$

Subtraction the second equation from the first one gives: $M\left(x^{k+1} - x\right) = N\left(x^{k+1} - x\right)$. Thus:

$$e^{k+1} = M^{-1}Ne^k = M^{-1}N\left(M^{-1}Ne^{k-1}\right) = .. =$$

$$= \left(M^{-1}N\right)^k e^0.$$

Since we are considering $\rho\left(M^{-1}N\right) < 1$ it follows that $\left(M^{-1}N\right)^k \to 0$ if $k \to +\infty$ $\boxdot$.

In the area of iterative methods the development of algorithms typically follows the following steps:

(1) A splitting $A = M - N$ is proposed such that the iteration matrix $G = M^{-1}N$ satisfies $\rho\left(M^{-1}N\right) < 1$; (2) Further results about $\rho(G)$ are established to gain intuition about how the error e^k tends to zero.

Performing these steps have also some computational cost which depends from the specific problem. Besides the convergence rate is problem dependent.

A simpler way to avoid these problems can be designed when the matrix A is symmetric and positive definite (all eigenvalues are non-null and positive). In this case, the above theorem can be used to show that the iteration scheme (Gauss-Seidel iteration):

```
For $k = 0, 1, 2...$
For $i = 0, ..., n-1$
```

$$x_i^{k+1} = \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n-1} a_{ij}x_j^k\right)}{a_{ii}}, \quad (49)$$

converges for any x^0 [6].

These methods need special structures for matrix A. If $A \in \Re^{n\times n}$ is nonsingular then A^TA is also nonsinglar, symmetric and positive definite. However, the linear system $A^TAx = A^Tb$ may becomes ill-conditioned and thus numerical instabilities take place [6].

Another problem is that we do not have a bound for the number of iterations.. Gradient Conjugated methods overcomes this problem, but still needs A to be symmetric and positive defined.

An alternative approach is the Generalized Minimal Residual (GMRES) algorithm described next.

9.1 GMRES

Let $\|\cdot\|$ and $(\cdot,\cdot)$ denote 2-norm and standard inner product.

Consider an approximate solution of the form $x_0 + z$, where x_0 is an initial guess and z is a member of the Krylov space $K = span\left\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\right\}$, in which $r_0 = b - Ax_0$, and k is the dimension of K. GMRES algorithm determines z such that the 2-norm of the residual $\|b - A(x_0 + z)\|$ is minimized [17]. To achieve this goal, let us firstly consider the procedure bellow.

Modified Gram-Schmidt procedure.

```
$u_1 = \frac{r_0}{\|r_0\|}$,
For $i = 1, ..., k$
  $\tilde{u}_{i+1} = Au_i$,
  For $j = 1, ..., i$
    $\beta_{i+1} = \left(\tilde{u}_{i+1}, u_i\right)$,
    $\tilde{u}_{i+1} \longleftarrow \tilde{u}_{i+1} - \beta_{i+1}u_j$,
  $u_{i+1} = \frac{\tilde{u}_{i+1}}{\left\|\tilde{u}_{i+1}\right\|}$.
```

Let $U_k = [u_1, u_2, ..., u_k]$. Then it can be shown that:

$$AU_k = U_{k+1}H_k,$$

where H_k is the following $(k+1) \times k$ upper Hessenberg matrix:

$$H_k = \begin{bmatrix} \beta_{2,1} & \beta_{3,1} & ... & \beta_{k,1} & \beta_{k+1,1} \\ \left\|\tilde{u}_2\right\| & \beta_{3,2} & ... & \beta_{k,2} & \beta_{k+1,2} \\ 0 & \left\|\tilde{u}_3\right\| & ... & \beta_{k,3} & \beta_{k+1,3} \\ ... & ... & ... & ... & ... \\ 0 & 0 & ... & \left\|\tilde{u}_{k+1}\right\| & \beta_{k+1,k} \\ 0 & 0 & ... & 0 & \left\|\tilde{u}_{k+1}\right\| \end{bmatrix}$$

Let $z = \sum_{j=1}^{k} y_j u_j$ and $e = \{\|r_0\|, 0, ..., 0\}^T$ where e has $k+1$ entries. Note that $r_0 = U_{k+1}e$. Thus:

$$\|b - A(x_0 + z)\| = \left\|r_0 - A\left(\sum_{j=1}^{k} y_j u_j\right)\right\| =$$

$$\|r_0 - AU_ky\| = \|U_{k+1}(e - H_ky)\| = \|e - H_ky\|.$$

Thus, the minimization problem can be written as:

$$\min_{z \in K} \|b - A(x_0 + z)\| = \min_{y \in \Re^k} \|e - H_ky\|.$$

The minimization problem can be solved efficiently by observing that H_k is almost triangular. Therefore, the Q-R algorithm is explored to obtain the minimizer [6]. The key idea of the Q-R algorithm is to obtain an orthogonal matrix:

$$R = R_k R_{k-1} ... R_1,$$

where each $R_j, j = 1, 2, ..., k$ is a $(k+1) \times (k+1)$ Givens rotation of the form:

$$R_j = \begin{bmatrix} I_{j-1} & & \\ & \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} & \\ & & I_{k-j} \end{bmatrix},$$

in which I_i is the identity matrix of dimension i, and c_j and s_j satisfies $c_j + s_j = 1$ and are constructed such that the matrix:

$$\tilde{H} = R H_k,$$

is upper triangular [6]. Let $\tilde{e} = R.e$. Then:

$$\|e - H_k y\| = \left\| R^T \left(\tilde{e} - \tilde{H}\, y \right) \right\| = \left\| \tilde{e} - \tilde{H}\, y \right\|$$

Consequently:

$$\min_{y \in \Re^k} \left\| \tilde{e} - \tilde{H}\, y \right\| = \left| \tilde{e}_{k+1} \right|,$$

and y satisfies:

$$\begin{bmatrix} \tilde{H}_{1,1} & \tilde{H}_{1,1} & \cdots & \tilde{H}_{1,k-1} & \tilde{H}_{1,k} \\ 0 & \tilde{H}_{2,2} & \cdots & \tilde{H}_{2,k-1} & \tilde{H}_{2,k-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \tilde{H}_{k-1,k-1} & \tilde{H}_{k-1,k} \\ 0 & 0 & \cdots & 0 & \tilde{H}_{k,k} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_{k-1} \\ y_k \end{bmatrix} = \tag{50}$$

$$= \begin{bmatrix} \tilde{e}_1 \\ \tilde{e}_2 \\ \cdots \\ \tilde{e}_{k-1} \\ \tilde{e}_k \end{bmatrix}$$

which is simply solved by back-substitution.

The main loop of the GMRES method can be summarized as follows: (1) Calculation of an orthonormal basis for the Krylov space by the modified Gram-Schmidt procedure; (2) Triangulation of the Hessenberg matrix by the Q-R algorithm; (3) Back-substitution to solve (50); (4) Evaluate the error $\left| \tilde{e}_{k+1} \right|$.

10 Acknowledgments

We would like to acknowledge CNPq for the financial support for this work.

References

[1] C. Adamis. *Artificial Life*. Springer-Verlag New York, Inc., 1998.

[2] R. Beale and T. Jackson. *Neural Computing*. MIT Press, 1994.

[3] C. Chapra and R.P. Canale. *Numerical Methods for Engineers*. MacGraw-Hill International Editions, 1988.

[4] C. Cohen-Tannoudji, B. Diu, and F. Laloe. *Quantum Mechanics*, volume I. Wiley, New York, 1977, 1977.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[6] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1985.

[7] J. J. Grefenstette, editor. *Genetic Algorithms for Machine Learning*. Kluwer Academic Publishers, 1994.

[8] K. Hoffman and R. Kunze. *Linear Algebra*. Prentice-Hall, 1961.

[9] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 1975.

[10] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc., 1989.

[11] K.A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, 1975.

[12] K.A. De Jong. Introduction to the second special issue on genetic algorithms. *Machine Learning*, 5(4):351–353, 1990.

[13] Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, 1996.

[14] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press., December 2000.

[15] M. Oskin, F. Chong, and I. Chuang. A practical architecture for reliable quantum computers. *IEEE Computer*, 35(1):79–87, 2002.

[16] J. Preskill. Quantum computation - caltech course notes. Technical report, 2001.

[17] F. Shakib, T. J. Hughes, and Z. Johan. A multi-element group preconditioned gmres algorithm for nonsymmetric systems arising in finite element analysis. *Computer Methods in Applied Mechanics and Eng.*, 75:415–456, 1989.

[18] J. Sotomayor. *Lições de Equações Diferenciais Ordinárias*. Projeto Euclides, Gráfica Editora Hamburgo Ltda, São Paulo, 1979.

[19] Ya. Z. Tsypkin and Z. J. Nikolic. *Foundations of the Theory of Learning Systems*. Academic Press, New York and London, 1973.

[20] Dan Ventura. Learning quantum operators. In *Proceedings of the International Conference on Computational Intelligence and Neuroscience*, pages 750–752, March 2000.

[21] Alden H. Wright. Genetic algorithms for real parameter optimization. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA, 1991.