

# AN IMPROVED ITERATIVE METHOD FOR SOLVING GENERAL SYSTEM OF EQUATIONS VIA GENETIC ALGORITHMS

Seyed Abolfazl Shahzadehfazeli<sup>1,2</sup> Zainab Haji Abootorabi<sup>2,3</sup>

<sup>1</sup>Parallel Processing Laboratory, Yazd University, Yazd, Iran

<sup>2</sup>Department of Computer Science, Faculty of Mathematics, Yazd University, Yazd, Iran

<sup>3</sup>Department of Mathematics, PNU University, Yazd, Iran

## ABSTRACT

*Various algorithms are known for solving linear system of equations. Iteration methods for solving the large sparse linear systems are recommended. But in the case of general  $n \times m$  matrices the classic iterative algorithms are not applicable except for a few cases. The algorithm presented here is based on the minimization of residual of solution and has some genetic characteristics which require using Genetic Algorithms. Therefore, this algorithm is best applicable for construction of parallel algorithms. In this paper, we describe a sequential version of proposed algorithm and present its theoretical analysis. Moreover we show some numerical results of the sequential algorithm and supply an improved algorithm and compare the two algorithms.*

## Keywords

*Large sparse linear systems, Iterative Genetic algorithms, Parallel algorithm.*

## 1. INTRODUCTION

Let  $A$  be a general  $n \times m$  matrix. The main problem is to solve the linear system of equations:

$$Ax = b \quad (1)$$

where  $x \in R^m$  and  $b \in R^n$  are the solution and the given right hand side vectors. We can determine from matrix  $A$  and the vector  $b$ , the existence and uniqueness of the solution of (1). Theoretically the Gaussian or Gauss-Jordan elimination algorithm is an appropriate tool to solve the system (1) and to decide the question of solvability. when we use floating point arithmetic for large systems, these direct algorithms are inapplicable. For these cases the iterative algorithms are suitable. Effective iterative algorithms are known for symmetric positive definite linear systems. In general, iterative algorithms can be written in the form of:

$$x(n) = Bx(n-1) + d, \quad n=1, 2, \dots \quad (2)$$

where  $B$  and  $d$  are such a matrix and vector that make stationary solution of (2) equivalent with (1), see ([1]). These iterative algorithms can be applied for general non symmetric linear systems as well, if we solve the following normal system:

$$A^T Ax = A^T b = v \quad (3)$$

instead of the original one. A disadvantage of this approach is that the resulting linear system (3) for matrices with full rank will be Hermitian ones, however, its condition number will be the square of the original condition number. Therefore, the convergence will be very slow. For general linear systems when A is non-Hermitian, instead of using some variant of the Conjugate Gradient (CG) algorithms, one of the most successful schemes is the generalized minimal residual algorithm (GMRES), see ([9, 10]) and the biconjugate gradient algorithm (BCG) see ([2]).

A more effective approach was suggested by Freund and Nachtigal ([5]) for the case of general nonsingular non-Hermitian systems which is called the quasi minimal residual algorithm (QMR). An iterative minimal residual algorithm which is slightly different from the above ones uses Genetic Algorithms (GA), see ([4, 6,7, 8]).

In the following, we describe an improved method using genetic algorithms, in which, the initial population is larger, uses a broader search field and its crossover operator on initial population enhances the algorithm convergence speed. Generally, genetic algorithm with larger search space, does not guarantee the convergence speed see ([3]).

In this paper, it is shown that our improved method is in practice much faster than previous types. This advantage can be very important for development of these algorithms for parallel processing. The result obtained in [8] is briefly reviewed here to clarify the improved algorithm.

## 2. AN ITERATIVE MINIMAL RESIDUAL ALGORITHM

The most of iterative algorithms for solving linear systems are based on some minimization algorithm. We can obtain the normal system (3) in the following way by the least square minimization. We have to solve the following problem:

$$\min_{x \in R^n} \|Ax - b\|_2^2 = \min_{x \in R^n} (Ax - b, Ax - b) = \min_{x \in R^n} (r, r) = \min_{r \in R^m} \|r\|_2^2 \quad (4)$$

where  $r=Ax-b$  is the residual of the vector  $x$ .

It is easy to show that the equation (4) can be written as in (3). More precisely, the necessary condition for the existence and uniqueness of the solution of (4) is obtained for the fulfillment of (3). The Hermitian property of the normal matrix  $A^T A$  is a sufficient condition for the uniqueness. For general non-Hermitian matrices this condition is not fulfilled in general. One possible algorithm to solve the problem (4) can be obtained from the following theorem.

**Theorem 1.** Let  $A \in R^m \rightarrow R^n$  and  $b \in R^n$  be arbitrary matrix and vector. Moreover, let  $x^\alpha \in R^n$  and  $x^\beta \in R^n$  be arbitrary different vectors for which  $A(x^\alpha - x^\beta) \neq 0$ .

Let us introduce the following notations:

$$r^s = Ax^s - b, \quad S = \alpha, \beta$$

and

$$X^{\alpha,\beta} = cx^\alpha + (1-c)x^\beta, \quad r^{\alpha,\beta} = cr^\alpha + (1-c)r^\beta$$

where  $c \in R$ . We have  $Ax^{\alpha,\beta} - b = r^{\alpha,\beta}$ . Then, the solution of the minimization problem of (4) is the vector  $x^{\alpha,\beta}$  with  $c$ , where

$$c = \frac{(r^\beta - r^\alpha, r^\beta)}{\|r^\alpha - r^\beta\|_2^2}$$

Moreover,

$$\|r^{\alpha,\beta}\|_2 \langle \min_{\alpha,\beta} \{ \|r^\alpha\|_2, \|r^\beta\|_2 \} \rangle.$$

---

### The Algorithm 1

---

From Theorem 1 we obtain an algorithm (see [8]), which generates an approximate solution sequence  $x^k$ ,  $k=1, 2, 3, \dots$  with residual vectors  $r^k$ ,  $k=1, 2, 3, \dots$

- 1) Let  $x^1$  be an arbitrary vector and  $\epsilon$  the tolerance.
  - 2) Calculate  $r^1 = Ax^1 - b$ .
  - 3) Generate an arbitrary vector,  $x^2$  such that  $r^1 - r^2 \neq 0$ .
  - 4) Calculate the  $c^{1,2}$ .
  - 5) Calculate the new  $x^{1,2} := c^{1,2} x^1 + (1 - c^{1,2}) x^2$  and  $r^{1,2} := c^{1,2} r^1 + (1 - c^{1,2}) r^2$  vectors.
  - 6)  $x^1 := x^{1,2}$  and  $r^1 := r^{1,2}$ .
  - 7) If  $r^1 < \epsilon$  then go to 8, else go to 3.
  - 8) The approximate solution is  $x^1$ .
  - 9) End of algorithm.
- 

The simplest algorithm which can be obtained from Theorem 1 is the algorithm 1. Therefore, this algorithm does not converge faster than the classical ones.

### 3. THE IMPROVED ALGORITHM USING GA

Genetic algorithms (GAs) were proposed first time by John Holland and were developed by Holland and his colleagues at the University of Michigan in the 1960s and the 1970s. On continuous and discrete combinatorial problems, GAs work very well. But they tend to be computationally expensive. GAs are examples of algorithms that are used in this field and have improved tremendously in the past two decades. A genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems. (GA)s are in the class of global search heuristics. (GA)s are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, selection, crossover and mutation.

**Selection:** Choice of individual genomes from a population for using the crossover operator is the stage of a genetic algorithm which is called Selection. There are many ways how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

**Crossover:** After we have decided what encoding we will use, we can make a step to crossover. Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent. There are many methods how to do crossover. For example Single point crossover, Two point crossover, Uniform crossover and Arithmetic crossover.

**Mutation:** After a crossover is performed, mutation takes place. This is to prevent falling all solutions in population into a local optimum of solved problem. Mutation changes randomly the new offspring. As well as the crossover, the mutation depends on the encoding. For example mutation could be exchanging two genes, when we are encoding permutations. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1.

The most important parts of the genetic algorithm are the crossover and mutation. The performance is influenced mainly by these two operators. Crossover and mutation are two basic operators of GA and performance of GA is very dependent on them. Implementation and type of operators depends on a given problem and encoding.

The evolution usually starts from a population of randomly generated individuals and happens in generations. The fitness of every individual in the population, evaluate in each generation and select multiple individuals from the current population and modify to form a new population. In the next iteration of the algorithm use the new population. The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

---

### The Basic Genetic Algorithm

---

- 1) Generate random population of n chromosomes.
  - 2) Evaluate the fitness function of each chromosome x in the population.
  - 3) Create a new population by repeating following steps until the new population is complete.
    - a) **Selection:** Select two parent chromosomes from a population according to their fitness.
    - b) **Crossover:** With a crossover probability crossover the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    - c) **Mutation:** With a mutation probability mutate new offspring at each locus.
    - d) Place new offspring in a new population.
  - 4) Use new generated population for a further run of algorithm.
  - 5) If the end condition is satisfied, stop, and return the best solution in current population.
  - 6) Go to step 2
- 

The three most important aspects of using genetic algorithms are:

- 1) Definition of the objective function.
- 2) Definition and implementation of the genetic representation.
- 3) Definition and implementation of the genetic operators. Once these three have been defined, the generic algorithm should work fairly well.

In algorithm 1, we choose  $x^1$  and  $x^2$  arbitrarily, then use crossover operator to reach an optimal  $x^{1,2}$  and replace it for  $x^1$ . Then, we randomly select  $x^2$  again. Finally this process is continued until a fairly accurate approximation to the answer is achieved for linear equations  $Ax=b$ . But in the improved algorithm, instead of  $x^1$  and  $x^2$  and instead of the original population from two-parent, m-parent is chosen. (Note in the allocate names,  $x^1, x^2, \dots, x^m$ , m is the number of columns of matrix A).

The crossover operator performed on the initial population generates vectors,  $x^{1,2}, \dots, x^{m-1,m}$ . This process is repeatedly performed on the newly generated vectors until a single vector  $x^{1,2,3,\dots,m}$  as

an approximate initial solution is obtained. This is now replaced by  $x^1$  also we randomly select  $x^2, \dots, x^m$  again for second population and the algorithm is repeated again and again until a close solution is obtained. The following table shows how the new vectors are generated. For detail refer to the algorithm 2.

$x^1$				
$x^2$	$x^{1,2}$			
$x^3$	$x^{2,3}$	$x^{1,2,3}$		
$\dots$	$\dots$	$x^{2,3,4}$		
$x^m$	$x^{m-1,m}$	$\dots$	$\dots$	$x^{1,2,3,\dots,m}$

Now the algorithm 1 is improved in order to increase the convergence speed.

---

**The Algorithm 2**

---

- 1) Let  $x^1$  be an arbitrary vector and  $\epsilon$  the error tolerance and  $i=1$ .
- 2) Calculate  $r^1 = Ax^1 - b$ .
- 3) Generate an arbitrary vector  $x^2, \dots, x^m$  such that  $r^i - r^j \neq 0, (i \neq j) \text{ } \forall i, j = 1, \dots, m$ .
- 4) Calculate the

$$C_k = \frac{(r^{k+1} - r^k, r^{k+1})}{\|r^{k+1} - r^k\|_2^2}, \text{ for } k = i, 2, \dots, m - 1 .$$

- 5) Calculate the new  $x^{k,k+1} = C_k x^k + (1 - C_k) x^{k+1}$  and  $r^{k,k+1} = C_k r^k + (1 - C_k) r^{k+1}$  vectors, for  $k = i, \dots, m - 1$ .
  - 6)  $x^{k+1} = x^{k,k+1}$ , for  $k = i, \dots, m - 1$ , and  $i = i + 1$ .
  - 7) If  $i = m - 1$ , then  $x^1 = x^{m-1,m}$  and  $r^1 = r^{m-1,m}$  else go to 4.
  - 8) If  $\|r^1\|_2 < \epsilon$  then go to 9, else go to 3.
  - 9) The approximate solution is  $x^1$ .
  - 10) End of algorithm.
- 

**4. NUMERICAL EXPERIMENTS**

In this section, we compare algorithm 1 and algorithm 2. Also, we use the different examples and review speed of the algorithm and we show some examples in the summary table and an example to bring more detail.

In the examples, the condition number of the matrices  $A$  are chosen rather small (The coefficient matrices are well-conditioned).

The following table (table1) compares the number of iterations by the two algorithms. Figure 1 shows that for matrix A1 with the condition number 80.903 and spectral radius 15.7009, the algorithm 1 converges after 136720 iterations while the number of iterations in the improved algorithm (algorithm 2) is 16129. This is a notable reduction.

Table1. The number of iterations.

Matrix	Dim.	Tol.	No. of iter. algorithm 1	No. of iter. algorithm 2
A <sub>1</sub>	15×20	10 <sup>-3</sup>	136720	16129
A <sub>2</sub>	20×15	10 <sup>-3</sup>	10691	1812
A <sub>3</sub>	20×25	10 <sup>-3</sup>	52273	8285
A <sub>4</sub>	25×20	10 <sup>-3</sup>	665	279
A <sub>5</sub>	25×30	10 <sup>-3</sup>	119041	22920
A <sub>6</sub>	30×25	10 <sup>-3</sup>	805	349
A <sub>7</sub>	35×30	10 <sup>-3</sup>	1228	436
A <sub>8</sub>	40×35	10 <sup>-3</sup>	1390	500

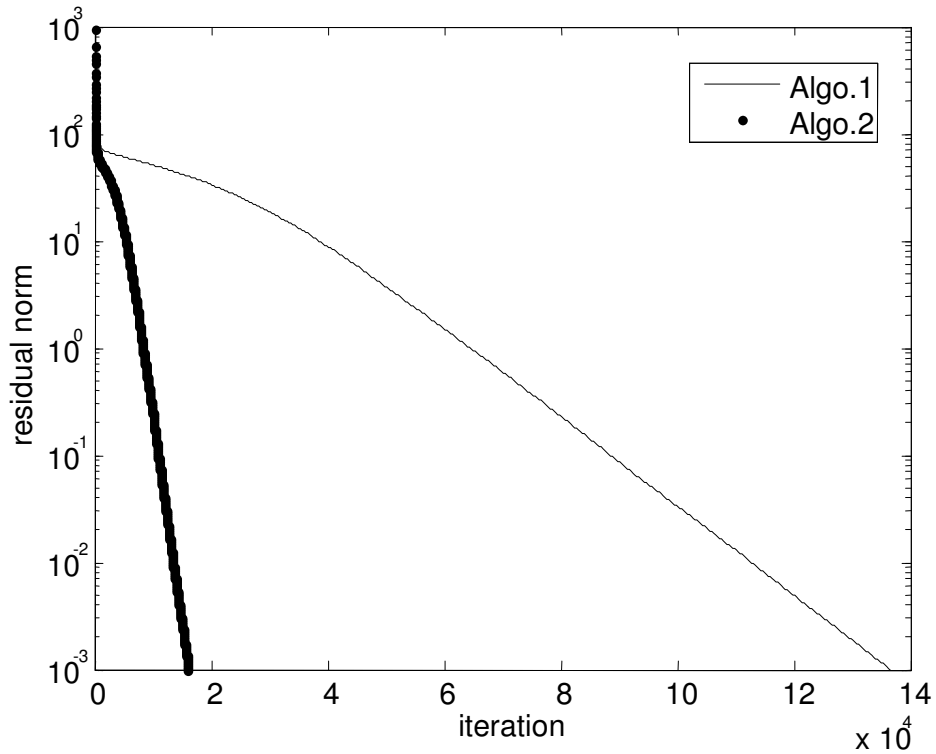


Figure 1. Speed of convergence of the Algorithm 1 and Algorithm 2 on the A1 matrix.

## 5. CONCLUSION

In this paper, for solving systems of linear equations an improved algorithm is presented. In contradiction to other iterative methods (Jacobi, Gauss-Seidel, conjugate gradient and even Gauss-elimination methods), this method has not any limitations.

Genetic algorithm enhances an appropriate response to eliminate restrictions and is a simple method for obtaining the solution. As the examples show, the number of iterations in algorithm 2 is incredibly reduced. The merit of the algorithm is its simplicity to use specially for non-square systems and to extend to large systems of equations by incorporating parallel computing.

## REFERENCES

- [1] Hageman L. A. & Joung D. M., (1981) Applied Iterative Methods, Computer Science and Applied Mathematics, Academic Press.
- [2] Hestenes M.R. & Stiefel, E. (1954) Methods of conjugate gradients for solving linear systems; J. Res. Natl. Bur. Stand. 49, 409- 436, .
- [3] Hoppe T., (2006) Optimization of Genetic Algorithms, Drexel University, Research Paper.
- [4] Koza J. R., Bennett H. B., Andre D., & Keane M. A., (1999) Genetic programming III: Drawinian Invention and Problem Solving, Morgan Kaufmann Publishers.
- [5] Lanczos C., (1952) Solution of systems of linear equations by minimized iterations, J Res. Nat. Bur. Standards, 49, 33-53.
- [6] Michalewicz & Zbeigniew, (1996) Genetic algorithms + Data Structures = Evolution Program, Springer – Verlag, Thirst edition.
- [7] Mitchell & Melanie, (1996) An Introduction to Genetic Algorithms, Cambridge, MA: The MIT Press.
- [8] Molnárka G. & Miletic, (2004) A Genetic Algorithm for Solving General System of Equations, Department of Mathematics, Széchenyi István University, Győr, Hungary.
- [9] Molnárka G. & Török B. (1996) Residual Elimination Algorithm for Solving Linear Equations and Application to Sparse Systems, Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM), Issue 1, Numerical Analysis, Scientific Computing, Computer Science, 485-486.
- [10] Saad Y. & Schultz M. H. (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput., 7, 856-869.