

**STUDY OF HYBRID EVOLUTIONARY COMPUTATION  
TECHNIQUES FOR SOLVING LARGE SET OF LINEAR  
EQUATIONS AND PARTIAL DIFFERENTIAL EQUATIONS**

By

**G. M. Moniruzzaman**

Roll No. 0451505



A thesis submitted for the partial fulfillment of the requirements for the degree of  
Master of Philosophy in Mathematics



Khulna University of Engineering & Technology  
Khulna 920300, Bangladesh.

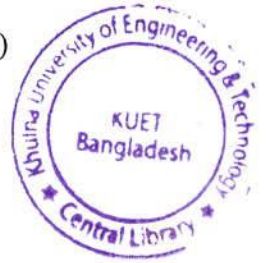
**October 2010**

# Approval

This is to certify that the thesis work submitted by G. M. Moniruzzaman entitled "*Study of Hybrid Evolutionary Computation Techniques for Solving Large Set of Linear Equations and Partial Differential Equations*" has been approved by the Board of Examiners for the partial fulfillment of the requirements for the degree of Master of Philosophy in the Department of Mathematics, Khulna University of Engineering & Technology, Khulna, Bangladesh in October, 2010.

## BOARD OF EXAMINERS

1. Arif 30.10.10  
Dr. Mohammad Arif Hossain  
Professor  
Department of Mathematics  
Khulna University of Engineering & Technology  
Bangladesh.  
Chairman  
(Supervisor)
2. Arif 30/10/10  
Head  
Department of Mathematics  
Khulna University of Engineering & Technology  
Bangladesh.  
Member
3. Arif  
Dr. Md. Bazlar Rahamn  
Professor  
Department of Mathematics  
Khulna University of Engineering & Technology  
Bangladesh.  
Member
4. Arif 30/10/10  
Dr. Md. Abul Kalam Azad  
Professor  
Department of Mathematics  
Khulna University of Engineering & Technology  
Bangladesh.  
Member
5. Arif Khan  
Dr. Nurul Alam Khan 30-10-10  
Professor  
Department of Mathematics  
Jahangirnagar University  
Savar, Dhaka  
Bangladesh.  
Member (External)



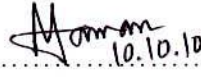
## Declaration

---

This is to certify that the thesis work entitled "Study of Hybrid Evolutionary Computation Techniques for Solving Large Set of Linear Equations and Partial Differential Equations" has been carried out by G. M. Moniruzzaman in the Department of Mathematics, Khulna University of Engineering & Technology, Khulna, Bangladesh. The above thesis work or any part of this work has not been submitted anywhere for the award of any degree or diploma.



Mohammad Arif Hossain



G. M. Moniruzzaman

## Dedication

---

To my respectable Father and late Mother whose constant guidance and inspiration helped me to choose the correct path of life.

&

To my beloved wife, affectionate two daughters who directly and indirectly inspire me for doing research works.

## Acknowledgements

---

I wish to express my profound gratitude to my supervisor Dr. Mohammad Arif Hossain, Professor, Department of Mathematics, Khulna University of Engineering & Technology, for his constant *guidance* and encouragement during my research work and for his valuable suggestions, criticism and guidance throughout all phases of the research work. Dr. Hossain has a lot of research experience in this area. He has been a great source of ideas, knowledge and feedback for me.

I heartily express my gratefulness to Dr. Md. Bazlar Rahman, Professor, Department of Mathematics, Khulna University of Engineering & Technology. His valuable suggestion throughout the entire period of research work helps me to complete my thesis.

I heartily express my gratefulness to Dr. A. R. M. Jalal Uddin Jamali, Associate Professor, Department of Mathematics, Pintu Chandra Shill, Assistant Professor, Department of Computer Science and Engineering who directly help me to develop Algorithm Design. Dr. Jamali has a research experience in this area. He has been a great source of ideas, knowledge and inspiration for my research work.

Md. Mizanur Rahman, Junior system Engineer, Md. Moududur Rahman Shamim, 4<sup>th</sup> Year Student, bearing roll No.- 0607035, Department of Computer Science and Engineering, Khulna University of Engineering & Technology, also help me in clinical or technical works and algorithm development. I would like to thank Md. Zahidul Islam, Head Assistant, Department of Electronics & Communications Engineering, for his timely helps during my research period. I am also thankful to all members of the Department of Mathematics for their assistance during my research work.

I am obliged to express my heartiest thanks to my wife, Khadiza Akter, for her constant inspiration and encouragement. Finally, I would like to express my sincere thanks to my two daughters, Jannat Zaman and Tasneem Zaman, for their sacrifice of affection and love what they should deserve, during my research period. I also like to offer greatest thanks to my father-in-law, Khondokar Md. Abu Taher.

## List of Publications

---

1. Jamali, A R M J U, M. Arif Hossain, G. M. Moniruzzaman and M. M. A. Hashem (in press), “*Crossover – a Redundant Operator for Solving System of Linear Equations by Uniform Adaptive Hybrid Evolutionary Algorithms*”, Journal of the Bangladesh Mathematical Society, Dhaka University.
2. Jamali, A R M J U, Mohammad Arif Hossain, G. M. Moniruzzaman and M. M. A. Hashem, “*For Solving Linear Equations Recombination is a Needless Operation in Time-Variant Adaptive Hybrid Algorithms*”, The 8th International Conference on Computer and Information Technology (ICCIT-05), Dhaka, Bangladesh, 28 – 30 December, 2005.
3. Jamali, A R M J U, M. Arif Hossain, G. M. Moniruzzaman and M. M. A. Hashem, “*For Solving Linear Equations by Uniform Adaptive Hybrid Evolutionary Algorithms – Crossover – a Superfluous Operator*”, The 16th Mathematics Conference of Bangladesh Mathematical Society, Dhaka, Bangladesh, 17-19 December, 2009,

# Contents

	Pages
Approval. . . . .	i
Declaration. . . . .	ii
Dedication. . . . .	iii
Acknowledgement. . . . .	iv
List of Publications. . . . .	v
Contents. . . . .	vi
Abstract. . . . .	viii
<b>Chapter</b>	
<b>1 Introduction</b>	<b>1</b>
<b>2 Some Classical Numerical approach to solve a set of linear equations</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Direct Methods. . . . .	6
2.2.1 Gauss Elimination Method. . . . .	6
2.2.2 Crout Method. . . . .	7
2.3 Iterative Methods . . . . .	9
2.3.1 Jacobi Method. . . . .	11
2.3.2 Gauss-Seidel Method. . . . .	12
2.3.3 Successive Relaxation (SR) Technique. . . . .	13
<b>3 Partial Differential Equation</b>	<b>15</b>
3.1 Introduction . . . . .	15
3.2 Finite Difference Method. . . . .	16
3.2.1 Liebmann's Iterative Method. . . . .	18
3.3 Examples. . . . .	19
<b>4 An Overview of Evolutionary Computations</b>	<b>25</b>
4.1 Introduction. . . . .	25
4.2 Variants of Evolutionary Algorithms. . . . .	26
4.3 Basic Mechanisms of Evolutionary Algorithms. . . . .	27
4.3.1 Time-Variant Mutation. . . . .	29
4.3.2 Development of Time-Variant Adaptive Parameters. . . . .	30
4.4 Modern Trends: Hybrid Algorithms. . . . .	32
4.5 Properties of Evolutionary Algorithms. . . . .	33
4.6 Merits and Demerits of Evolutionary Algorithms. . . . .	35
4.6.1 Merits. . . . .	35
4.6.2 Demerits. . . . .	36
4.7 Some Hybrid Algorithms. . . . .	36
4.7.1 Jacobi Based Uniform Adaptive Hybrid Algorithm. . . . .	37
4.7.2 Jacobi Based Time Variant Adaptive Hybrid Algorithm. . . . .	40
4.7.3 Gauss-Seidel Based Time Variant Adaptive Hybrid Algorithm. . . . .	41

<b>5</b>	<b>Evaluation of Uniform Adaptive Hybrid Evolutionary Algorithms for solving a set of Linear Equations</b>	<b>45</b>
5.1	Introduction.....	45
5.2	The Existing Uniform Adaptive Hybrid Evolutionary Algorithms.....	46
5.3	Necessity of Crossover.....	49
5.4	The Proposed Modified Hybrid Evolutionary Algorithms.....	49
5.5	Concluding Remarks.....	53
<b>6</b>	<b>Use of Hybrid Evolutionary Algorithms for solving Partial Differential Equations</b>	<b>55</b>
6.1	Introduction.....	55
6.2	Solution of Laplace's Equation.....	55
6.3	Solution of Poisson's Equation.....	58
	<b>Concluding Remarks</b>	<b>60</b>
	<b>Reference</b>	<b>62</b>





## Abstract

---

Hybrid Algorithms for solving set of linear equations are hybridization of evolutionary techniques and classical methods for solving set of linear equations. The classical iterative methods for solving set of linear equations are slow in terms of convergence and can be made faster by introducing relaxation factor  $\omega$  ( $0 < \omega < 2$ ). The process is very sensitive to the relaxation factor and the estimation of its optimum value is very difficult. Adaptation and selection mechanism of evolutionary computations serves the purpose of finding the optimum value of the relaxation factor and then the solution comes out. The four Hybrid Evolutionary Algorithms (JBUA, GSBUA, JBTVA and GSBTVA) were in front of us. Thorough study of the Uniform Adaptive Hybrid Evolutionary Algorithms JBUA and GSBUA showed that the crossover operation present in them is needless and thus we have proposed two modified Algorithms MJBUA and MGSBUA. We have tested the proposed MGSBUA separately for solving partial differential equations (especially in case of Laplace's equation). The solution of the discretized form is compared with the analytical one and the same set is also solved by the Gauss-Seidel method. It is found that our proposed method is faster and better accuracy can be achieved. We also have solved a sample Poisson's equation using our proposed algorithm. It is found that MJBUA and MGSBUA hybrid algorithms are faster and memory effective than their original counterparts.

## Introduction

The Evolutionary Computation techniques are inspired by the natural process of evolution [Hashem (1999)]. Selection and variation are the two basic principles in evolution and when these principals are found in any computation technique then that can be termed as Evolutionary computation technique. The algorithms which contain these strategies are termed as Evolutionary Algorithms. The variations of Evolutionary Algorithms bring differing philosophies of how to algorithmically abstract the model of natural evolution. Due to difference in commitment to levels of abstraction distinct emphasis are given that leads to a commitment to representations and philosophy of operators. Four main streams of these general algorithms, developed independent of each other, come forward – (i) Genetic Algorithms [Holland (1962), Bäck and Schwefel ((1993)], (ii) Evolutions strategies [Rechenberg (1993), Bäck and Schwefel (1993) and Hashem (1999)], (iii) Evolutionary Programming [Fogel et al. (1966), Bäck and Schwefel (1993)] and (iv) Genetic Programming (GP) [Koza (1994) and Hashem (1999)]. They have their own capabilities to yield good approximate solutions of optimization problem. Different algorithms emphasizes different features as being most important for a successful evolution process. Evolutions strategies and Evolutionary programming concentrate on mutation as the main search operator, while the rule of pure random mutation in Genetic Algorithms and Genetic Programming is usually seen to be of secondary importance. Recombination and probabilistic selection mechanisms have their own merits in different algorithms.

The inherent strength of Evolutionary Algorithms lies in the choice of the mutation steps (Rechnberg (1994), Fogel (1995), Bäck et al. (1996)]. According to the biological evidence Time Variant Mutation operator can improve the fine local tuning and can reduce the disadvantage of uniform mutation (Michalewicz (1996), Bäck et al. (1997), Hashem (1999)].

The peculiarity of Evolutionary Computations is maintaining a set of points (called population) that are searched in parallel. Each point (individual) is evaluated according to the objective function (fitness function). Further a set of genetic operators is given that work on populations. They contribute to the two basic principles in evolution – *selection* and *variation*. Selection focuses the search for the “better” regions of the search space by given individuals with “better” fitness values and higher probability to be member of the next generations (loop iteration). On the other hand, variation operators create new points in the search space. Here not only random changes (mutations) of particular point are possible but also the random mixing of the information of two or more individuals (crossover/ recombination) are possible [Bäck and Schwefel (1993), Schoenauer and Michalewicz (1997) and Bäck et al. (1997), Hashem (1999)]. Evolutionary Computations are often characterized as combining features from path-oriented methods and volume-oriented methods. Evolutionary Computations combine these contrary features in so far that in the beginning of the search the population is usually spread out in the whole search space, corresponding to a volume-oriented search. In the latter stages of the search algorithm has focused few (or single) region due to selection and the selected region is examined further. In this respect the algorithm behaves like a path-oriented search [Hashem (1999)]. Another possible identification of these two stages of the search could be the correspondence of the first stage to a global reliability strategy (coarse grin search) and the second stage to a local refinement strategy (fine grin search) [Yuret (1994), Hashem (1999)]. It is also observed that there are two important issues in the simulated search process of natural evolution: population diversity (exploration) and selective pressure (exploitation). These factors are strongly related – a strong selective pressure “supports” the premature convergence of evolutionary search and a weak selective pressure can make the search ineffective. Thus it is important to strike a balance between these two factors [Hashem (1999), Michalewicz (1996), Blicke (1997)].

Large set of linear equations are very common in Engineering and other physical applications. Though many classical methods are available for solving them yet rapidly convergent and efficient methods are still of interest. Iterative methods are preferred to solve them as computers can be utilized for solution purpose. The convergence of the Jacobi and the Gauss-Seidel methods (both are iterative in nature) is slow. Their convergence can be made faster by introducing successive relaxation technique. But the

speed depends on the relaxation factor and the technique is very much sensitive to that. The optimum value of the relaxation factor is difficult to estimate. In hybrid algorithms the fitness of certain individual serves the optimality of relaxation parameter in terms of the error of estimation. In this way the classical iterative methods with successive relaxation is hybridized with the evolutionary algorithms. The generation of new population, mutation etc. of evolutionary algorithms serves to generate new relaxation factors to estimate its fitness. The partial differential equations, after discretization, also give rise of set of linear equations. Thus the methods usable to solve set of linear equations are also useful to get numerical solutions of partial differential equations.

For our study purpose we have chosen four hybrid evolutionary algorithms – Jacobi Based Uniform Adaptive, Gauss-Seidel Based Uniform Adaptive, Jacobi Based Time Variant Adaptive and Gauss-Seidel Based Time Variant Adaptive hybrid evolutionary algorithms. All of them can be used to solve set of linear equations effectively.

The organization of this thesis is presented below:

**Chapter 2** contains discussion on some classical numerical approach to solve a set of linear equations. There Jacobi's iteration method, Gauss-Seidel method, Successive relaxation method, along with Gauss elimination method is discussed. **Chapter 3** starts with the classification of Partial Differential Equations. How the discretized form of Partial Differential Equations can be solved is presented there in limited form. The overview on the Evolutionary Computation is presented in **Chapter 4**. Different aspects of Evolutionary Computations, its merits and demerits, along with some hybrid evolutionary algorithms are incorporated in **Chapter 4**. After thorough investigations on the chosen uniform adaptive hybrid evolutionary algorithms we have found that the presence of crossover operation in them is needless to solve a set of linear equations. This matter is presented in **Chapter 5**. Utilizing our findings we have presented two modified hybrid evolutionary algorithms there and the performance of them is also discussed. The result of the primary investigations of the modified algorithms inspired us to use them in the solution of Partial Differential Equations. We have used one of them in the next chapter i.e. in **Chapter 6**. There we have presented the solution of a steady state heat distribution problem with certain boundary conditions. The discretized equation gives rise of 121 equations with 121 unknowns. We have solved that set with our own devised tool and also by using a classical

numerical method. The results are compared with the analytical solution and a good match is found. Also we have used that method to solve a sample Poisson equation and the obtained solution is also presented in **Chapter 6**. Finally we have made our Concluding Remarks. Also the cited references are listed at the end of the thesis.



**Some Classical Numerical Approach to solve a set of Linear Equations**

**2.1 Introduction**

The importance of solving linear equations can be summarized in a single statement: solving linear equations pervades and enriches almost all areas in numerical computation. Numerous classical methods are available for the solution of system of linear equations using computers. Yet this field is constantly expanding as more and more new concepts and algorithms are developed almost every day. The reasons for such a rapid growth in this area are the advent of very high-speed large-memory computers and the non-availability of a best suited computational method in solving system of linear equations for all types of a given problem. Since linear equations can be expressed as matrix equations, these constitute an important aspect of matrix algebra. This chapter overviews the elementary concept of linear equations in matrix algebra and the classical numerical methods of solving linear equations.

Let us consider a set of n equations with n unknowns as follows

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\
 \dots & \\
 a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m
 \end{aligned}
 \tag{2.1}$$

This can be written as  $Ax = b$  (2.2)

where  $A = [a_{ij}]$ ,  $x = [x_1, x_2, x_3, \dots, x_n]^T$  and  $b = [b_1, b_2, b_3, \dots, b_n]^T$

There are mainly two methods to solve linear equations: Direct methods and Iterative methods.

## 2.2 Direct Methods

The term direct method indicates a method that solves a set of equations by techniques in which it is not needed to guess an approximate solution. This method involves elimination of a term containing one of the unknowns in all but one equation. One such step reduces the order of equations by one. Repeated elimination leads finally to one equation with one unknown. There are many direct methods to solve system of linear equations such as Gauss elimination method, Gauss-Jordon method, Crout method, Doolittle's method, etc [Gerald and Wheatley (1994), Jain et al. (1985), Forsythe and Moler (1967)]. The direct methods are efficient and effective for small number of unknowns. But direct methods are not suitable for solving very large set of linear equations. Since the order of operation of direct methods are  $O(n^3)$  (only consider multiplication and divisions) [Gerald and Wheatley (1994)] so it may produce a significant amount of round off error in calculation. Direct methods also inefficient for large sparse and structured matrices. Two well-known classical direct methods named Gauss Elimination and Crout LU decomposition methods are described below:

### 2.2.1 Gauss Elimination Method

A simple and most well known direct method of solving linear equations (Small and dense coefficient matrix) is Gauss elimination method. For small coefficient matrix, this method is frequently used. This method possess a systematic strategy for deducing the system of equations to the upper triangular form using the forward elimination approach and then back substitution process is used to obtain the set of values of the unknowns. The strategy, therefore, comprised two phases:

1. *Forward elimination phase*: This phase is concerned with the manipulation of equations in order to eliminate some unknowns from the equations and produce an upper triangular system. By the following way the coefficient matrix of Eqn. (2.2) is reduced to triangular matrix. The relation for obtaining the coefficient of the  $k$ th derived system has the general form:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \quad (2.3)$$

where  $i = k + 1$  to  $n$ ;  $j = k + 1$  to  $n$ ; and

$$a_{ij}^{(0)} = a_{ij} \quad \text{for } i = 1 \text{ to } n, \quad j = 1 \text{ to } n$$

The  $k$ th equation, which is multiplied by the factor  $a_{ik}/a_{kk}$ , is called the pivot equation and  $a_{kk}$  is called pivot element. The process of dividing the  $k$ th equation by  $a_{ik}/a_{kk}$  is referred to as normalization.

2. *Back substitution process*: This phase is concerned with the actual solution of the equations and uses the back substitution process on the reduced upper triangular system. After reducing the system of Eqn. (2.2), by the following way the relation for obtaining the  $k$ th unknown,  $x_k$ , has the general form:

$$x_k = \frac{1}{a_{kk}^{(k-1)}} \left[ b_k^{(k-1)} - \sum_{j=k+1}^n a_{kj}^{(k-1)} x_j \right] \quad (2.4)$$

$$\text{where } k = n-1 \text{ to } 1, \text{ and } x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}} \quad (2.5)$$

### 2.2.2 Crout Method

Although Gaussian elimination is the best known of the direct LU decomposition methods, Crout (or Doolittle) method is widely used. In direct method, Crout method is popular in programs because the storage space may be economized. There is no need to store the zeros in either **L** or **U**, and the ones on the diagonal of **U** can also be omitted. The LU decomposition is produced by Crout reduction method [Gerald and Wheatley (1994)] as follow:

$$l_{ik} = a_{ik} - \sum_{j=1}^{k-1} l_{ij} u_{jk}, \quad k \leq i, \quad i = 1, 2, \dots, n, \quad (2.6)$$



$$u_{ik} = \frac{1}{l_{ii}} \left( a_{ik} - \sum_{j=1}^{i-1} l_{ij} u_{jk} \right), \quad i \leq k, \quad k=1,2,\dots,n \quad (2.7)$$

(For  $k=1$ , the role for  $l$  reduces to  $l_{i1} = a_{i1}$  for  $i=1,2,\dots,n$ . And for  $i=1$ , the role for  $u$  reduces to  $u_{1k} = \frac{a_{1k}}{l_{11}}$  for  $k=2,3,\dots,n$ ).

where  $\mathbf{A} = [a_{ij}]$  is the coefficient matrix as in Eqn. (2.2),  $\mathbf{L} = [l_{ij}]$  is the lower triangular matrix and  $\mathbf{U} = [u_{ij}]$  is the upper triangular matrix.

Then the matrix  $\mathbf{A}$  can be transformed by the above equations and becomes

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & \cdots \\ a_{31} & a_{32} & a_{33} & \vdots & \vdots \\ \vdots & \vdots & \vdots & a_{n-1,n-1} & a_{n-,n} \\ a_{n1} & \cdots & \cdots & a_{n,n-1} & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} l_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & l_{22} & u_{23} & \vdots & u_{2n} \\ l_{31} & l_{32} & l_{33} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{(n-1)n} \\ l_{n1} & l_{n2} & \cdots & \cdots & l_{nn} \end{bmatrix} \quad (2.8)$$

Because the  $\mathbf{L}$  and  $\mathbf{U}$  matrices are condensed into one array and store their elements in the space  $\mathbf{A}$ , this method is often called a *compact scheme*.

Then the solution of the set of the Eqn. (2.2) is readily obtained with the  $\mathbf{L}$  and  $\mathbf{U}$  matrices by the following formulas:

The general equation for the reduction of  $b$  to  $b'$  are

$$\left. \begin{aligned} b'_1 &= \frac{b_1}{l_{11}} \\ b'_i &= \frac{1}{l_{ii}} \left( b_i - \sum_{j=1}^{i-1} l_{ij} b'_{jk} \right), \quad i=2,3,\dots,n \end{aligned} \right\} \quad (2.9)$$



And the equations for the back-substitution are

$$\left. \begin{aligned} x_n &= b'_n, \\ x_k &= b'_k - \sum_{j=k+1}^n u_{kj} x_j, \quad k=n-1, n-2, \dots, 1 \end{aligned} \right\} \quad (2.10)$$

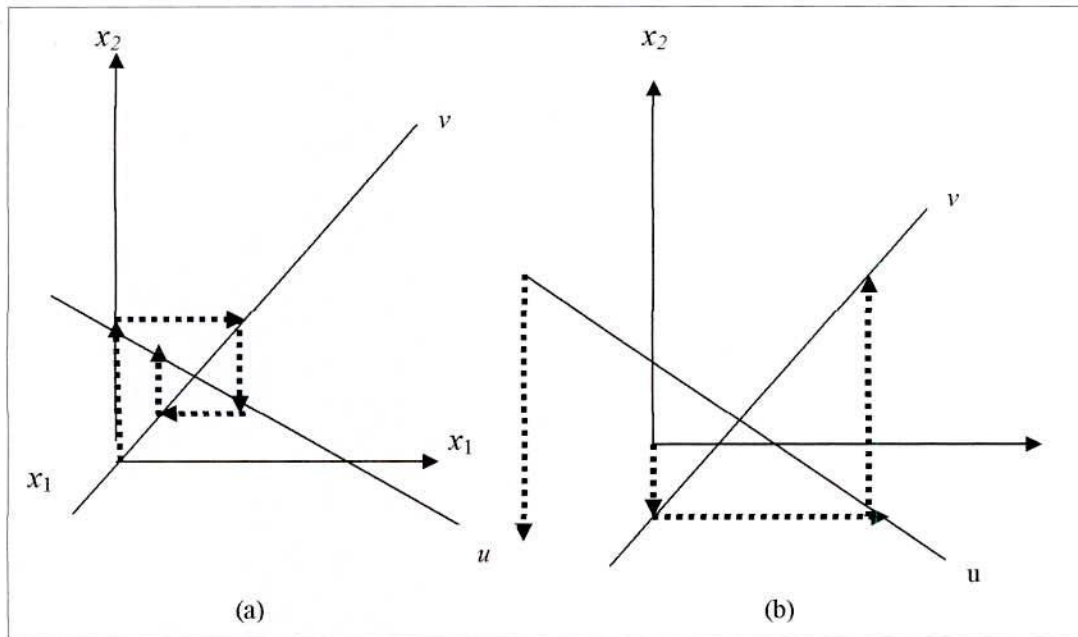
## 2.3 Iterative Methods

As opposed to the direct methods of solving a set of linear equations by Elimination, iterative methods are discussed now. Direct methods for solving linear systems, with their large number of operations proportional to  $n^3$  [Gerald and Wheatley (1994)], have a tendency to accumulate round off errors so that for a not well-conditioned coefficient matrix  $\mathbf{A}$ , the solution can become entirely useless. On the other hand, iterative methods are unaffected by round off error to a large extent, because each approximate solution with its inherent computational error can easily be improved upon in the following iteration steps. Iterative methods typically require around  $n^2$  operations [Gerald and Wheatley (1994)] for each iteration step. But unfortunately, they do not converge for all solvable systems [Chapra and Canale (1990)]. **Figure-2.1** illustrates the convergence and divergence of iterative methods applied to the same functions (line  $u$  and  $v$  in the figure). Thus the order in which the equations are implemented (as depicted by the direction of the first arrow from the origin in the figure) dictates whether the computation converges or diverges [Chapra and Canale (1990)]. In certain cases, these methods are preferred over the direct methods – especially when the coefficient matrices are sparse (has many zeros); in that situation they may be more rapid. They may be more economical in memory requirements of a computer. Apart from this, because of round off error, direct methods sometimes prove inadequate for large systems. Iterative methods may sometimes be used to reduce round off error in the solutions computed by direct methods, as discussed earlier.

An iterative technique to solve the linear system  $\mathbf{Ax} = \mathbf{b}$  starts with an initial approximation  $\mathbf{x}^{(0)}$  to the solution  $\mathbf{x}$  and generate a sequence of vectors  $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$  that converge to  $\mathbf{x}$ . Iterative technique involves a process that converts the system  $\mathbf{Ax} = \mathbf{b}$  into an equivalent system of the form

$$\mathbf{x} = \mathbf{Hx} + \mathbf{V} \quad (2.11)$$

For some fixed matrix  $\mathbf{H}$ , called iteration matrix, and vector  $\mathbf{V}$  [Jain et al. (1985), Chapra and Canale (1990), Mathews (2001)]. After the initial vector  $\mathbf{x}^{(0)}$  is selected, the sequence of approximate solution vectors is generated by computing



**Figure 2.1:** Illustration of (a) convergence and (b) divergence of iterative methods. Notices that the same functions (line  $u$  and  $v$  in the figure) are plotted in both cases.

$$\mathbf{x}^{(k)} = \mathbf{H}\mathbf{x}^{(k-1)} + \mathbf{V}, \quad \text{for each } k = 1, 2, \dots \quad (2.12)$$

An iteration matrix  $\mathbf{H}$  can be viewed as a correction on the last computed iteration [Chapra and Canale (1990)]

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{Z}^{(k)} \quad (2.13)$$

where  $\mathbf{Z}^{(k)}$  is called the correction vector or residual vector.

Subtracting Eqn. (2.11) from Eqn. (2.12) and if the error is defined as

$$\boldsymbol{\varepsilon}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}, \quad (2.14)$$

then

$$\boldsymbol{\varepsilon}^{(k+1)} = \mathbf{H}\boldsymbol{\varepsilon}^{(k)}, \quad k = 0, 1, 2, \dots \quad (2.15)$$

from which follows

$$\boldsymbol{\varepsilon}^{(k+1)} = \mathbf{H}^{k+1}\boldsymbol{\varepsilon}^{(0)}, \quad k = 0, 1, 2, \dots \quad (2.16)$$

There are mainly two basic iterative methods – Jacobi method and Gauss-Seidel method. The rate of convergence of both methods is relatively slow. The rate of convergence may be accelerated by using Successive Relaxation (SR) technique [Gerald and Wheatley

(1994), Varga (1962), Engeln-Müllges and Uhlig (1996)]. The two well-known iterative methods are discussed below including SR technique.

### 2.3.1 Jacobi Method

Assume that a linear system given (in the form Eqn. (2.2)) is

$$\mathbf{Ax} = \mathbf{b} \text{ with } |\mathbf{A}| \neq 0$$

Assume without loss of generality that none of the diagonal entries is zero; otherwise interchange its rows. Then

$$(\mathbf{D} + \mathbf{U} + \mathbf{L})\mathbf{x} = \mathbf{b}, \text{ where } \mathbf{A} = (\mathbf{D} + \mathbf{U} + \mathbf{L})$$

$$\text{or } \mathbf{D}\mathbf{x} = -(\mathbf{U} + \mathbf{L})\mathbf{x} + \mathbf{b}$$

$$\text{or } \mathbf{x} = -\mathbf{D}^{-1}(\mathbf{U} + \mathbf{L})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}$$

$$\text{or } \mathbf{x} = \mathbf{H}_j \mathbf{x} + \mathbf{V}_j \tag{2.17}$$

where  $\mathbf{H}_j = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ , called Jacobi iteration matrix, and  $\mathbf{V}_j = \mathbf{D}^{-1}\mathbf{b}$  [Engeln-Müllges and Uhlig (1996), Jain et al. (1985), Burder and Faires (1997), Cheney and Kincaid (1999)].

By solving the  $i$ th equation of Eqn.(2.2) for  $x_i$ , then an equivalent form for the system is [Antia (1991)]

$$x_i = -\sum_{\substack{k=1 \\ k \neq i}}^n \frac{a_{ik}}{a_{ii}} x_k + \frac{b_i}{a_{ii}}, \quad i = 1, \dots, n \tag{2.18}$$

And construct the sequence  $\{\mathbf{x}^{(k)}\}$  for an initial vector  $\mathbf{x}^{(0)}$  by setting

$$\begin{cases} \mathbf{x}^{(k+1)} = \mathbf{H}_j \mathbf{x}^{(k)} + \mathbf{V}_j & \text{with} \\ \mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})^t & \text{for } k = 0, 1, 2, \dots \end{cases} \tag{2.19}$$

Expressed in component-wise, this Jacobi iteration becomes

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j \neq i}}^n \frac{a_{ij}}{a_{ii}} x_j^{(k)}, \quad i = 1, \dots, n, \text{ and } k = 0, 1, 2, \dots \tag{2.20}$$

The iteration matrix  $\mathbf{H}_j$  can be viewed as a correction on the last computed iteration as Eqn. (2.13) i.e.

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)}$$

where

$$\mathbf{z}^{(k)} = \mathbf{V}_j - (\mathbf{I} - \mathbf{H}_j)\mathbf{x}^{(k)} \quad (2.21)$$

Jacobi method is also known as the method of simultaneous displacement method [Antia (1991)].

### 2.3.2 Gauss-Seidel Method

The Gauss-Seidel method differs from the Jacobi method slightly. The difference between the Jacobi and Gauss-Seidel methods is that in the later, as each component of  $\mathbf{x}^{(k)}$  is computed, and used it immediately in the iteration [Engeln-Müllges and Uhlig (1996), Jain et al. (1985), Burder and Faires (1997), Cheney and Kincaid (1999)]. Assume that a linear system given (in the form Eqn. (2.2)) is

$$\mathbf{Ax} = \mathbf{b} \text{ with } |\mathbf{A}| \neq 0$$

Assume without loss of generality that none of the diagonal entries of is zero; otherwise interchange it rows. Since in Gauss-Seidel method used on the right hand side all the available values from the present iteration. So

$$(\mathbf{D} + \mathbf{U} + \mathbf{L})\mathbf{x} = \mathbf{b}, \text{ where } \mathbf{A} = (\mathbf{D} + \mathbf{U} + \mathbf{L})$$

$$\text{or } (\mathbf{D} + \mathbf{L})\mathbf{x} = -\mathbf{U}\mathbf{x} + \mathbf{b}$$

$$\text{or } \mathbf{x} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}\mathbf{x} + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b}$$

$$\text{or } \mathbf{x} = \mathbf{H}_g \mathbf{x} + \mathbf{V}_g \quad (2.22)$$

where  $\mathbf{V}_g = (\mathbf{L} + \mathbf{D})^{-1}\mathbf{b}$  and  $\mathbf{H}_g = -(\mathbf{L} + \mathbf{D})^{-1}\mathbf{U}$ , called Gauss-Seidel iteration matrix.

And construct the sequence  $\{\mathbf{x}^{(k)}\}$  for an initial vector  $\mathbf{x}^{(0)}$  by setting

$$\begin{cases} \mathbf{x}^{(k+1)} = \mathbf{H}_g \mathbf{x}^{(k)} + \mathbf{V}_g & \text{with} \\ \mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})^t, & \text{for } k = 0, 1, 2, \dots \end{cases} \quad (2.23)$$

Expressed in component wise, this Gauss-Seidel iteration becomes

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} - \sum_{j=i+1}^n \frac{a_{ij}}{a_{ii}} x_j^{(k)}, i=1, \dots, n, \text{ and } k = 0, 1, \dots \quad (2.24)$$

The iteration matrix  $\mathbf{H}_g$  can be viewed as a correction on the last computed iteration as Eqn. (2.13) i.e

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \mathbf{Z}^{(k)}$$

where

$$\mathbf{Z}^{(k)} = \mathbf{V}_g - (\mathbf{I} - \mathbf{H}_g) \mathbf{X}^{(k)} \quad (2.25)$$

Gauss-Seidel method is also known as the method of successive displacement method.

### 2.3.3 Successive Relaxation (SR) Technique

Relaxation method represents a slight modification of the Jacobi/Gauss-Seidel method and is designed to enhance convergence [Carre' (1961), Young, (1954), Gerald and Wheatley (1994), Varga (1962), Engeln-Müllges and Uhlig (1996)]. Define an auxiliary vector  $\tilde{\mathbf{x}}$  as

$$\tilde{\mathbf{x}}^{(k+1)} = -\mathbf{D}^{-1} \mathbf{L} \mathbf{x}^{(k)} - \mathbf{D}^{-1} \mathbf{U} \mathbf{x}^{(k)} + \mathbf{D}^{-1} \mathbf{b}, \text{ for Jacobi method and}$$

$$\tilde{\mathbf{x}}^{(k+1)} = -\mathbf{D}^{-1} \mathbf{L} \mathbf{x}^{(k+1)} - \mathbf{D}^{-1} \mathbf{U} \mathbf{x}^{(k)} + \mathbf{D}^{-1} \mathbf{b}, \text{ for Gauss-Seidel method}$$

Then using SR technique the final solution is now written as

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \omega \mathbf{Z}^{(k)} \quad (2.26)$$

where  $\mathbf{Z}^{(k)}$  is the correction vector and  $\omega$  is a relaxation factor.

$$\text{or } \mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \omega (\tilde{\mathbf{x}}^{(k+1)} - \mathbf{X}^{(k)})$$

$$\text{or } \mathbf{X}^{(k+1)} = (1 - \omega) \mathbf{X}^{(k)} + \omega \tilde{\mathbf{x}}^{(k+1)} \quad (2.27)$$

Here  $\mathbf{x}^{(k+1)}$  is weighted mean of  $\tilde{\mathbf{x}}^{(k+1)}$  and  $\mathbf{x}^{(k)}$  and  $\omega$  is a weighted factor that is assigned a value between 0 and 2 [Krishnamurthy and Sen (1989), Gerald and Wheatley (1994), Varga (1962), Engeln-Müllges and Uhlig (1996)].

- (i) For  $\omega = 1$ , the Eqn. (2.26) reduced to the Jacobi/Gauss-Seidel method [Krishnamurthy and Sen (1989), Gerald and Wheatley (1994)].

- (ii) If  $\omega$  is set at a value between 0 and 1, the result is weighted average of corresponding previous result and sum of other (present or previous) result. It is typically employed to make a non-convergence system or to hasten convergence by dampening out oscillations. This approach is called successive under relaxation [Gerald and Wheatley (1994), Krishnamurthy and Sen (1989)].
- (iii) For value of  $\omega$  from 1 to 2, extra weight is placed. In this instance, there is an implicit assumption that the new value is moving in the correct direction towards the true solution but at a very slow rate. Thus, the added weight  $\omega$  is intended to improve the estimate by pushing it closer to the truth. Hence this type of modification, which is called over relaxation, is designed to accelerate the convergence of an already convergent system. This approach is called successive over relaxation (SOR) [Gerald and Wheatley (1994), Krishnamurthy and Sen (1989)].
- (iv) The combine approach, i.e. for value of  $\omega$  from 0 to 2, is called successive relaxation or SR technique [Gourdin and Boumahrat (1996), Engeln-Müllges and Uhlig (1996)].

## Partial Differential Equation

### 3.1 Introduction

Many physical phenomena in applied science and engineering when formulated into mathematical models fall into a category of systems known as *Partial differential equations*. A partial differential equation is a differential equation involving more than one independent variable. These variables determine the behavior of the dependent variable as described by their *partial derivatives* contained in the equation. Some of the problems which lend themselves to partial differential equations include:

1. Study of displacement of a vibrating string,
2. Heat flow problems,
3. Fluid flow analysis,
4. Electrical potential distribution,
5. Analysis of torsion in a bar subject to twisting,
6. Study of diffusion of matter, and so on.

Most of these problems can be formulated as second-order partial differential equations (with the highest order of derivative being the second). If we represent the dependent variable as  $f$  and the two independent variables as  $x$  and  $y$ , then we will have three possible

second-order partial derivatives  $\frac{\partial^2 f}{\partial x^2}$ ,  $\frac{\partial^2 f}{\partial x \partial y}$  and  $\frac{\partial^2 f}{\partial y^2}$  in addition to the two first-order

partial derivatives  $\frac{\partial f}{\partial x}$  and  $\frac{\partial f}{\partial y}$ .

We can write a second-order equation involving two independent variables in general form as

$$A \frac{\partial^2 f}{\partial x^2} + B \frac{\partial^2 f}{\partial x \partial y} + C \frac{\partial^2 f}{\partial y^2} = F\left(x, y, f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \quad (3.1)$$



where the coefficients A, B and C may be constants or functions of x and y. Depending on the values of these coefficients, equation (3.1) may be classified into one of the three types of equations, namely, *elliptic*, *parabolic* and *hyperbolic*.

$$\text{Elliptic,} \quad B^2 - 4AC < 0$$

$$\text{Parabolic,} \quad B^2 - 4AC = 0$$

$$\text{Hyperbolic,} \quad B^2 - 4AC > 0$$

If A, B and C are functions of x and y, then depending on the values of these coefficients at various points in the domain under consideration, an equation may change from one classification to another.

Solution of partial differential equations is too important to ignore but too difficult to cover in depth in brief. Since the application of analytical methods becomes more complex, we seek the help of numerical techniques to solve partial differential equations. There are basically two numerical techniques, namely, *finite-difference* method and *finite-element* method that can be used to solve partial differential equations. The finite-element method is very important for solving equations where regions are irregular. We will discuss here the application of finite-difference methods only, which are based on formulae for approximating the first and second derivatives of a function. We will also consider problems, only those where the coefficients A, B and C are constants.

### 3.2 Finite difference method

Consider the problem

$$a \frac{\partial^2 f}{\partial x^2} + b \frac{\partial^2 f}{\partial x \partial y} + c \frac{\partial^2 f}{\partial y^2} = F \left( x, y, f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \quad (3.2)$$

Equation (3.2), when  $a = 1$ ,  $b = 0$ ,  $c = 1$ , and  $F(x, y, f, f_x, f_y) = 0$ , becomes

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \nabla^2 f = 0 \quad (3.3)$$

The operator

$$\nabla^2 = \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)$$

is called the *Laplacian operator* and Eqn. (3.3) is called *Laplace's equation*. (Many authors use  $u$  in place of  $f$ )

To solve the Laplace equation on a region in the  $xy$ -plane, we subdivide the region in two-dimensional finite difference grid. Consider the portion of the region near  $(x_i, y_i)$ . We have to approximate

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

$$(f_{i,j})_{xx} = \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h^2} \quad (3.4)$$

$$(f_{i,j})_{yy} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{k^2} \quad (3.5)$$

Replacing the second-order derivatives by their finite difference equivalents from equation (3.4) and (3.5) at the point  $(x_i, y_i)$ , we obtain,

$$\nabla^2 f_{i,j} = \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{h^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{k^2}$$

If we assume, for simplicity,  $h = k$ , then we get

$$\nabla^2 f_{i,j} = \frac{1}{h^2} (f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{i,j}) = 0 \quad (3.6)$$

Note that Equation (3.6) contains four neighboring points around the central point  $(x_i, y_i)$  (on all the four sides) as shown in **Figure 3.1**. Equation (3.6) is shown as the *five-point difference formula* for Laplace's equation.

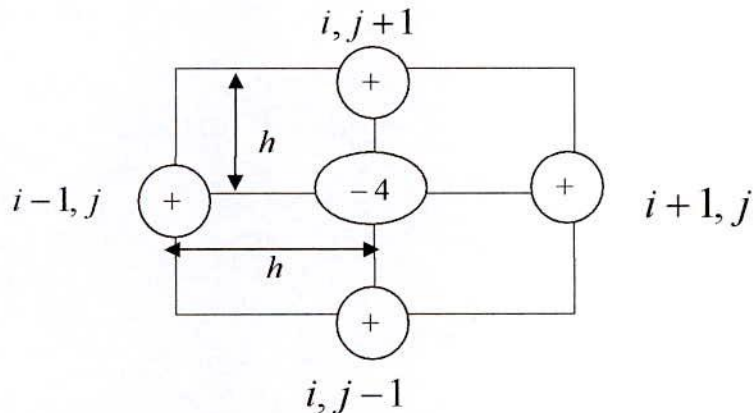


Fig. 3.1 Grid for Laplace's equation

We can also represent the relationship of pivotal values pictorially as in equation (3.7).

$$\nabla^2 f_{i,j} = \frac{1}{h^2} \begin{Bmatrix} 1 & -4 & 1 \\ 1 & & 1 \end{Bmatrix} f_{i,j} = 0 \quad (3.7)$$

From equation (3.6) we can show that the function value at the grid point  $(x_i, y_i)$  is the average of the values at the four adjoining points. That is,

$$f_{i,j} = \frac{1}{4} (f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}) \quad (3.8)$$

To evaluate numerically the solution of Laplace's equation at the grid points, we can apply equation (3.8) at the grid points where  $f_{i,j}$  is required (or unknown), thus obtaining a system of linear equations in the pivotal values  $f_{i,j}$ . The system of linear equations may be solved using either direct methods or iterative methods.

### 3.2.1 Liebmann's Iterative Method

We have Laplace's equation  $\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \nabla^2 f = 0$  (3.9)

We know that a diagonally dominant system of linear equations can be solved by iteration methods such as Gauss-Seidel method. When such iteration is applied to Laplace's equation, the iterative method is called *Liebmann's iterative method*.

To obtain the pivotal values of  $f$  by Liebmann's iterative method, we solve for  $f_{i,j}$  the equations obtained from (3.8). That is,

$$f_{i,j} = \frac{1}{4} (f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}) \quad (3.10)$$

The value  $f_{i,j}$  at the point  $(i, j)$  is the average of the values of  $f$  at the four adjoining points. If we know the "initial values" of the functions at the right-hand side of equation (3.10) we can estimate the value  $f$  at the point  $(i, j)$ . We can substitute the values, thus obtained, into the right-hand side to achieve improved approximations. This process may continue till the values  $f_{i,j}$  converge to constant values.

Initial values may be obtained by either taking *diagonal average* or *cross average* of the adjoining four points.

### 3.3 Examples

#### Example 1:

Consider a steel plate of size 15 cm × 15 cm. If two of the sides are held at 100°C and the other two sides are held at 0°C, what are the steady state temperature at interior points assuming a grid size of 5 cm × 5 cm.

#### Solution:

A problem with the values known on each boundary is said to have *Dirichlet boundary conditions*. The problem is illustrated below.

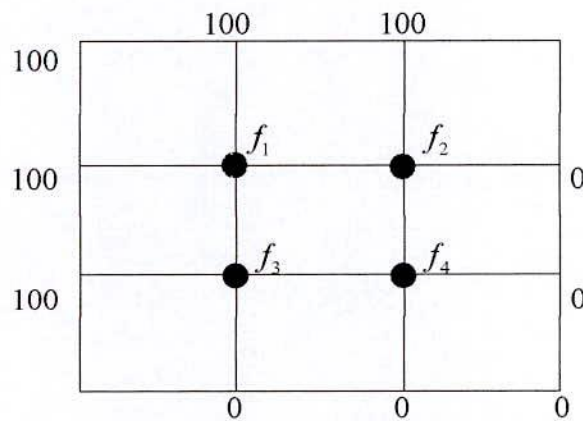


Fig. 3.2 Grid for Laplace's equation

The system of equation is as follows:

$$\text{At point 1: } f_2 + f_3 - 4f_1 + 100 + 100 = 0$$

$$\text{At point 2: } f_1 + f_4 - 4f_2 + 100 + 0 = 0$$

$$\text{At point 3: } f_1 + f_4 - 4f_3 + 100 + 0 = 0$$

$$\text{At point 4: } f_2 + f_3 - 4f_4 + 0 + 0 = 0$$

That is,

$$-4f_1 + f_2 + f_3 + 0 = -200$$

$$f_1 - 4f_2 + 0 + f_4 = -100$$

$$f_1 + 0 - 4f_3 + f_4 = -100$$

$$0 + f_2 + f_3 - 4f_4 = 0$$

Solution of this system is

$$f_1 = 75 \qquad f_2 = 50$$

$$f_3 = 50 \qquad f_4 = 25$$

Note that there is a symmetry in the temperature distribution, i.e. it can be stated that

$$f_2 = f_3$$

and therefore the number of equations in **Example 1** may be reduced to three equations with three unknowns as shown below.

$$-4f_1 + 2f_2 = -200$$

$$f_1 - 4f_2 + f_4 = -100 \tag{3.11}$$

$$f_2 - 2f_4 = 0$$

**Example 2:**

Solve the problem in **Example 1** using **Liebmann's iterative method** correct to one decimal place.

**Solution:**

By applying equation (3.8) to every grid points, we obtain

$$f_1 = \frac{f_2 + f_3 + 200}{4}$$

$$f_2 = \frac{f_1 + f_4 + 100}{4}$$

$$f_3 = \frac{f_1 + f_4 + 100}{4} \quad (3.12)$$

$$f_4 = \frac{f_2 + f_3}{4}$$

Appropriate initial values for the iterative solution are obtained by taking diagonal average at 1 and cross average at other points, assuming first  $f_4 = 0$ .

$$f_1 = \frac{1}{4}(100 + 100 + 100 + 0) = 75.00 \text{ (Average)}$$

$$f_2 = \frac{1}{4}(75 + 100 + 0 + 0) = 43.75$$

$$f_3 = \frac{1}{4}(100 + 75 + 0 + 0) = 43.75$$

$$f_4 = \frac{1}{4}(43.75 + 43.75) = 21.88$$

Note that  $f_2$ ,  $f_3$ , and  $f_4$  are computed using the latest values on the right hand side.

Using these initial values in equation (3.12) and performing iterations gives the values as shown in **Table 3.1**.

**Table 3.1**

$f_i$	Initial Values	Iterations			
		1	2	3	4
$f_1$	75.00	71.88	74.22	74.81	74.95
$f_2$	43.75	48.44	49.61	40.90	49.98
$f_3$	43.75	48.44	49.61	40.90	49.98
$f_4$	21.88	24.22	24.81	24.95	24.99

The process may be continued till we get identical values in the last two columns. Note that the values are approaching to correct answers obtained in **Example – 1**.

**Example 3:**

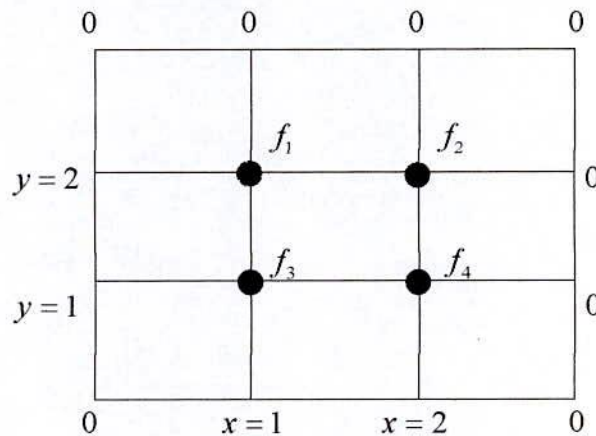
Solve the Poisson equation  $\nabla^2 f = 2x^2y^2$  over the square domain  $0 \leq x \leq 3$  and  $0 \leq y \leq 3$  with  $f = 0$  on the boundary and  $h = 1$  by finite difference formula.

**Solution:** We have the finite difference formula for solving Poisson's equation

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = g(x, y) \text{ then take the form}$$

$$f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{i,j} = h^2 g_{i,j} \quad (3.13)$$

The domain is divided into squares of one unit size as illustrated below:



By applying equation (3.13) at each grid point, we get the following set of equations:

At point 1:  $0 + 0 + f_2 + f_3 - 4f_1 = 2(1)^2(2)^2$   
 i.e.  $f_2 + f_3 - 4f_1 = 8$  (3.14.a)

At point 2:  $0 + 0 + f_1 + f_4 - 4f_2 = 2(2)^2(2)^2$   
 i.e.  $f_1 - 4f_2 + f_4 = 32$  (3.14.b)

At point 3:  $0 + 0 + f_1 + f_4 - 4f_3 = 2(1)^2(1)^2$   
 i.e.  $f_1 - 4f_3 + f_4 = 2$  (3.14.c)

At point 4:  $0 + 0 + f_2 + f_3 - 4f_4 = 2(2)^2(1)^2$   
 i.e.  $f_2 + f_3 - 4f_4 = 8$  (3.14.d)

Rearranging the equations (a) to (d), we get

$$\begin{aligned} -4f_1 + f_2 + f_3 &= 8 \\ f_1 - 4f_2 + f_4 &= 32 \\ f_1 - 4f_3 + f_4 &= 2 \\ f_2 + f_3 - 4f_4 &= 8 \end{aligned} \tag{3.15}$$

Solving the equation (3.15) by elimination method, we get the answers.

$$\begin{aligned} f_1 &= -\frac{22}{4} & f_2 &= -\frac{43}{4} \\ f_3 &= -\frac{13}{4} & f_4 &= -\frac{22}{4} \end{aligned}$$

**Example 4:**

Solve the problem in **Example – 3** using **Liebmann's iterative method.**

**Solution:** By using the equations (3.15), we have

$$\begin{aligned} -4f_1 + f_2 + f_3 &= 8 \\ f_1 - 4f_2 + f_4 &= 32 \\ f_1 - 4f_3 + f_4 &= 2 \\ f_2 + f_3 - 4f_4 &= 8 \end{aligned} \tag{3.16}$$

By rearranging the above equations, we have

$$\begin{aligned} f_1 &= \frac{1}{4}(f_2 + f_3 - 8) \\ f_2 &= \frac{1}{4}(f_1 + f_4 - 32) \\ f_3 &= \frac{1}{4}(f_1 + f_4 - 2) \\ f_4 &= \frac{1}{4}(f_2 + f_3 - 8) \end{aligned} \tag{3.17}$$



Note that  $f_1 = f_4$ , Therefore the equation (3.17) becomes,

$$f_1 = \frac{1}{4}(f_2 + f_3 - 8)$$

$$f_2 = \frac{1}{4}(2f_1 - 32)$$

$$f_3 = \frac{1}{4}(2f_1 - 2)$$

Assume starting values as  $f_2 = 0 = f_3$

*Iteration 1:*

$$f_1 = -2, \quad f_2 = -9, \quad f_3 = -1$$

*Iteration 2:*

$$f_1 = -\frac{18}{4}, \quad f_2 = -\frac{41}{4}, \quad f_3 = -\frac{11}{4}$$

*Iteration 3:*

$$f_1 = -\frac{22}{4}, \quad f_2 = -\frac{43}{4}, \quad f_3 = -\frac{13}{4}$$

*Iteration 4:*

$$f_1 = -\frac{22}{4}, \quad f_2 = -\frac{43}{4}, \quad f_3 = -\frac{13}{4}$$

---

## An Overview of Evolutionary Computations

### 4.1 Introduction

The Evolutionary Computation (EC) techniques are inspired by the natural process of evolution [Hashem (1999)]. The peculiarity of ECs is maintaining a set of points (called population) that are searched in parallel. Each point (individual) is evaluated according to the objective function (fitness function). Further a set of genetic operators is given that work on populations. They contribute to the two basic principles in evolution – *selection* and *variation*. Selection focuses the search for the “better” regions of the search space by given individuals with “better” fitness values and higher probability to be member of the next generations (loop iteration). On the other hand, variation operators create new points in the search space. Here not only random changes (mutations) of particular point are possible but also the random mixing of the information of two or more individuals (crossover/ recombination) are possible [Bäck and Schwefel (1993), Schoenauer and Michalewicz (1997) and Bäck et al. (1997), Hashem (1999)]. ECs are often characterized as combining features from path-oriented methods and volume-oriented methods. ECs combine these contrary features in so far that in the beginning of the search the population is usually spread out in the whole search space, corresponding to a volume-oriented search. In the latter stages of the search algorithm has focused too few (or single) region due to selection and the selected region is examined further. In this respect the algorithm behaves like a path-oriented search [Hashem (1999)]. Another possible identification of these two stages of the search could be the correspondence of the first stage to a global reliability strategy (coarse grin search) and the second stage to a local refinement strategy (fine grin search) [Yuret 1994, Hashem (1999)]. It is also observed that there are two important issues in the simulated search process of natural evolution: population diversity (exploration) and selective pressure (exploitation). These factors are strongly related – a strong selective pressure “supports” the premature convergence of evolutionary search and

➤ a weak selective pressure can make the search ineffective. Thus it is important to strike a balance between these two factors [Hashem (1999), Michalewicz (1996), Blicke (1997)].

## 4.2 Variants of Evolutionary Algorithms

The variations of Evolutionary Algorithms (EAs) that are of current interest bring differing philosophies of how to algorithmically abstract the model of natural evolution. Because of differing commitment to levels of abstraction, each uses a distinct emphasis that leads to a commitment to representations and philosophy of operators. Four main streams of instances of these general algorithms, developed independently of each other, can now a days be identified – (i) Genetic Algorithms (GAs) [Holland (1962), Bäck and Schwefel (1993)], (ii) Evolutions Strategies (ESs) [Rechenberg (1973), Bäck and Schwefel (1993) and Hashem (1999)], (iii) Evolutionary Programming (EP) [Fogel et al. (1966), Bäck and Schwefel (1993)] and (iv) Genetic Programming (GP) [Koza (1994) and Hashem (1999)]. Each of these main stream algorithms have clearly demonstrated their capability to yield good approximate solutions even in the cases of complicated multimodal, discontinues, non-differentiable, and noisy or moving response surfaces of optimization problems. The variety of data-structures, variation of operators and selection mechanisms give possible ways of classifying Genetic Algorithms. However, the different terms are mostly historical. Moreover, the differences between the variants are fluid. Furthermore, these algorithms are specified for parameter optimization problems.

It is a remarkable fact that each algorithm emphasizes different features as being most important for a successful evolution process. In analogy to repair-enzymes, which give evidence for a biological self-control of mutation rates of nucleotide bases in DNA, both ESs and EP use self-adaptation processes for the mutation rates. In canonical GAs, this concept was successfully tested only recently [Bäck (1992)], but still need more time to be recognized and applied. Both ESs and EP concentrate on mutation as the main search operator, while the rule of pure random mutation in canonical GAs and GPs is usually seen to be of secondary importance. On the other hand, recombination plays a major rule in canonical GAs and GPs, but recombination is missing completely in EPs and is urgently necessary for use in connection to self-adaptation in ESs. Finally, canonical GAs, GPs and EPs emphasize on a necessarily probabilistic selection mechanism, while from the ESs

point of view selection is completely deterministic without any evidence for the necessity of incorporating probabilistic rules. In contrast, both ESs and EPs definitely exclude some individuals from being selected for reproduction, i.e. they use extinctive selection mechanisms, while canonical GAs and GPs generally assign a non-zero selection probability to each parent individual, which can be termed as *preservative selection mechanism*. The characteristic similarities and differences of the evolutionary algorithms discussed in this chapter are summarized in **Table 4.1** [Hashem (1999)].

**Table 4.1:** Main characteristics of evolutionary algorithms.

Characteristics	GA	ES	EP	GP
Abstraction level	Organism	Individual behavior	Species behavior	Organism
Representation	Binary-valued	Real-valued	Real-valued	Tree like Structure
Self-adaptation	None	Standard deviation & covariance	Standard deviation	None
Fitness	Scaled objective function value	Objective function value	Objective function value	Objective Function value
Mutation	Background operation	Main operation	Only operation	Secondary Operation
Recombination	Main operation	Different variants, important for self-adaptation	None	Main operation
Selection	Probabilistic, preservative	Deterministic, extinctive	Probabilistic, extinctive	Probabilistic, preservative

### 4.3 Basic Mechanisms of Evolutionary Algorithms

For the sake of clarity, we shall try to introduce a general framework according as much as possible for most of the existing EAs. The EAs can be classified as probabilistic search algorithms, which maintain a population of  $n$  individuals,  $\Pi(t) = \{\psi_1(t), \psi_2(t), \dots, \psi_n(t)\}$  where  $\psi_i(t) \in \mathbf{S}$  for generation  $t$  which simultaneously sample of the search space  $\mathbf{S}$ . Each individual represents a potential solution to the problem at hand and is implemented as some complex data structure and/or

object variable vector  $\xi$  with component  $\xi_i \in \mathfrak{R} \forall i \in \{1, 2, \dots, n\}$ . Each solution  $\psi(t)$  is evaluated to produce some measure of its “fitness”  $\phi(t)$ . After initialization of the population, a new population is formed by three main operators – crossover (recombination), mutation and selection. There is higher order transformation:  $\zeta$  (crossover operator), which creates new individuals (offspring) ( $\zeta : (\xi \times \xi)^u \rightarrow \xi^v$ ) where  $v$  is the offspring population size and an unary transformation:  $\zeta'$  (mutation operator), which modifies these new individuals (offspring) by a small change ( $\zeta' : \xi \rightarrow \xi$ ). A selection operation ( $\zeta : (\xi^v|_{\text{offspring}} \cup \xi^u|_{\text{parent}}) \rightarrow \xi^u$ ) is then applied to choose the parent population for the next generation. After some number of generations the program converges – it is hoped that the best individual represents a near optimum solution [Hashem (1999), Bäck and Schwefel (1993)].

Variation is introduced into the population by crossover and/or mutation. Since these operators usually create offspring at new positions in the search space, they are also called “explorative” operators. The several instances of the EAs differ in the way that how individuals are represented and in the realization of the recombination operator. Common representations are, for example, bit strings, vectors of real or integer values (for parameter optimization), trees (for function optimization), graphs or any other problem dependent data-structure. Based on information-theoretical considerations, John Holland suggests that the bit-string representation is optimal. Bäck et al. (1993) suggests from practical experience, as well as some theoretical point of view that the bit-string representations have some disadvantages such as the coding and decoding functions might introduce additional multimodality along with the objective function [Michalewicz (1994, 1994a), Michalewicz and Attia (1994), Kim and Myung (1997) and Chellapilla et al. (1998)].

Along with a particular data-structure, variation operators have to be defined which can be divided in asexual and sexual variation operators. The asexual variation (mutation) consists of a random change of the information represented by an individual. If the individual is represented as a vector, mutation is the random change of elements of the vector. How this change is performed depends on the type of the vector-elements. If the vector is a simple bit-string, mutation is to toggle the bit or not (with equal probability). For real or integer

values more sophisticated mutation operators are necessary. The most general approach is to define a probability distribution over the domain of possible values for a particular vector element. A new value is then chosen according to this distribution. During sexual variation (Crossover /recombination) two individuals exchange or blend part of their information. Two individuals are chosen from the population and named parents. How the exchange or blend of information is performed depends of the chosen representation. There is no need to restrict the number of parents for crossover to two. Recent research shows that increasing the number of mates leads to an increased performance [Blickle (1997), Salomon (1998)]. There is an ongoing debate between different communities which operator- mutation or crossover – is more important. Some researchers found evidence that the crossover operator might be “simulated” by mutation [Fogel (1995)].

### 4.3.1 Time-Variant Mutation

The inherent strength of EAs – towards convergence and high precision results – lies in the choice of the mutation steps i.e. standard deviation [Rechnberg (1994), Fogel (1995), Bäck et. al. (1996)]. According to the biological evidence, a special dynamic Time-Variant Mutation (TVM) operator is proposed aiming to both improving the fine local tuning and reducing the disadvantage of uniform mutation [Michalewicz (1996), Bäck et. al. (1997), Hashem (1999)]. Moreover, it can exploit the fast (but not premature) convergence. By this mutation scheme, a natural behavioral change at the level of individuals will be achieved. The TVM is defined for a child – as that of EAs do [Bäck et. al. (1993), Schwefel et al. (1995)] as  $\forall i \in \{1, 2, \dots, n\}$

$$\xi_i^{(m)} = \xi_i + \sigma(t) \cdot N_i(0,1) \quad (4.1)$$

where  $N_i(0, 1)$  indicates that Gaussian random value with zero-mean and unity variance, it is sampled anew for each value of the index  $i$  and  $\sigma(t)$  is the time-variant mutation step generating function at the generation  $t$ , which is defined by

$$\sigma(t) = \left[ 1 - q^{\left(1 - \frac{1}{T}\right)^t} \right] \quad (4.2)$$

where  $q \in (0,1)$ , is uniform random number,  $T$  is the maximal generation,  $\gamma$  is a real-valued parameter determining the degree of dependency on generations. The parameter  $\gamma$  is also called an exogenous parameter of the method [Hashem 1999].

The function  $\sigma(t)$  returns a value in the range  $[0, 1]$ , that falls within so-called evolution window [Rechebberg (1994)] such that the probability of  $\sigma(t)$  being closed to 0 as the generation  $t$  increases. This property of  $\sigma(t)$  causes to search the problem space uniformly (volume- oriented search) initially when  $t$  is small and very locally (path – oriented search) at larger  $t$  stages. Another possible identification of these two stages of the search could be the correspondence of the first stage to a global reliability strategy (coarse grain search) and the second stage to a local refinement strategy fine grain search [Michalewicz (1994a), Michalewicz and Attia (1994), Michalewicz (1996)].

### 4.3.2 Development of Time-Variant Adaptive Parameters

As pointed earlier the inherent strength of EA – towards convergence and high precision results – lies in the choice of the mutation steps, i.e. standard deviation. Obvious biological evidence is that a rapid change is observed at early stages of life and a slow change is observed at latter stages of life in all kind of animals/ plants. These changes are more often occurred dynamically depending on the situation exposed to them. Jamali et al. (2004b) introduced a new Time-Variant Adaptive (TVA) parameter aiming at both improving the fine local tuning and reducing the disadvantage of uniform adaptation of relaxation factors as well as mutation for solving linear equations.

#### Formulas

The time variant adaptive (TVA) parameters are defined as

$$p_x = E_x \times N(0,0.25) \times T_\omega \quad (4.3)$$

and is denoted as adaptive (TVA) probability parameter of  $\omega_x$ , and

$$p_y = E_y \times |N(0,0.25)| \times T_\omega \quad (4.4)$$

and is denoted as adaptive (TVA) probability parameter of  $\omega_y$

$$\text{where } T_\omega = \lambda \ln\left(1 + \frac{1}{t + \lambda}\right), \lambda > 10 \quad (4.5)$$

$$\text{or } T_\omega = \left(1 - \frac{t}{T}\right)^\gamma \quad (4.6)$$

Here  $\lambda$  and  $\gamma$  are exogenous parameters, used for increased or decreased of rate of change of curvature with respect to number of iterations;  $t$  and  $T$  denote number of generation and maximum number of generation respectively. Also  $N(0, 0.25)$  is the Gaussian distribution with mean 0 and standard deviation 0.25.

Now  $E_x$  and  $E_y$  denote the approximate initial boundary of the variation of TVA parameters of  $\omega_x$  and  $\omega_y$ , respectively. And if  $\omega^*$  is denoted as the optimal relaxation factor then

$$E_x = p_x |_{\max} = \frac{\omega_y \sim \omega_x}{2(\omega_x + \omega_y)}, \text{ so that } \omega_x^m = (0.5 + p_x |_{\max})(\omega_x + \omega_y) \approx \omega_y$$

$$\text{and } E_y = p_y |_{\max} = \frac{\omega^* \sim \omega_y}{\omega_y - \omega_L} \quad \text{or} \quad \frac{\omega^* \sim \omega_y}{\omega_U - \omega_y}, \text{ so that}$$

$$\omega^* \approx \omega_y^m = \begin{cases} \omega_y + p_y |_{\max} (\omega_U - \omega_y), & \text{when } \omega_y > \omega_x \\ \omega_y + p_y |_{\max} (\omega_L - \omega_y), & \text{when } \omega_y < \omega_x \end{cases}$$

### Properties

The functions  $p_x$  and  $p_y$  return values in the range  $[-E_x, E_x]$  and  $[0, E_y]$  respectively, which falls within the so-called evolution window [Rechenberg (1994), Yao and Liu (1997)] such that probability of  $p_x$  and  $p_y$  tend to 0 as generation of population increased.

This property of  $p_x$  and  $p_y$  causes to search the space uniformly (volume-oriented search) initially when generation,  $t$ , is small and very locally (path oriented search) at larger  $t$  stages. Another possible identification of these two stages of search could be correspondence of the first stage to a global reliability strategy (coarse grain search) and the second stage to a local refinement strategy (fine grain search) [Hashem (1999)].



Now from Eqn. (4.5) i.e.  $T_{\omega} = \lambda \ln(1 + 1/(t + \lambda))$  (denoted as Lambda based TVA parameter) it is obvious that when the value of  $\lambda$  is small then, initially, rate of change of the function  $T_{\omega} = \lambda \ln(1 + 1/(t + \lambda))$  is very rapid; on the other hand when the value of  $\lambda$  is relatively large, then initially, rate of change of this function is relatively slow. For all the cases, in later stages, the rate of change is slow.

Again from Eqn. (4.6) i.e.  $T_{\omega} = (1 - t/T)^{\gamma}$  (denoted as Gamma based TVA parameter) it is obvious that when the value of  $\gamma$  is large then initially, rate of change of this function is very rapid; on the other hand when the value of  $\gamma$  is relatively small, then initially, rate of change of this function is relatively slow. The rate of change of this function is all most constant in all stages for each value of  $\gamma$ .

#### **4.4 Modern Trends: Hybrid Algorithms**

Many researchers modified further evolutionary algorithms “by adding” some problem specific knowledge to the algorithm. Several papers have discussed initialization techniques, different representations, decoding techniques (mapping from genetic representations to phenotype representations) and the use of heuristics for genetic operators. Such hybrid/nonstandard systems enjoy a significant popularity in evolutionary computation community. Very often these systems, extended by the problem-specific knowledge, outperform other classical evolutionary methods as well as other standard techniques. For example, a system Genetic-2N [Michalewicz (1994a)] constructed for the nonlinear transportation problem used a matrix representation for its chromosomes, a problem-specific mutation (main operator, used with probability 0.4) and arithmetical crossover (background operator, used with probability 0.05) [Schoenauer and Michalewicz (1997)]. It is hard to classify this problem: it is not an evolution strategy, since it did not use Gaussian mutation, nor did it encode any control parameters in its chromosomal structures. Clearly, it has nothing to do with genetic programming and very little (matrix representation) with evolutionary programming approaches. It is just an evolutionary computation technique aimed at particular problem.

Recently, hybridization of evolutionary algorithm with classical Gauss-Seidel based SR method has successfully been used to solve large set of linear equations; where relaxation factor,  $\omega$ , is self-adapted by using uniform adaptation technique [He et al. (2000)].

The key idea behind this hybrid algorithm that combines the SR technique and evolutionary computation techniques is to self-adapt the relaxation factor  $\omega$  which is used in the classical SR technique. For different individuals in a population, different relaxation factors are used to solve equations. The relaxation factors will be adapted based on the fitness of individuals (i.e. based on how well an individual solves the equations). Similar to many other evolutionary algorithms, this hybrid algorithm always maintains a population of approximate solution to linear equations. Each solution is represented by an individual. The initial solution is usually generated by the SR technique using an arbitrary relaxation factor  $\omega$ . The fitness of an individual is evaluated by the error estimate of the approximate solution. The relaxation factor is adapted after each generation, depending on how well an individual performs.

#### 4.5 Properties of Evolutionary Algorithms

ECs are normally classified as stochastic optimization algorithms. Within this categorization, the most important properties of ECs can be itemized as bellow:

● **Accuracy:** The accuracy describes the difference between the optimal solution and the solution obtained by the optimization method. This distinguishes between exact methods and ECs. Exact methods guarantee to find the optimum. This guarantee is paid with the complexity of the optimization method that has to be at least as high as the complexity of the problem to be solved. For example, the branch and bound algorithm is an exact method for solving linear optimization problems with integer restrictions. On the other hand, ECs do obtain only near-optimal solutions; furthermore, the accuracy of the solution often can not be predicted for these algorithms [Hashem (1999)].

● **Time-complexity:** The complexity of an EC method (or an algorithm in general) is measured by the order of the number of elementary operations independent of the input size. The input size is the amount of data necessary to specify the problem. As there are

many different problem instances having the same problem size, there are different possibilities to define the complexity. Most commonly the complexity is measured in the worst case asymptotic complexity. "Worst-case" means that the complexity of the algorithm is determined by the "hardest" problem of fixed size input. EAs have polynomial execution time allowing problems with a several order of magnitudes of higher dimensionality to be considered. Usually the absolute complexity depends upon the underlying machine model or implementation. Hence, the asymptotic complexity measures the relative increases in time with length of the problem instance and not the absolute time [Hashem (1999)].

● **Space-complexity:** The space (memory) demand of an evolutionary algorithm is an important property that may limit the applicability of the algorithm. Similar to the time-complexity measure a worst-case space demand is most commonly used [Hashem (1999)].

● **Utilization of a priori-knowledge:** It is obvious, that an algorithm that considers a priori-knowledge about the problem will outperform a method using less knowledge. The least knowledge that must be known (or must be computable) is the value of the objective function. Additional information could be used to restrict the search space, and to use symmetries in the objective function, etc. But most of the EAs perform blind search without priori-knowledge [Hashem (1999)].

● **Balance between global reliability and local refinement:** Two competing goals have to be achieved by an optimization method. First, as the global minimum can be located anywhere in the search space no parts of the region can be neglected. Global reliability, therefore, corresponds to a strategy where the search points are uniformly distributed over the whole search space. Secondly, the assumption that the chance of finding a good point in the neighborhood of a good point is higher than in the neighborhood of bad point. This assumption will surely be fulfilled for a continuous function. However, in general this assumption can not be made. Nevertheless for pragmatic reasons, most optimization methods make this assumption. This leads to a strategy that focus on particular regions or in other words that performs a local refinement of the search at "promising" points. Interestingly, ECs have incorporated a mixture of these two basic strategies [Hashem (1999)].

## 4.6 Merits and Demerits of Evolutionary Algorithms

### 4.6.1 Merits

The identified merits of ECs can be itemized as

● **Large application domain:** ECs have been applied successfully in a wide variety of application domains. One reason for this might be the intuitive concept of evolution and the modesty to the ECs with regard to the structure of the specific optimization problem. Especially the intuitive concept makes it easy to implement an algorithm that works [Hashem (1999)].

● **Suitable for complex search spaces:** It is extremely difficult to construct heuristics for complex combinatorial problems. In these problems, the choice of one variable may change the meaning or quality of another, i.e., there are high correlation between variables. ECs have been successfully applied to such instances. Obviously, the success of the ECs depends on the particular implementation and not all flavors ECs are equally well suited. As a rule of thumb, it is always good to combine an EC with available (problem-dependent) optimization heuristics [Hashem (1999)].

● **Robustness:** Robustness means that different run of an EA for the same problem yields similar results i.e. there is no great deviation in the quality of the solution. But a Monte-Carlo-based algorithm performed in average as good as a GA, the variation in the results was much higher [Hashem (1999)].

● **Easy to parallelize :** The population concept of ECs makes parallelization easy. This can reduce the execution time of the algorithm. Whole population can be divided into sub-population and each sub-population is assigned to each processor that evolves almost independently of the other populations. Furthermore, a topology of the population is defined such that each sub-population has only few “neighbours” A few individuals *migrate* between neighbours and form a loose coupling between the sub-populations [Hashem (1999)].

## 4.6.2 Demerits

The identified demerits of ECs can be itemized as

- **High computational time:** The modest demand on the objective function is paid with a relatively high computational time. This time demand not only arises from the population concept but also from the difficulty of the problems. An application specific heuristic that makes use of domain –knowledge is likely to outperform an EC [Hashem (1999)].

- **Difficult adjustment of parameters:** In every EA, a large number of parameters need to be adjusted, for example the kind of selection and crossover operator to use, the population size the probabilities of applying certain operator and the form of fitness function. Due to this fact, successful applications are often the result of a lengthy trial and error procedure whose sole purpose is to adjust the parameters of the algorithm for a particular problem class or even problem instance. Furthermore EAs are often very sensitive to the fitness function such that slight changes in the fitness function may lead to completely different behavior [Hashem (1999)].

- **Heuristic principle:** ECs don't guarantee to find the global optimum. The theoretical proofs of global convergence are useless from practical point of view as they assume infinite computation time. Under this premise, even random search can reach the global optimum. Of more importance is the fact that for most instances of EC, the accuracy of a solution obtained in a limited amount of computation time can not be predicted or guaranteed [Hashem (1999)].

## 4.7 Some Hybrid Algorithms

In this section some available hybrid algorithms will be discussed. The chosen algorithms are – Jacobi Based Uniform, Jacobi Based Time Variant and Gauss-Seidel Based Time Variant Adaptive Hybrid Algorithms.

### 4.7.1 Jacobi Based Uniform Adaptive Hybrid Algorithm

The Jacobi Based Uniform Adaptive (JBUA) evolutionary algorithm is proposed by Jamali (2004). It uses evolutionary computation techniques and Jacobi based SR technique. The JBUA hybrid evolutionary algorithm does not require a user to guess or estimate the optimal relaxation factor  $\omega$ . The algorithm initializes uniform relaxation factors in a given domain and “evolves” it. It integrates the Jacobi-based SR method with evolutionary computation techniques, which uses initialization, recombination, mutation, adaptation, and selection mechanisms. It makes better use of a population by employing different equation-solving strategies for different individuals in the population. Then these individuals can exchange information through recombination and the error is minimized by mutation and selection mechanisms.

#### The Basic Equations of Jacobi Based SR Method

Let us consider a system of linear equations

$$\sum_{j=1}^n a_{ij} x_j = b_i, \quad (i = 1, 2, \dots, n) \quad (4.7)$$

In Jacobi method by using SR technique [Engeln-Müllges, and Uhlig (1996)] Eqn. (4.7) is given by

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}} \left( b_i - \sum_{j=1}^n a_{ij} x_j^{(k)} \right), \quad (i = 1, 2, \dots, n) \quad (4.8)$$

In matrix form Eqn. (4.8) can be rewritten in matrix-vector equation as using the concepts discussed in § 2.3.1 and 2.3.3:

$$\mathbf{x}^{(k+1)} = \mathbf{H}_\omega \mathbf{x}^{(k)} + \mathbf{V}_\omega \quad (4.9)$$

where  $\mathbf{H}_\omega$ , called Jacobi iteration matrix, and  $\mathbf{V}_\omega$  are given successively by

$$\mathbf{H}_\omega = \mathbf{D}^{-1} \{ (1 - \omega) \mathbf{I} - \omega (\mathbf{L} + \mathbf{U}) \}, \quad (4.10)$$

and

$$\mathbf{V}_\omega = \omega \mathbf{D}^{-1} \mathbf{b}. \quad (4.11)$$

## The Algorithm

Similar to many other evolutionary algorithms, the JBUA hybrid algorithm always maintains a population of approximate solution to linear equations. Each solution is represented by an individual. The initial population is generated randomly from the field  $\mathfrak{R}^n$ . Different individuals use different relaxation factors. Recombination in the hybrid algorithm involves all individuals in a population. If the population size is  $N$ , then the recombination will have  $N$  parents and generates  $N$  offspring through linear combination. Mutation is achieved by performing one iteration of Jacobi based SR method as given by Eqn. (4.9). The mutation is stochastic since  $\omega$  used in the iteration is initially generated between  $\omega_L$  and  $\omega_U$  and  $\omega$  is adapted stochastically in each generation (iteration). The fitness of an individual is evaluated based on the error of an approximate solution. For example, given an approximate solution (i.e. individual)  $\tilde{\mathbf{x}}$ , its error is defined by  $\|e(\tilde{\mathbf{x}})\| = \|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{b}\|$ . The relaxation factor is adapted after each generation, depending on how well an individual performs (in term of error). The main steps of the JBUA hybrid evolutionary algorithm described as below:

### **Step 1: Initialization**

Generate an initial population of approximate solution to the system of linear Eqn. (4.7) using different arbitrary relaxation factors. Denote the initial population as

$$\mathbf{X}^{(0)} = \{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)} \dots \mathbf{x}_N^{(0)}\} \quad (4.12)$$

where each individual  $\mathbf{x}_i \in \mathfrak{R}^n$ ;  $N$  is the population size. Let 0 is assigned to  $k$ , where  $k$  is the generation counter. Also initialize relaxation factor  $\omega_1, \omega_2, \dots, \omega_N$  randomly from  $(\omega_L, \omega_U)$  where  $\omega_L$  and  $\omega_U$  are lower and upper boundary of  $\omega$ 's.

### **Step 2: Recombination**

Now generate  $\mathbf{X}^{(k+c)} = \{\mathbf{x}_1^{(k+c)}, \mathbf{x}_2^{(k+c)}, \dots, \mathbf{x}_N^{(k+c)}\}$  as an intermediate population through the following recombination:

$$\mathbf{X}^{(k+c)} = \mathbf{R}(\mathbf{X}^{(k)})' \quad (4.13)$$

where

$$\mathbf{R} = (r_{ij})_{N \times N} \quad (4.14)$$

so that

$$\sum_{j=1}^N r_{ij} = 1 \text{ and } r_{ij} \geq 0 \text{ for } 1 \leq i \leq N$$

i.e.  $\mathbf{R}$  is a stochastic matrix [Kriyszg (1993)]. Superscript “ ’ ” is a transposed operator. Note that the symbol  $c$ , as a superscript, is used just as an indicator of crossover.

### Step 3: Mutation

Then generate the next intermediate population  $\mathbf{X}^{(k+m)}$  from  $\mathbf{X}^{(k+c)}$  as follows:

For each individual  $\mathbf{x}_i^{(k+c)}$  ( $1 \leq i \leq N$ ) in population  $\mathbf{X}^{(k+c)}$  produces an offspring according to Eqn. (4.9) as

$$\mathbf{x}_i^{(k+m)} = \mathbf{H}_{\omega_i} \mathbf{x}_i^{(k+c)} + \mathbf{V}_{\omega_i}, \quad i = 1, 2, \dots, N. \quad (4.15)$$

where  $\mathbf{H}_{\omega_i}$  is called Jacobi iteration matrix corresponding  $\omega_i$  and given by

$$\mathbf{H}_{\omega_i} = \mathbf{D}^{-1} \{ (1 - \omega_i) \mathbf{I} - \omega_i (\mathbf{L} + \mathbf{U}) \}, \quad (4.16)$$

and

$$\mathbf{V}_{\omega_i} = \omega_i \mathbf{D}^{-1} \mathbf{b}. \quad (4.17)$$

Here  $\omega_i$  is denoted as relaxation factor of the  $i$ th individual and  $\mathbf{x}_i^{(k+m)}$  is denoted as  $i$ th (mutated) offspring, so that only one iteration is carried out for each mutation. Note that the symbol  $m$ , as a superscript, is used just as an indicator of mutation.

### Step 4: Adaptation

Let  $\mathbf{x}$  and  $\mathbf{y}$  be two offspring individuals corresponding to relaxation factors  $\omega_x$  and  $\omega_y$  and  $\|e(\mathbf{x})\|$  and  $\|e(\mathbf{y})\|$  are their corresponding errors (fitness value). Then the relaxation factors  $\omega_x$  and  $\omega_y$  are adapted as follows:

(a) If  $\|e(\mathbf{x})\| > \|e(\mathbf{y})\|$ ,

(i) then move  $\omega_x$  toward  $\omega_y$  (i.e.  $\omega_x$  is adapted to  $\omega_x^m$ ) by using



$$\omega_x^m = (0.5 + p_x)(\omega_x + \omega_y) \quad (4.18)$$

$$\text{where } p_x \in (-0.01, 0.01) \quad (4.19)$$

And (ii) move  $\omega_y$  away from  $\omega_x$  (i.e.  $\omega_y$  is adapted to  $\omega_x^m$ ) by using

$$\omega_y^m = \begin{cases} \omega_y + p_y(\omega_U - \omega_y), & \text{when } \omega_y > \omega_x \\ \omega_y + p_y(\omega_L - \omega_y), & \text{when } \omega_y < \omega_x \end{cases} \quad (4.20)$$

$$\text{where } p_y \in (0.008, 0.012) \quad (4.21)$$

(b) If  $\|e(\mathbf{x})\| < \|e(\mathbf{y})\|$ , adapt  $\omega_x$  and  $\omega_y$  in the same way as above but reverse the order of  $\omega_x^m$  and  $\omega_y^m$ .

(c) If  $\|e(\mathbf{x})\| = \|e(\mathbf{y})\|$ , no adaptation. So that

$$\omega_x^m = \omega_x \text{ and } \omega_y^m = \omega_y.$$

Here uniform adaptation technique is used to adapting the relaxation factors [He et. al. (2000)].

### Step 5: Selection and Reproduction

The best  $N/2$  individuals in population  $\mathbf{X}^{(k+m)}$  will reproduce (i.e. each individual generates two offspring), and then form the next generation  $\mathbf{X}^{(k+1)}$  of  $N$  individuals.

### Step 6: Halt

If the error of the population  $\|e(\mathbf{X})\| = \min\{\|e(\tilde{\mathbf{x}})\|: \tilde{\mathbf{x}} \in \mathbf{X}\}$  is less than a given threshold  $\eta$  then the algorithm terminates; otherwise, go to **Step -2**.

## 4.7.2 Jacobi Based Time Variant Adaptive Hybrid Algorithm

The Jacobi Based Time Variant Adaptive (JBTV) hybrid algorithm is proposed by Jamali et al. (2004). In that algorithm Jacobi based SR method, evolutionary computation techniques and time variant adaptation techniques are used. That also does not require a user to guess or estimate the optimal relaxation factor  $\omega$ . The algorithm initializes uniform relaxation factors in a given domain and “evolves” it. The proposed algorithm integrates the Jacobi-based SR method with evolutionary computation techniques, which uses

recombination, mutation and selection mechanisms. It makes better use of a population by employing different equation-solving strategies for different individuals in the population. Then these individuals can exchange information through recombination and the error is minimized by mutation and selection mechanisms.

The main steps of the JBTVA hybrid evolutionary algorithm are Initialization, Recombination, Mutation, Adaptation, Selection, Reproduction and Halt respectively. Initialization, Recombination, Selection, adaptation mechanisms and Halt criteria of this proposed algorithm is same as those of Gauss-Seidel Based Time Variant Adaptive algorithm and is discussed in the next subsection. And Mutation mechanism is same as that of JBUA algorithm (see § 4.7.1).

### 4.7.3 Gauss-Seidel Based Time Variant Adaptive Hybrid Algorithm

The Gauss-Seidel based Time-variant adaptive (GSBTVA) hybrid evolutionary algorithm is the hybridization of evolutionary algorithm with classical Gauss-Seidel based SR method in which a time-variant adaptation (TVA) technique is used instead of uniform adaptation (UA). In sequel, it is described here elaborately.

#### The Basic Equations of Gauss-Seidel Based SR Method

Here also the system of linear Eqn. is taken as

$$\sum_{j=1}^n a_{ij} x_j = b_i, \quad (i = 1, 2, \dots, n) \quad (4.22)$$

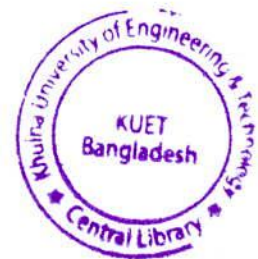
In Gauss-Seidel based SR method [Engeln-Müllges, and Uhlig (1996)] Eqn. (4.22) is given by

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left( - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} + b_i \right), \quad (4.23)$$

$$i = 1, 2, \dots, n \text{ and } k = 0, 1, \dots$$

In matrix form Eqn. (4.23) can be rewritten in matrix-vector equation as using the concepts discussed in § 2.3.2 and 2.3.3:

$$\mathbf{x}^{(k+1)} = \mathbf{H}_\omega \mathbf{x}^{(k)} + \mathbf{V}_\omega, \quad (4.24)$$



where  $\mathbf{H}_\omega$  is called Gauss-Seidel iteration matrix and given by

$$\mathbf{H}_\omega = (\mathbf{I} + \omega \mathbf{D}^{-1}\mathbf{L})^{-1} \{ (1 - \omega)\mathbf{I} - \omega \mathbf{D}^{-1}\mathbf{U} \} \quad (4.25)$$

and

$$\mathbf{V}_\omega = \omega (\mathbf{I} + \omega \mathbf{D}^{-1}\mathbf{L})^{-1} \mathbf{D}^{-1}\mathbf{b} \quad (4.26)$$

### **The Algorithm**

Similar to many other evolutionary algorithms, the GSBTVA hybrid algorithm also always maintains a population of approximate solution to linear equations. The initialization of population and recombination mechanisms of this algorithm is same as those of JBUA algorithm. Mutation is achieved by performing one iteration of Gauss-Seidel based SR method as given by Eqn. (4.24). The mutation is stochastic since  $\omega$  used in each mutation step, is adapted stochastically in each generation (iteration). The fitness of an individual is evaluated based on the error of an approximate solution. The relaxation factor is adapted after each generation, depending on how well an individual performs (in term of error). The main steps of the GSBTVA hybrid evolutionary algorithm described as below:

#### **Step 1: Initialization**

Generate, randomly from  $\mathfrak{R}^n$ , an initial population of approximate solutions to the linear Eqn. (4.7) using different relaxation factor for each individual of the population. Denote the initial population as  $\mathbf{X}^{(0)} = \{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \dots, \mathbf{x}_N^{(0)}\}$  where  $N$  is the population size. Let  $0$  is assigned to  $k$ , where  $k$  is the generation counter. And initialize corresponding relaxation factor  $\omega$  as:

$$\omega_i = \begin{cases} \omega_L + \frac{d}{2} & \text{for } i = 1 \\ \omega_{i-1} + d & \text{for } 1 < i \leq N \end{cases} \quad (4.27)$$

where  $d = \frac{\omega_U - \omega_L}{N}$

### Step 2: Recombination

Now generate  $\mathbf{X}^{(k+c)} = \{\mathbf{x}_1^{(k+c)}, \mathbf{x}_2^{(k+c)}, \dots, \mathbf{x}_N^{(k+c)}\}$  as an intermediate population through the following recombination:

$$\mathbf{X}^{(k+c)} = \mathbf{R}(\mathbf{X}^{(k)})' \quad (4.28)$$

where  $\mathbf{R}$  is a stochastic matrix. Superscript " ' " is a transposed operator.

### Step 3: Mutation

Then generate the next intermediate population  $\mathbf{X}^{(k+m)}$  from  $\mathbf{X}^{(k+c)}$  as follows: For each individual  $\mathbf{x}_i^{(k+c)}$  ( $1 \leq i \leq N$ ) in population  $\mathbf{X}^{(k+c)}$  produces an offspring according to (see Eqn. (4.24))

$$\mathbf{x}_i^{(k+m)} = \mathbf{H}_{\omega_i} \mathbf{x}_i^{(k+c)} + \mathbf{V}_{\omega_i}, \quad i = 1, 2, \dots, N. \quad (4.29)$$

where  $\mathbf{H}_{\omega_i}$  is called Gauss-Seidel iteration matrix corresponding  $\omega_i$  and given by

$$\mathbf{H}_{\omega_i} = (\mathbf{I} + \omega_i \mathbf{D}^{-1} \mathbf{L})^{-1} \{(\mathbf{I} - \omega_i) \mathbf{I} - \omega_i \mathbf{D}^{-1} \mathbf{U}\} \quad (4.30)$$

and

$$\mathbf{V}_{\omega_i} = \omega_i (\mathbf{I} + \omega_i \mathbf{D}^{-1} \mathbf{L})^{-1} \mathbf{D}^{-1} \mathbf{b} \quad (4.31)$$

Here  $\omega_i$  is denoted as relaxation factor of the  $i$ th individual and  $\mathbf{x}_i^{(k+m)}$  is denoted as  $i$ th (mutated) offspring, so that only one iteration is carried out for each mutation.

### Step 4: Adaptation

Let  $\mathbf{x}^{(k+m)}$  and  $\mathbf{y}^{(k+m)}$  be two offspring individuals corresponding to relaxation factors  $\omega_x$  and  $\omega_y$  and  $\|e(\mathbf{x}^m)\|$  and  $\|e(\mathbf{y}^m)\|$  are their corresponding errors (fitness value). Then the relaxation factors  $\omega_x$  and  $\omega_y$  are adapted as follows:

(d) If  $\|e(\mathbf{x}^m)\| > \|e(\mathbf{y}^m)\|$ , (i) then move  $\omega_x$  toward  $\omega_y$  by using

$$\omega_x^m = (0.5 + p_x)(\omega_x + \omega_y) \quad (4.32)$$

and (ii) move  $\omega_y$  away from  $\omega_x$  using

$$\omega_y^m = \begin{cases} \omega_y + p_y(\omega_U - \omega_y), & \text{when } \omega_y > \omega_x \\ \omega_y + p_y(\omega_L - \omega_y), & \text{when } \omega_y < \omega_x \end{cases} \quad (4.33)$$

where  $p_x = E_x \times N(0, 0.25) \times T_\omega$ , and  $p_y = E_y \times |N(0, 0.25)| \times T_\omega$ , as defined earlier (Eqn. (4.3) and Eqn. (4.4)).

- (e) If  $\|e(\mathbf{x}^m)\| < \|e(\mathbf{y}^m)\|$ , then adapt  $\omega_x$  and  $\omega_y$  in the same way as above but reverse the order of  $\omega_x^m$  and  $\omega_y^m$ .
- (f) If  $\|e(\mathbf{x}^m)\| = \|e(\mathbf{y}^m)\|$ , no adaptation. So that  $\omega_x^m = \omega_x$  and  $\omega_y^m = \omega_y$ .

### Step 5: Selection and Reproduction

Selection mechanism is same as that of JBUA algorithm. That is, select the best  $N/2$  offspring individuals according to their fitness values (errors). Then reproduce of the above selected offspring (i.e. each parent's individual generates two offspring). Then form the next generation of  $N$  individuals.

### Step 6: Termination

If  $\min\{\|e(\mathbf{z})\| : \mathbf{z} \in \mathbf{X}\} < \eta$  (Threshold error), then stop the algorithm and get unique solution. If  $\min\{\|e(\mathbf{z})\| : \mathbf{z} \in \mathbf{X}\} \rightarrow \infty$ , then stop the algorithm but fail to get any solution. Otherwise go to Step 2.

---

## Evaluation of Uniform Adaptive Hybrid Evolutionary Algorithms for Solving a Set of Linear Equations

### 5.1 Introduction

Large set of linear equations frequently arise directly or indirectly in the real world problems and though there are many classical methods available for solving them, scientists still have their keen interest to find out the methods, which converge rapidly and efficiently.

For solving large set of linear equations, especially for sparse and structured coefficients, iterative methods are preferable, as iterative methods are unaffected by round off errors [Gerald and Wheatley (1998)]. The rate of convergence of the well-known classical numerical iterative methods (the Jacobi and the Gauss-Seidel method) is very slow and can be accelerated by using successive relaxation (SR) technique [Young (1971) and Engeln-Mullges and Uhlig (1996)]. But the speed of convergence depends on the relaxation factor  $\omega$  ( $0 < \omega < 2$ ) and SR technique is very much sensitive to the relaxation factor [Hagaman and Young (1981), Stocr and Bulirsch (1991)]. Moreover, it is often very difficult to estimate the optimal relaxation factor, which is a key parameter of the SR technique [Hagaman and Young (1981), Gourdin and Boumahrat (1996)].

The evolutionary algorithms (EA) are developed from some natural phenomena: genetic inheritance and Darwinian strife for survival [Bäck et al. (1997), Bäck and Schwefel ((1993) and Schocnauer and Michalewicz (1997)]. Generally, most of the works on EA can be classified as evolutionary optimization (either numerical or combinatorial) or evolutionary learning [Hashem (1999), Watanabe and Hashem (2004), Michalewicz (1994), Michalewicz and Attia (1996), Salomon and Van Hemmen (1996) and Jun et al. (2000)]. Recently, Jamali et al. (2003) has developed Gauss-Seidel based uniform adaptive

(GSBUA) hybrid evolutionary algorithm and Jacobi based uniform adaptive (JBUA) hybrid evolutionary algorithm for solving large set of linear equations. In these algorithms both crossover and mutations operations are present. Furthermore, Gauss-Seidel based Time variant adaptive (GSBTVA) hybrid evolutionary algorithm [Jamali et al. (2004b)] and Jacobi based Time variant adaptive (JBTVA) hybrid evolutionary algorithm [Jamali et al. (2004a)] have been developed for solving large set of linear equations by integrating classical numerical methods with Time variant adaptive (TVA) evolutionary computation techniques. In the later two algorithms, both crossover and mutations operations are also present. The uniform adaptation or time variant adaptation techniques are introduced for self-adaptation of relaxation factor. The idea of self-adaptation was also applied in many different fields [Salomon and Van Hemmen (1996), Bäck (1997) and Beyer and Deb (2001)].

## 5.2 The Existing Uniform Adaptive Hybrid Evolutionary Algorithms

The main aim of the hybridization of the classical SR methods with the evolutionary computation techniques is to self-adapt the relaxation factor used in the classical SR technique. The relaxation factors are adapted on the basis of the fitness of individuals (i.e. how well an individual solves the equations). Similar to many other evolutionary algorithms, the hybrid algorithm always maintains a population of approximate solutions to the linear equations. Each solution is represented by an individual. The initial population is generated randomly from the field  $\mathcal{R}^n$ . Different individuals use different relaxation factors. Crossover in the hybrid algorithm involves all individuals in a population. If the population is of size  $N$ , then the crossover will have  $N$  parents and generates  $N$  offspring through linear combination. Mutation is achieved by performing one iteration of classical (Gauss-Seidel or Jacobi) method with SR technique. The mutation is stochastic since  $\omega$ , used in the iteration are initially determined between  $\omega_L (=0)$  and  $\omega_U (=2)$ , is adapted stochastically in each generation. The fitness of an individual is evaluated on the basis of the error of an approximate solution. For example, given an approximate solution (i.e., an individual)  $\mathbf{z}$ , its error is defined by  $\|e(\mathbf{z})\| = \|\mathbf{Az} - \mathbf{b}\|$ . The relaxation factors are adapted after each generation, depending on how well an individual performs (in terms of the error). The main steps of the existing JBUA and the GSBUA hybrid algorithms are – Initialization, Crossover, Mutation, Adaptation and Selection mechanism [Jamali et al.

(2003)]. The pseudo-code structure of the existing hybrid evolutionary algorithms [Jamali et al. (2003)] is given bellow:

**Algorithm\_JBUA/GSBUA()**

**begin**

$k \leftarrow 0$  ; /\* Initialize the generation counter \*/

Initialize population:  $\mathbf{X}^{(0)} = \{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)} \dots \mathbf{x}_N^{(0)}\}$  ;

/\* Here  $\mathbf{x}_i^{(k)}$   $i \Rightarrow$   $i$ -th individual at  $k$ -th generation \*/

Initialize relaxation factors:  $\omega_i \in (0, 2)$  randomly

Evaluate population:  $\|e(\mathbf{X})\| = \{\|e(\mathbf{z})\| : \mathbf{z} \in \mathbf{X}\}$  ;

**While (not termination-condition) do**

**begin**

Select individuals for reproduction:

Apply operators:

Crossover:  $\mathbf{X}^{(k+c)} = \mathbf{R}(\mathbf{X}^{(k)})'$  ;

/\*  $\mathbf{R}$  is stochastic matrix & Superscript  $c$  indicates Crossover \*/

Mutation:  $\mathbf{x}_i^{(k+m)} = \mathbf{H}_{q(\omega_i)} \mathbf{x}_i^{(k+c)} + \mathbf{V}_{q(\omega_i)}$  ;

/\* where  $q \in \{j, g\}$ , “ $j$ ” indicate Jacobi based method and “ $g$ ” indicate Gauss-Seidel based method \*/

Evaluate newborn offspring:  $\|e(\mathbf{X}^{(k+m)})\| = \{\|e(\tilde{\mathbf{x}}^{(k+m)})\| : \tilde{\mathbf{x}}^{(k+m)} \in \mathbf{X}^{(k+m)}\}$  ;

Adaptation of  $\omega$  :  $\omega_x = f_x(\omega_x, \omega_y, p_x)$  &

$$\omega_y = f_y(\omega_x, \omega_y, p_y) ;$$

/\*  $p_x$  and  $p_y$  are adaptive probability functions \*/

Selection and reproduction:  $\mathbf{X}^{(k+1)} = \zeta(\mathbf{X}^{(k+m)})$  ;

$k \leftarrow k + 1$  ;

/\* Increase the generation counter \*/

**end**

**end**



As the adaptation and the selection are the main characteristic mechanisms of the existing hybrid algorithms (as well as proposed modified algorithm also), so we have described them in brief bellow:

**Adaptation:**

Let  $\mathbf{X}^{(k+m)}$  and  $\mathbf{y}^{(k+m)}$  be two offspring individuals with relaxation factors  $\omega_x$  and  $\omega_y$  and with errors (fitness values)  $\|e(\mathbf{x}^m)\|$  and  $\|e(\mathbf{y}^m)\|$  respectively. Then the relaxation factors  $\omega_x$  and  $\omega_y$  are adapted as follows:

If  $\|e(\mathbf{x}^m)\| > \|e(\mathbf{y}^m)\|$ ,

(i) then  $\omega_x$  is moved toward  $\omega_y$  by setting

$$\omega_x^m = (0.5 + p_x)(\omega_x + \omega_y) \quad (5.1)$$

and

(ii)  $\omega_y$  is moved away from  $\omega_x$  by setting

$$\omega_y^m = \begin{cases} \omega_y + p_y(\omega_U - \omega_y), & \text{when } \omega_y > \omega_x \\ \omega_y + p_y(\omega_L - \omega_y), & \text{when } \omega_y < \omega_x \end{cases} \quad (5.2)$$

where

$$p_x \in (-0.01, 0.01) \text{ and } p_y \in (0.008, 0.012),$$

are the uniform adaptive (UA) parameters of  $\omega_x$  and  $\omega_y$  respectively. Note that  $\omega_x^m$  and  $\omega_y^m$  are adapted relaxation factors correspond to  $\omega_x$  and  $\omega_y$  respectively.

If  $\|e(\mathbf{x}^m)\| < \|e(\mathbf{y}^m)\|$ , then  $\omega_x$  and  $\omega_y$  are adapted in the same way as above but in the reverse order of  $\omega_x^m$  and  $\omega_y^m$ .

If  $\|e(\mathbf{x}^m)\| = \|e(\mathbf{y}^m)\|$ , no adaptation will take place i.e.

$$\omega_x^m = \omega_x \text{ and } \omega_y^m = \omega_y.$$

### **Selection and Reproduction:**

The best  $N/2$  offspring individuals are selected according to their fitness values (errors). Then the selected offspring are reproduced (i.e. each parent individuals generate two offspring). Thus the next generation of  $N$  individuals is formed.

### **5.3 Necessity of Crossover**

Here have tried to investigate the necessity of the crossover operation available in the existing uniform adaptive hybrid evolutionary algorithms [Jun et al. (2000) and Jamali et al. (2003)]. For the purpose two modified uniform adaptive hybrid evolutionary algorithms (i.e. modified GSBUA and the modified JBUA algorithms) are proposed to solve large set of linear equations. The proposed modified hybrid algorithms are modified from the existing GSBUA and the JBUA algorithms and contain all the evolutionary operations available in the existing algorithms except crossover operation. The proposed modified hybrid algorithms initialize random relaxation factors in a given domain and “evolve” it by uniform adaptation technique as well. The main mechanisms of the proposed modified algorithms are initialization, mutation, uniform adaptation, and selection mechanisms (i.e. crossover operation is absent). It makes better use of a population by employing different equation-solving strategies for different individuals in the population. The errors are minimized by mutation and selection mechanisms. The investigation is done by comparing the proposed modified hybrid algorithms containing only mutation operation with existing hybrid algorithms containing both crossover and mutation operations.

### **5.4 The Proposed Modified Hybrid Evolutionary Algorithms**

The key idea behind the proposition of the modified algorithms (Modified Jacobi Based Uniform Adaptive (MJBUA) hybrid evolutionary algorithm and the Modified Gauss-Seidel Based Uniform Adaptive (MGSBUA) hybrid evolutionary algorithm) is to examine the necessity of crossover operation for solving linear equations. So the proposed modified hybrid evolutionary algorithms contains all steps of the JBUA and GSBUA hybrid evolutionary algorithms except the step – crossover. And we are not repeating the pseudo-code structures of the both modified uniform hybrid evolutionary algorithms here, as they will be same as that of JBUA and GSBUA except crossover portion.

The system of  $n$  linear equations with  $n$  unknowns  $x_1, x_2, \dots, x_n$  can be written as

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad (i = 1, 2, \dots, n) \text{ or equivalently, in matrix form}$$

$$\mathbf{Ax} = \mathbf{b} \tag{5.3}$$

In order to evaluate the effectiveness of the proposed MJBUA and MGSBUA hybrid algorithm, a number of numerical experiments have been carried out to solve the Eqn. (5.3) the following settings were valid for all the experiments:

Dimension of unknown variable,  $n = 200$

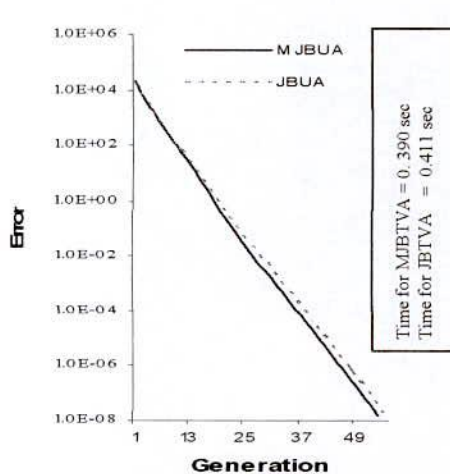
Population size,  $N=2$

Boundary of relaxation factor,  $(\omega_L, \omega_U) = (0, 2)$

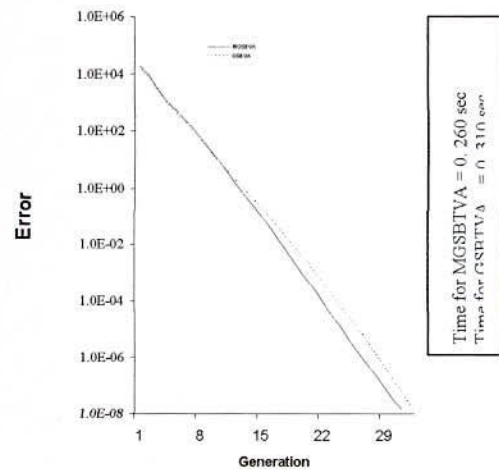
Initial domain from which each individual  $\mathbf{x}$  of population  $\mathbf{X}$  be initialized in  $(-30, 30)$

Threshold error,  $\eta = 10^{-7}$

Also the relaxation factors and the stochastic matrix  $\mathbf{R}$  are generated randomly.



**Figure 1:** Comparison between MJBUA and JBUA algorithms



**Figure 2:** Comparison between MGSBUA and GSBUA algorithms

The first problem (problem  $P_1$  in Table 1 & 2) was set by considering  $a_{ii} \in (100,200)$ ;  $a_{ij} \in (-10,10)$ ;  $b_i \in (100,200)$ ,  $i, j = 1, \dots, n$ . A single set of parameters was generated randomly from the above-mentioned problem and the following two experiments were carried out.

The problem was solved with an error smaller than the threshold error  $10^{-7}$ .

In the first experiment, the comparison between the JBUA and the proposed MJBUA had been made. Figure 1 shows the numerical results of this experiment. From this experiment, two important observations came out. Firstly the proposed MJBUA algorithm is comparable with the JBUA algorithm in terms of number of generation. Secondly the proposed MJBUA algorithm required less amount of time than that of JBUA algorithm.

In the second experiment, the comparison between the GSBUA and the proposed MGSBUA had been made. **Figure 2** shows the numerical results of this experiment. Again two observations came out – (i) the proposed MGSBUA algorithm is comparable with the GSBUA algorithm in terms of number of generation and (ii) the proposed MGSBUA algorithm required less amount of time than that of GSBUA algorithm.

**Table 1:** Comparison between the JBUA and the proposed MJBUA hybrid algorithms for several randomly generated test problems

Label of Test Problems	Domain of the elements of the coefficient matrix <b>A</b> & the right side constant vector <b>b</b> of the test Problems	MJBUA Alg.		JBUA Alg.	
		Generation (Elapsed)	Elapse time (mili sec)	Generation (elapsed)	Elapse time (mili sec)
$P_1$	$a_{ii} \in (100,200)$ ; $a_{ij} \in (-10,10)$ ; $b_i \in (100,200)$	62	390	60	411
$P_2$	$a_{ii} \in (1,400)$ ; $a_{ij} \in (-4,4)$ ; $b_i = 100$	171	1016	168	1320
$P_3$	$a_{ii} \in (-50,50)$ ; $a_{ij} \in (-1,1)$ ; $b_i \in (-1,1)$	37	609	37	640
$P_4$	$a_{ii} = 100$ ; $a_{ij} \in (-1,1)$ ; $b_i \in (-100,100)$	42	539	44	582
$P_5$	$a_{ii} = 50$ ; $a_{ij} \in (-10,10)$ ; $b_i \in (-5,5)$	10	156	11	172
$P_6$	$a_{ii} = 50$ ; $a_{ij} = (-1,1)$ ; $b_i = 2$	10	188	11	219
$P_7$	$a_{ii} = 20i$ ; $a_{ij} = (100-j)/20$ ; $b_i = 10i$	79	1282	82	1312
$P_8$	$a_{ii} = 20n$ ; $a_{ij} = j$ ; $b_i = i$	Not converged	--	Not converged	-----
$P_9$	$a_{ii} \in (-20,200)$ ; $a_{ij} \in (-2,3)$ ; $b_i \in (-2,3)$	475	7406	489	7641
$P_{10}$	$a_{ii} = 40$ ; $a_{ij} \in (-4,4)$ ; $b_i = 200$	73	1170	74	1200
$P_{11}$	$a_{ii} \in (-50,50)$ ; $a_{ij} \in (-1,1)$ ; $b_i \in (-1,1)$	34	530	33	560

Table 1 and 2 represent eleven test problems in each labeled from  $P_1$  to  $P_{11}$ , with dimension,  $n = 200$ . For each test problem  $P_i$ ;  $i = 1, 2, \dots, 11$ , the coefficient matrix  $A$  and constant vector  $b$  all were generated uniformly and randomly within given domains. **Table 1** shows the comparison between the number of generations (iterations) of the JBUA and the proposed MJBUA hybrid algorithms with respect to the considered threshold error,  $\eta$ . One observation can be made immediately from this table that the proposed MJBUA hybrid algorithm is comparable with the existing JBUA hybrid algorithm for all the problems. Another observation is that the proposed MJBUA required less amount of time than that of JBUA for all the cases.

**Table 2:** Comparison between the GSBUA and the proposed MGSBUA hybrid algorithms for several randomly generated test problems

Label of Test Problems	Domain of the elements of the coefficient matrix $A$ & the right side constant vector $b$ of the test Problems	MGSBUA Alg.		GSBUA Alg.	
		Generation (Elapsed)	Elapse time (mili sec)	Generation (elapsed)	Elapse time (mili sec)
$P_1$	$a_{ii} \in (100, 200)$ ; $a_{ij} \in (-10, 10)$ ; $b_i \in (100, 200)$	34	260	35	310
$P_2$	$a_{ii} \in (1, 400)$ ; $a_{ij} \in (-4, 4)$ ; $b_i = 100$	92	710	93	734
$P_3$	$a_{ii} \in (-50, 50)$ ; $a_{ij} \in (-1, 1)$ ; $b_i \in (-1, 1)$	44	625	45	656
$P_4$	$a_{ii} = 100$ ; $a_{ij} \in (-1, 1)$ ; $b_i \in (-100, 100)$	40	495	41	591
$P_5$	$a_{ii} = 50$ ; $a_{ij} \in (-10, 10)$ ; $b_i \in (-5, 5)$	8	141	7	156
$P_6$	$a_{ii} = 50$ ; $a_{ij} = (-1, 1)$ ; $b_i = 2$	13	204	13	218
$P_7$	$a_{ii} = 20i$ ; $a_{ij} = (100-j)/20$ ; $b_i = 10i$	112	1468	117	1578
$P_8$	$a_{ij} = 20n$ ; $a_{ij} = j$ ; $b_i = i$	60	859	58	938
$P_9$	$a_{ii} \in (-20, 200)$ ; $a_{ij} \in (-2, 3)$ ; $b_i \in (-2, 3)$	554	7141	559	7219
$P_{10}$	$a_{ii} = 40$ ; $a_{ij} \in (-4, 4)$ ; $b_i = 200$	107	1422	111	1500
$P_{11}$	$a_{ii} \in (-50, 50)$ ; $a_{ij} \in (-1, 1)$ ; $b_i \in (-1, 1)$	13	111	13	145

**Table 2** shows the comparison between the number of generations (iterations) of the GSBUA and the proposed MGSBUA hybrid algorithms with respect to the considered threshold error,  $\eta$ . One observation can be made immediately from this table that the proposed MGSBUA hybrid algorithm is comparable with the GSBUA hybrid algorithm for all the problems. Another observation is that the proposed MGSBUA required less amount of time than that of GSBUA for all the cases.

**Table 3:** Effect of dimension of coefficient matrix on both MGSBUA and GSBUA algorithms

Dimension ( $n$ )	MGSBUA Alg.		GSBUA Alg.	
	Generation (Elapsed)	Elapse time (mili sec)	Generation (elapsed)	Elapse time (mili sec)
200	168	2200	169	2266
100	34	171	46	293
50	20	16	20	31
25	19	2	19	3
10	15	0	15	0

**Table 4:** Effect of dimension of coefficient matrix on both MJBUA and JBUA algorithms

Value of $n$	MJBUA Alg.		JBUA Alg.	
	Generation (Elapsed)	Elapse time (mili sec)	Generation (elapsed)	Elapse time (mili sec)
200	162	2656	170	2828
100	32	140	31	156
50	19	15	19	31
25	15	1	15	2
10	12	0	12	0

In order to study the effect of the dimension  $n$  of the coefficient matrix  $\mathbf{A}$  on the modified hybrid evolutionary algorithms, we set problems with  $a_{ii} = n$ ,  $a_{ij} \in (-n/4, n/4)$ ,  $b = n$ ; where the value of  $n$  were set at 200, 100, 50, 25 and 10. For each value of  $n$  the problem was generated randomly within proposed domain. Table 3 and 4 show the generation history for both the modified hybrid algorithms and their existing counter parts for the above problems. From the tables, it is observed that the effect of dimensions of the coefficient matrix on MGSBUA and GSBUA as well as both MJBUA and JBUA hybrid algorithms are almost same.

It is to be mentioned here that a total of ten independent runs with different sample paths were conducted. The average results are reported here. Also for all the experiments, the times were measured in the same environment.

## 5.5 Concluding Remarks

MJBUA and MGSBUA are the two modified uniform adaptive hybrid evolutionary algorithms, which have been proposed for solving large set of linear equations which do not contain the crossover operation. By omitting the crossover operations from existing JBUA and GSBUA (Jun et al. (2000) and Jamali et al. (2003)) the proposed MJBUA and MGSBUA hybrid algorithms respectively have been developed. The necessity of the

crossover operation is investigated by comparing the performance of the proposed Algorithms with that of the JBUA and the GSBUA hybrid algorithms respectively. The preliminary investigation has showed that both the proposed MJBUA and MGSBUA hybrid algorithms are comparable in terms of generation (iteration) with the JBUA and GSBUA respectively. Also the both proposed MJBUA and MGSBUA hybrid algorithms required less amount of time than the JBUA and the GSBUA hybrid algorithm respectively. Furthermore since proposed modified hybrid algorithms have no crossover operation, so they require less memory allocation and less computational effort to solve the problems. Moreover, the proposed modified hybrid algorithms are also very simple and easier to implement both in sequential and parallel computing environments. It may thus conclude that for solving set of linear equations by uniform adaptive hybrid evolutionary algorithms, crossover is a needless operation.

**Use of Proposed Hybrid Evolutionary Algorithms for Solving Partial Differential Equations.**

**6.1 Introduction**

Solution of Laplace's equation  $\nabla^2\phi = 0$  will be performed using one of the Proposed Modified Hybrid Evolutionary Algorithms. The problem will also be solved by Gauss-Seidel method. Both the solution will be compared with the analytical solution and the RMS error will be calculated. A sample Poisson's equation will also be discretized to transfer into a set of linear equations. That set will also be solved by one of the proposed Algorithm, MGSBUA Algorithm.

**6.2 Solution of Laplace's Equation**

Let us consider a steel plate of size 36cm  $\times$  36cm. The upper side is held at 100<sup>0</sup>C and the other sides are held at 0<sup>0</sup>C. Now we are to find the steady state temperature at interior points assuming a grid size of 3cm  $\times$  3cm. Here we obtain 13  $\times$  13 grid points on the plate. There will be 11  $\times$  11 grid points inside the plate where the temperature distribution is to be calculated.

Initial values may be obtained by either taking diagonal average or cross average of the adjoining four points. They are associated with the point patterns



respectively, and may be more vividly pictured by the stencils





$$\phi(x, y) = \sum_{n=1}^{\infty} \frac{200[1 - (-1)^n]}{n\pi \cdot \sinh \frac{n\pi b}{l}} \cdot \sinh \frac{n\pi y}{l} \cdot \sin \frac{n\pi x}{l} \quad (6.3)$$

The presence of the hyperbolic function is trouble creating. If we want to take more terms in the summation the  $\sinh \frac{n\pi y}{l}$  becomes larger and larger. So we restrict ourselves up to 50 terms only and the output result (6.3) is taken up to 6 decimal places only.

The following matrix represents the analytical result.

49.240183	68.930570	77.099542	80.999858	82.844247	83.394362	82.844247	80.999858	77.099542	68.930570	49.240183
28.029296	46.964189	57.851131	63.865378	66.902297	67.832005	66.902297	63.865378	57.851131	46.964189	28.029296
18.327701	33.042382	43.202833	49.544496	52.971909	54.052922	52.971909	49.544496	43.202833	33.042382	18.327701
12.850216	23.937285	32.366485	38.071833	41.323675	42.375842	41.323675	38.071833	32.366485	23.937285	12.850216
9.317473	17.639133	24.302695	29.043628	31.846087	32.769986	31.846087	29.043628	24.302695	17.639133	9.317473
6.846087	13.071833	18.202833	21.964189	24.240183	25.000000	24.240183	21.964189	18.202833	13.071833	6.846087
5.022558	9.635470	13.499968	16.383616	18.153913	18.749648	18.153913	16.383616	13.499968	9.635470	5.022558
3.621661	6.966553	9.795235	11.928167	13.249081	13.695779	13.249081	11.928167	9.795235	6.966553	3.621661
2.504230	4.824423	6.797167	8.293784	9.225429	9.541412	9.225429	8.293784	6.797167	4.824423	2.504230
1.574475	3.035811	4.282064	5.230785	5.823099	6.024330	5.823099	5.230785	4.282064	3.035811	1.574475
0.759817	1.465658	2.068527	2.528265	2.815722	2.913465	2.815722	2.528265	2.068527	1.465658	0.759817

We have solved the system of equations represented by (6.2) by Gauss-Seidel method and obtained the following result, which is obtained after 109 iteration.

49.231949	68.195547	76.616603	80.705450	82.638024	83.213237	82.638393	80.706113	76.617416	68.196314	49.232446
28.732703	46.934371	57.566362	63.568255	66.634544	67.577638	66.635234	63.569493	57.567880	46.935803	28.733631
18.765227	33.244032	43.147698	49.368342	52.756009	53.829238	52.756951	49.370033	43.149771	33.245989	18.766496
13.085121	24.130310	32.413933	38.003525	41.194115	42.228490	41.195230	38.005525	32.416385	24.132625	13.086622
9.446029	17.779832	24.376316	29.040100	31.790920	32.697779	31.792121	29.042254	24.378958	17.782327	9.447646
6.920300	13.168423	18.273604	21.992119	24.234261	24.982071	24.235462	21.994274	18.276246	13.170917	6.921917
5.067856	9.701656	3.559696	16.422913	18.174423	18.763185	18.175544	16.424924	13.562160	9.703983	5.069365
3.650479	7.012189	9.842539	11.967582	13.279575	13.722861	13.280546	11.969323	9.844674	7.014204	3.651785
2.522719	4.855369	6.832300	8.327100	9.255296	9.569928	9.256061	8.328474	6.833983	4.856958	2.523750
1.585674	3.055232	4.305392	5.254564	5.845960	6.046819	5.846483	5.255502	4.306542	3.056318	1.586378
0.765152	1.475094	2.080213	2.540625	2.828003	2.925706	2.828264	2.541094	2.080788	1.475637	0.765504

We also have solved (6.2) by one of the proposed Modified Hybrid Evolutionary Algorithm, MGSBUA Algorithm. Here threshold error is taken as  $\eta = 10^{-7}$ . The estimation procedure of the error is discussed earlier (§ 4.7.1).

For the following result the number of generations (iteration) required was 109.

49.233647	68.198716	76.620933	80.710572	82.643542	83.218755	82.643542	80.710572	76.620933	68.198716	49.233647
28.735872	46.940286	57.574442	63.577814	66.644842	67.587936	66.644842	63.577814	57.574442	46.940286	28.735872
18.769557	33.252112	43.158736	49.381399	52.770076	53.843305	52.770076	49.381399	43.158736	33.252112	18.769557
13.090243	24.139869	32.426990	38.018972	41.210757	42.245132	41.210757	38.018972	32.426990	24.139869	13.090243
9.451548	17.790130	24.390383	29.056741	31.808849	32.715707	31.808849	29.056741	24.390383	17.790130	9.451548
6.925819	13.178721	18.287671	22.008761	24.252190	25.000000	24.252190	22.008761	18.287671	13.178721	6.925819
5.073005	9.711264	13.572820	16.438440	18.191151	18.779912	18.191151	16.438440	13.572820	9.711264	5.073005
3.654938	7.020510	9.853906	11.981028	13.294062	13.737347	13.294062	11.981028	9.853906	7.020510	3.654938
2.526236	4.861931	6.841264	8.337705	9.266721	9.581353	9.266721	8.337705	6.841264	4.861931	2.526236
1.588076	3.059714	4.311515	5.261808	5.853764	6.054622	5.853764	5.261808	4.311515	3.059714	1.588076
0.766353	1.477335	2.083274	2.544247	2.831905	2.929608	2.831905	2.544247	2.083274	1.477335	0.766353

The Elapsed time in sequential processor is: 0.063000 seconds.

To check the efficiency of the Modified Hybrid Evolutionary Algorithm the same number of iteration is taken. The corresponding results are compared with the analytical one. For the comparison the Root mean square (RMS) error is calculated. The RMS error for the Gauss-Seidel method is found as **0.235760** and that for the proposed Modified Hybrid Evolutionary Algorithm is found as **0.235158**. Thus we may conclude that for the same number of iteration Modified Hybrid Evolutionary Algorithm is better than the Gauss-Seidel method. To get the same accuracy of the solution Hybrid Evolutionary Algorithm requires 109 iteration where as Gauss-Seidel requires 312 iteration.

### 6.3 Solution of Poisson's Equation



Let us consider the Poisson's equation

$$\nabla^2 f = 8xy \quad (6.4)$$

over the square domain  $0 \leq x \leq 3$  and  $0 \leq y \leq 3$  with  $f = 0$  on the boundary and  $h = 0.25$ . We have to find  $f(x, y)$ .

The discretized form of the equation (6.4) becomes

$$f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{i,j} = 8x(i)y(j) \quad (6.5)$$

For  $h = 0.25$  we have  $11 \times 11$  grid points inside the boundaries. The matrix form will be as follows:-



## Concluding Remarks

---

Our target was to study the use of Hybrid Evolutionary Algorithms for solving large set of linear equations. Iterative methods are best suitable for solving large set of linear equations by computer. The convergence of the well known classical iterative methods is very slow and can be accelerated by using successive relaxation technique. The speed of convergence depends on the relaxation factor  $\omega$  ( $0 < \omega < 2$ ). It is very difficult to estimate the optimum relaxation factor. To estimate the optimal value of the relaxation factor the idea of evolution is utilized and thus Hybrid Evolutionary Algorithms evolved. Jacobi Based Uniform Adaptive Hybrid Algorithm, Gauss-Seidal Based Time Variant Adaptive Hybrid Algorithm and Jacobe Based Time Varint Adoptive Hybrid Algorithm has evolved with underlying idea i.e. using the ideas of evolutionary computations the optimal relaxation factor is estimated and using that optimum value of the relaxation factor the convergence of the classical iterative method is accelerated and thus the hybridization is done. After rigorously examining the existing Uniform Adaptive Hybrid Evolutionary algorithms we have found that *“For solving set of linear equations the presence of cross-over is not necessary”*. Hence we have proposed two modified algorithms. The modified algorithms have no cross over operations as a result they consume less memory and become faster. The proposed algorithms are tested to solve the steady state heat distribution on a square plate with considered boundary conditions. The obtained result is compared with the analytical one and found satisfactory. A sample Poisson’s equation is taken for solving. After discretization a set of linear equations is obtained. To solve that set we again have utilized one our proposed Modified evolutionary algorithm and the obtained solution is presented. It should be noted here that we have not tried to modify the Time Variant Adaptive Hybrid Algorithms.

From our study the following conclusions and remarks can be made:

- i) Hybridization of Evolutionary Computation and classical Iterative methods to solve set of linear equations are very effective.
- ii) The presence of cross-over step in the Uniform Adaptive Evolutionary Hybrid Algorithms to solve a set of linear equations is not necessary.
- iii) The two proposed modified hybrid algorithms are faster and memory effective than their original counterparts.
- iv) Partial differential equations, after discretization can be solved suitably by the proposed modified Hybrid Algorithms.
- v) More studies are needed on the Time Variant Adaptive Hybrid Algorithms.

## References

---

1. Antia, H. M. (1991), "*Numerical Methods for Scientist and Engineers*", Tata McGraw-Hill, New Delhi, pp. 01 – 109.
2. Bäck, T. (1992), "*The Interaction of Mutation Rate, Selection, and Self-adaptation within a genetic Algorithm, in Parallel Problem Solving from Nature*", Ino Procs. of the 1st European Conference on Artificial Life (F. J. Varela and P. Bourguine, Eds) MIT press, MA, pp. 263 – 271.
3. Bäck, T. (1997), "*Self-adaptation, in Handbook of Evolutionary Computation*", Oxford University Press.
4. Bäck, T., G. Rudolph and H-P. Schwefel. (1993), "*Evolutionary Programming and evolution Strategies: Similarities and Differences*", In. Procs. of the 2nd Annual conference on Evolutionary Programming. MIT Press, San Diego, CA. pp. 11-22.
5. Bäck, T. and H-P. Schwefel. (1993), "*An overview of Evolutionary Algorithms for Parameter Optimization*", IEEE Trans. on Evolutionary Computation, 1(1), pp. 1-23.
6. Bäck, T., M. Schutz and S. Khuri (1996), "*Evolution Strategies: An alternative evolutionary Computation Method*", In. Procs. of the 2nd Annual Conference on Evolutionary Programming (M.J. Alliot, E. Luttin, E. Ronald, M. Schoenhauer and D. Rogers, Eds.), Springer-Verlag, Berlin, pp. 3-20.
7. Bäck, T., U. Hammel, and H-P. Schwefel (1997), "*Evolutionary Computation: Comments on the History and Current State*", IEEE Trans. on Evolutionary Computation, 1(1), pp. 3-17.
8. Beyer, H.-G. and K. Deb (2001), "*On Self adaptive features in Real-Parameter Evolutionary Algorithm*", Transactions on Evolutionary Computation, 5(3), pp. 250-270.

9. Blickle, T. (1997), "*Theory of Evolutionary Algorithms and Application to System Synthesis*", A Doctoral Dissertation, Diss. ETH No. 11894, Swiss Federal Institute of Technology, Zurich, Switzerland.
10. Burder, R. L. and J. D. Faires (1997), "*Numerical Analysis (6<sup>th</sup> edition)*", Brooks/Cole – Thomson Learning, USA, pp. 250-472.
11. Carre', B.A. (1961), "*The Determination of the Optimum Accelerating Factor for Successive Over-Relaxation*", The Computer Journal, Vol 4, pp. 73-78.
12. Chapra, S. C. and R. P. Canale (1990), "*Numerical Method for Engineers (2nd edition)*", McGraw-Hill, New York.
13. Chellapilla, K. H. Birro and S. S. Rao (1998), "*Effectiveness of Evolutionary programming*", In: 3rd Annual conference on Genetic Programming (GP'98), duly 22-25, Univ. of Wisconsin, Madison.
14. Cheney, W. and D. Kincaid (1999), "*Numerical Mathematics and computing (4<sup>th</sup> edition)*", Brooks/Cole – Thomson Learning, USA pp. 240-316.
15. Engeln-Müllges, G. E. and F. Uhlig (1996), "*Numerical Algorithms with C*", Springer-Verlag, Heidelberg, pp. 59 – 142.
16. Fogel, D. B (1995), "*Evolutionary Computation: Towards a New Philosophy of Machine Inelegance*", IEEE Press, Piscataway, N J.
17. Fogel, L. J., A. J. Owens and M. J. Walsh (1966), "*Artificial Intelligence through Simulated Evolution*", Wiley, New York..
18. Forsythe, G. E. and C. B. Moler (1967), "*Computer Solution of Linear Algebraic Systems*", Prentice-Hall, Englewood Cliffs, New Jersey.
19. Gerald, C. F., and P. O. Wheatley (1994), "*Applied Numerical Analysis (5<sup>th</sup> edition.)*", Addison-Wesley, New York. pp. 102-209.
20. Gourdin, A. and M. Boumahrat (1996), "*Applied Numerical Methods*", Prentice Hall of India, New Delhi, pp. 212-232.
21. Hagaman, L. A. and D. M. Young (1981)' "*Applied Iterative, Methods*", Academic press, New York.
22. Hashem, M. M. A. (1999), "*Global Optimization Through a New Class of Evolutionary Algorithm*", Ph.D. dissertation, Diss. No. 19, Saga University, Japan, pp. 1-30.



23. He, J., J. Xu, and X. Yao (2000), "*Solving Equations by Hybrid Evolutionary Computation Techniques*", Transactions on Evolutionary Computation, 4(3), pp. 295-304.
24. Holland, J. H. (1962), "*Outline for a Logical Theory of Adaptive Systems*", Journal of the Association for Computing Machinery, 3, pp. 297-314.
25. Jain, M. K., S. R. K. Iyengar and R. K. Jain. (1985), "*Numerical Methods for Scientific and Engineering Computation (2nd edition)*", Wiley Eastern, India.
26. Jamali, A R M J U, M. M. A. Hashem and M. B. Rahman (2003), "*An Approach to Solve Linear Equations Using a Jacobi-Based Evolutionary Algorithm*", Proceeding of the ICEECE, December 22–24, Dhaka, Bangladesh, pp. 225-230.
27. Jamali, A. R. M. Jalal Uddin, M. M. A. Hashem and M. B. Rahman (2004a), "*Solving Linear Equations Using a Jacobi Based Time-Variant Adaptive Hybrid Evolutionary Algorithm*", Proceedings of The 7<sup>th</sup> International Conference on Computer and Information Technology (ICCIT) 2004, BRAC University, pp. 688-693.
28. Jamali, A. R. M. Jalal Uddin, M. M. A. Hashem and M. B. Rahman (2004b), "*An Approach to Solve Linear Equations Using Time-Variant Adaptive Based Hybrid Evolutionary Algorithm*", The Jahangirnagar University Journal of Science, Jahangirnagar University, Bangladesh, Vol. 27, pp. 277-289.
29. Jun, H., J. Xu and X. Yao (2000), "*Solving Equations by Hybrid Evolutionary Computation Techniques*", Transactions on Evolutionary Computation, Vol.4, No-3, pp 295-304.
30. Kim, J. H. and H. Myung (1997), "*Evolutionary Programming Techniques for Constrained Optimization Problems*", IEEE Trans. on evolutionary Computation, 1(2), pp. 129 – 140.
31. Koza, J. R. (1994), "*Genetic Programming on the Programming of Computers by Means of Natural Evolution*", MIT Press, Massachusetts.
32. Krishnamurthy, E. V. and S. K. Sen (1989), "*Numerical Algorithms computations in Science and Engineering*", Affiliated East-West Press New Delhi, pp. 157-259.

33. Mathews, J. H. (2001), "*Numerical Methods for Mathematics, Science, and Engineering, (2nd edition. & 6th reprint)*", Prentice-Hall of India, New Delhi.
34. Michalewicz, Z. (1994), *A Hierarchy of Evolution Programs, "An Experimental Study, Evolutionary Computation"*, 1(1), pp. 51 – 76.
35. Michalewicz, Z. (1994a), "*Evolutionary Computation Techniques for Nonlinear Programming Problems*", International Trans. On Operation Research, 19(2), pp. 223- 240. ([http:// www.coe.uncc.edu /~zbyszek/papers.html](http://www.coe.uncc.edu/~zbyszek/papers.html)).
36. Michalewicz, Z. (1996), "*Genetic Algorithms + Data Structure = Evolution Programs, (3rd Rev., and extended edition)*", Springer-Verlag, Berlin..
37. Michalewicz, Z. and N. F. Attia (1994)' "*Evolutionary Optimization of Constrained Problems*", Procs. of the 3rd. Annual Conference on Evolutionary Programming, River Edge, NJ, World Scientific, pp. 98-108.
38. Rechenberg, I. (1993), "*Evolutions Strategies: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution*", Fromman-Holzbook Verlag, Stuttgart.
39. Rechenberg, I. (1994), "*Evolution Strategy, In. Computational Intelligence: Imitating Life*" (J.M. Zurada, R.J. Marks II and C.J. Robinson,Eds.), IEEE Press, New York, NY, pp. 147-159.
40. Salomon, R. (1998), "*Evolutionary Algorithms and Gradient Search: Similarities and Differences*", IEEE Trans. on Evolutionary Computation, 2 (2), pp. 45 – 55.
41. Salomon, R. and J. L. V. Hemmen (1996), "*Accelerating Back Propagation Through Dynamic Self-adaptation*", Neural Networks, 9(4), pp. 589-601.
42. Schoenauer, M. and Z. Michalewicz. (1997), "*Evolutionary Computation*", Control and Cybernetics 26(3), pp. 303-388.
43. Schwefel, H.-P., G. Rudolph and T. Bäck. (1995), "*Contemporary Evolution Strategies in Advances Artificial Life*", Third International Conference on Artificial Life. Vol. 929 of lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, Germany. pp. 893-907.
44. Stoer, J. and R. Bulirsch. (1991/92), "*Introduction to Numerical Analysis (2nd edition)*", Springer, New York.

45. Varga, R. S. (1962), "*Matrix Iterative Analysis*", Prentice-Hall, Englewood Cliffs, New Jersey.
46. Watanabe, K. and M. M. A. Hashem (2004), "*Evolutionary Computation Technique for Nonlinear Programming Problem*", International Trans. on Operation Research, Vol. 1, No. 2, pp 223-240.
47. Yao, X., and Y. Liu (1997), "*Fast Evolutionary Strategies*", Control and Cybernetics, Special Issue on Evolutionary Computation, 26(3), pp.467 – 497.
48. Young, D. (1954), "*Iterative Method for Partial Difference Equations of Elliptic Type*", Trans. American Math. Soc, Vol 7(6), pp. 92-111.
49. Young, D. (1971), "*Iterative Solution of Large Linear System*", Academic Press, Now York.
50. Yuret, D. (1994), "*From Genetic Algorithm to Efficient Optimization*", MIT, A.I. Technical Report No. 1569.