

STUDY OF SOLVING LINEAR EQUATIONS BY HYBRID EVOLUTIONARY COMPUTATION TECHNIQUES

BY

Abdur Rakib Muhammad Jalal Uddin Jamali

Roll No. 0051502



A thesis submitted for the partial fulfillment of the requirements
for the degree of

Master of Philosophy

**Department of Mathematics
Khulna University of Engineering & Technology
Khulna, Bangladesh.**

October 2004

**STUDY OF SOLVING LINEAR EQUATIONS
BY HYBRID EVOLUTIONARY
COMPUTATION TECHNIQUES**

M. Phil Thesis

Abdur Rakib Muhammad Jalal Uddin Jamali

**Department of Mathematics
Khulna University of Engineering & Technology
Khulna, Bangladesh.**

October 2004

Solving a set of simultaneous linear equations is a fundamental problem that occurs in diverse applications. For solving large sets of linear equations, iterative methods are preferred over other methods specially when the coefficient matrix of the linear system is sparse. The rate of convergence of iterative (Jacobi & Gauss-Seidel) methods is increased by using successive relaxation (SR) technique. But SR technique is very sensitive to relaxation factor, ω . Recently, hybridization of evolutionary computation techniques with classical Gauss-Seidel-based SR method has successfully been used to solve large set of linear equations in which relaxation factors are self-adapted. Under this paradigm, this research work has developed a new class of hybrid evolutionary algorithms for solving system of linear equations. The *first* algorithm is the Jacobi-Based Uniform Adaptive (JBUA) hybrid algorithm, which has been developed within the framework of contemporary Gauss-Seidel-Based Uniform Adaptive (GSBUA) hybrid algorithm, and classical Jacobi method. The proposed JBUA hybrid algorithm can be implemented, inherently, in parallel processing environment efficiently whereas GSBUA hybrid algorithm cannot be implemented in parallel processing environment efficiently. The *second* algorithm is the Gauss-Seidel-Based Time-Variant Adaptive (GSBTVA) hybrid algorithm that has been developed within the framework of contemporary GSBUA hybrid algorithm and time-variant adaptive technique. In this algorithm two new time-variant adaptive operators have been introduced based on some observed biological evidences. The *third* algorithm is the Jacobi-Based Time-Variant Adaptive (JBTVA) hybrid algorithm that has been developed within the framework of GSBTVA and JBUA hybrid algorithms. This proposed JBTVA algorithm also can be implemented, inherently, in parallel processing environment efficiently. All the proposed hybrid algorithms have been tested on some test problems and compared with other hybrid evolutionary algorithms and classical iterative methods. Also the validity of the rapid convergence of the proposed algorithms are proved theoretically. The proposed hybrid algorithms outperform the contemporary GSBUA hybrid algorithm as well as classical iterative methods in terms of convergence speed and effectiveness.

Khulna University of Engineering & Technology
Department of Mathematics


We hereby recommend that the thesis prepared by

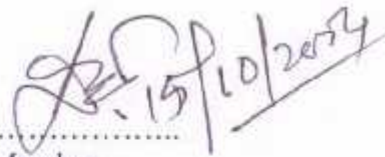
Abdur Rakib Muhammad Jalal Uddin Jamali
(Roll No. 0051502, Registration No. 107, Session 2000-2001)

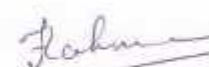
entitled “**Study of Solving Linear Equations by Hybrid Evolutionary Computation Techniques**” be accepted as fulfilling the part of the requirements for the degree of **Master of Philosophy** in the Department of Mathematics

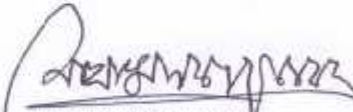
Committee of Examiners

1. Professor Dr. Md. Bazlar Rahman
Supervisor & Head
Department of Mathematics
Khulna University of Engineering & Technology
Bangladesh.
2. Professor Dr. M. M. A. Hashem
Co-supervisor
Department of Computer Science & Engineering
Khulna University of Engineering & Technology
Bangladesh.
3. Professor Dr. Fouzia Rahman
Department of Mathematics
Khulna University of Engineering & Technology
Bangladesh.
4. Professor Dr. M. Kaykobad
Department of Computer Science & Engineering
Bangladesh University of Engineering & Technology
Bangladesh.

 15/10/04
.....
Chairman

 15/10/2004
.....
Member


.....15-10-04
Member


.....
Member (External)

Statement of Originality



This thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any University, and to the best of my knowledge and belief, dose not contain any material previously published or written by another person except where due reference is made in the text.

.....
Abdur Rakib Muhammad Jalal Uddin Jamali

Dedication



To my respectable parents whose constant guidance and inspirations helped me to choose the correct path of life.

&

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

Acknowledgements

I wish to express my profound gratitude to my supervisor Dr. Md. Bazlar Rahman, Professor, Department of Mathematics, Khulna University of Engineering & Technology for his constant guidance and encouragement during my research work. I also wish to express my sincere gratitude to my co-supervisor, Dr. M. M. A. Hashem, Professor, Department of Computer Science and Engineering, Khulna University of Engineering & Technology for his valuable suggestions, criticism and guidance throughout all phases of the research work. Dr. Hashem 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 Mohammad Arif Hossain, Assistant Professor, Department of Mathematics, Munshi Tauhiduzzaman, Assistant Professor, Department of Humanities and Md. Mahfuz Hasan Milon, Ex-Lecturer, Department of Computer Science and Engineering, Khulna University of Engineering & Technology, for their generous assistance and valuable suggestions throughout the entire period of research work. I would like to thank Md. Akkas Uddin Pathan, Assistant Librarian, Md. Nurul Huda Assistant Librarian and Miss Dilu Ara Assistant Programmer, Computer Center, for their 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 wish to convey my thanks to the contemporary great people of evolutionary computation field like, Dr. Xin Yao of University of Birmingham (<http://www.cs.bham.ac.uk/~xin/>), UK, Dr. Jun He, Dr. Jiyou Xu Northern Jiaotong University (jhe1998@263.net), Beijing, China, Dr. David B. Fogel of Natural Selection (<http://www.natural.selection.com/people/dbf.html>) Inc. USA, Prof. H.-P. Schwefel of University of Dortmund, Germany (<http://ls11-www.informatik.uni-dortmund.de/people/schwefel/>), Prof. Z. Michalewicz (<http://www.coe.uncc.edu/~zbyszek/>) of University of North Carolina-Charlotte, USA for providing their recent published papers in their respective Internet Homepages from which I was able to expedite my research work.

I am obliged to express my heartiest thanks to my wife, Mrs. Nasima Parveen, for her constant inspiration and encouragement. Finally, I would like to express my sincere thanks to my son, Mohammad Tahmid Kawsar, for his sacrifice to get affectionate and love what they ought to deserve, during my research period.

Contents

Pages

Abstract.	i
Approval.	ii
Statement of Originality.	iii
Dedication.	iv
Acknowledgement.	v
Contents	vi
List of Figures.	viii
List of Tables.	ix
List of Symbols	x
Chapter	
1 Introduction	1
1.1 Background	1
1.2 Statement of the Problem.	5
1.3 Goals of the Thesis.	5
1.4 Contributions	7
1.5 Structure of the Thesis	8
2 An Overview of Classical Numerical Methods	09
2.1 Introduction	09
2.2 Problem	09
2.3 Consistency	10
2.4 Direct Methods.	11
2.4.1 Gauss Elimination	12
2.4.2 Crout Method.	12
2.5 Classical Iteration Methods.	14
2.5.1 Iterative Techniques	15
2.5.2 Jacobi Method.	16
2.5.3 Gauss-Seidel Method.	17
2.5.4 Successive Relaxation (SR) Technique.	18
2.6 Summary.	19
3 An Overview of Evolutionary Computation	20
3.1 Introduction.	20
3.2 Variants of Evolutionary Algorithms.	21
3.3 Basic Mechanisms of Evolutionary Algorithms.	23
3.3.1 Time-Variant Mutation.	24
3.4 Modern Trends: Hybrid Algorithms.	25
3.5 Properties of Evolutionary Algorithms.	26
3.6 Merits and Demerits of Evolutionary Algorithms.	28
3.6.1 Merits.	28
3.6.2 Demerits.	29
3.7 Summary.	30
4 A Jacobi Based Uniform Adaptive Hybrid Algorithm	31
4.1 Introduction.	31
4.2 Proposed Method	32
4.2.1 The Basic Equations of Jacobi Based SR Method.	32
4.2.2 The Algorithm.	33
4.3 Performance of the Proposed algorithm	35
4.4 Parallel Processing	40
4.5 Summary.	41

5	A Gauss-Seidel Based Time Variant Adaptive Hybrid Algorithm	42
5.1	Introduction.	42
5.2	Development of Time-Variant Adaptive Parameters.	43
5.2.1	Basic Notion	43
5.2.2	Formulas	43
5.2.3	Properties.	44
5.3	Proposed Method	46
5.3.1	The Basic Equations of Gauss-Seidel Based SR Method	46
5.3.2	The Algorithm.	47
5.4	Performance of the Proposed Algorithm.	49
5.5	Summary.	54
6	A Jacobi Based Time Variant Adaptive Hybrid Algorithm	55
6.1	Introduction.	55
6.2	Proposed Method	55
7.2.1	The Algorithm	56
6.3	Performance of the Proposed Algorithm	56
6.4	Parallel Processing	59
6.5	Summary.	59
7	Validity of the Algorithms	60
7.1	Introduction.	60
7.2	Theorems.	60
7.2.1	Convergence Theorem.	60
7.2.2	Adaptation Theorem.	62
7.3	Summary.	66
8	Discussions, Conclusions and Recommendations	67
8.1	Introduction	67
8.2	Discussions	67
8.3	Concluding Remarks	69
8.4	Recommendations for Future Research	71
	Appendices	72
A	Errors	72
A.1	Errors in Numerical Computation.	72
B	Definitions And Theorems	75
B.1	Some Definition of Matrices.	75
B.2	Some Definitions Related to Iterative Methods.	79
B.3	Some Theorems Related to Iterative Methods	82
C	Algorithms	84
C.1	Some Algorithms of Classical Iterative methods.	84
C.1.1	Algorithm of Jacobi Method.	84
C.1.2	Algorithm of Gauss-Seidel Method.	85
C.1.3	Algorithm of Jacobi based SR Method	85
C.1.4	Algorithm of Gauss-Seidel based SR Method.	86
C.2	Some Evolutionary Algorithms	87
C.2.1	A Pseudo-code Structure of Evolutionary Algorithms	87
C.2.2	A Pseudo-code Structure of Hybrid Evolutionary Algorithms	88
	Publications	89
	Reference	90

List of Figures



Figures		Pages
1.1	An abstract view of the evolutionary computation search cycle	3
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	15
4.1	Curve (a) represents proposed JBUA hybrid generation history, curve (b) represents classical Jacobi-SR iteration history and curve (c) represents classical Jacobi iteration history	39
5.1	Rate of change of the variation of T_w for various λ	45
5.2	Rate of change of the variation of T_w for various γ	45
5.3	Rate of change of the variation of p_x for $\lambda = 50$ and $E_x = 0.125$	45
5.4	Rate of change of the variation of p_y for $\lambda = 50$; $E_y = 0.03125$	45
5.5	Rate of change of the variation of p_x for $\gamma = 12$ and $E_x = 0.0315$	46
5.6	Rate of change of the variation of p_y for $\gamma = 12$ and $E_y = 0.03145$	46
5.7	Curve (A) represents the evolution history of proposed GSBTVA (used Lambda based TVA parameter) hybrid algorithm, curve (B) represents the evolution history of proposed GSBTVA (used Gamma based TVA parameter) hybrid algorithm and curve (C) represents the evolution history of GSBUA hybrid algorithm	51
5.8	Self-adaptation of $\tilde{S}_1 = 0.5$ in the UA-based Algorithm.	53
5.9	Self-adaptation of $\tilde{S}_2 = 1.5$ in the UA-based Algorithm.	53
5.10	Self-adaptation of $\tilde{S}_1 = 0.5$ in the TVA-based Algorithm.	53
5.11	Self-adaptation of $\tilde{S}_2 = 1.5$ in the TVA-based Algorithm.	53
5.12	A graphical view of self-adaptation process of relaxation factors $\tilde{S}_1 = 0.5$ and $\tilde{S}_2 = 1.5$ in the GSBTVA- Algorithm.	53
6.1	Curve (A) represents the evolution history of proposed JBTV hybrid algorithm and curve (B) represents the evolution history of JBUA hybrid algorithm.	57
7.1	Decrease both spectral radii of \tilde{S}_x, \tilde{S}_y when $\tilde{S}_x, \tilde{S}_y < \tilde{S}^*$	63
7.2	Decrease both spectral radii of \tilde{S}_x, \tilde{S}_y when $\tilde{S}_x, \tilde{S}_y > \tilde{S}^*$	63
7.3	Decrease spectral radius of \tilde{S}_x and \tilde{S}_y when $\tilde{S}_x < \tilde{S}^* < \tilde{S}_y$	64
7.4	Decrease spectral radius of \tilde{S}_x and \tilde{S}_y when $\tilde{S}_x > \tilde{S}^* > \tilde{S}_y$	64
C.1	A pseudo-code structure of evolutionary algorithms	86
C.2	A pseudo-code structure of hybrid evolutionary algorithms	87

List of Tables



Tables	Pages
3.1 Main characteristics of evolutionary algorithms	22
4.1 Comparison of Jacobi-based SR method and proposed JBUA hybrid algorithm	37
4.2 Comparison of Jacobi-based SR method and proposed JBUA hybrid algorithm	37
4.3 The dynamical change of relaxation factors, α , β , for corresponding individuals at different generations for proposed JBUA hybrid algorithm	38
4.4 Comparison between existing GSBUA and proposed JUUA hybrid algorithms for several randomly generated test problems.	40
5.1 Comparison of existing GSBUA and proposed GSBTVA hybrid algorithms for several randomly generated test problems.	52
6.1 Comparison between JBUA and proposed JBTVA hybrid algorithms for several randomly generated test problems.	58



Introduction

1.1 Background

Systems of linear equations are associated with many problems in engineering and science, as well as with applications of mathematics to the social sciences and the quantitative study of business, statistics and economic problems. Solving a set of simultaneous linear equations is probably the most important topic in numerical methods. Linear equations arise frequently in physical problems, since the simplest models for the physical world are linear. Even the most complicated problems are frequently approximated by a linear model as a first step. Further, the solution of system of nonlinear equations is achieved by an iterative procedure involving the solution of a series of linear equations, each of them approximating the nonlinear equations. Similarly, the solution of ordinary differential equations, partial differential equations and integral equations using finite difference method lead to system of linear or nonlinear equations. Linear equations also arise frequently in numerical analysis. For example, the method of undetermined coefficients, which is useful for deriving formulas for numerical differentiation, integration or solution of differential equations, generally lead to a system of linear equations. After invention of computers, one of the main issues is how one can increase the speed to solve linear equations. For example short-term weather forecasting, image processing, simulation to predict aerodynamics performance which of these application involve the solution of very large sets of simultaneous equations by numerical methods and solution time is an important factor for practical application of results. Because of the great importance of this topic, a large amount of literatures as well as software, using classical methods, are available for the solution of system of linear equations [Engeln-Müllges and Uhlig (1996), Press et al. (1988), Antia (1991), Dongrarra et al. (1997), Gregory (1969), Forsythe and Moler (1967) and Faddeev and Faddeeva, (1963)].

Consider a system of linear algebraic equations

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad (i = 1, 2, \dots, m) \quad (1.1.1)$$

If the number of variable n is equal to the number of equation m and if the coefficient matrix $\mathbf{A} = [a_{ij}]$ is nonsingular, then a unique solution may be expected [Antia (1991)]. For solving linear equations, there are two classes of classical numerical methods – direct methods and iterative methods. The well known direct methods are Gaussian elimination method, Gauss-Jordon method, LU Decomposition methods (such as Crout method) etc [Engeln-Müllges and Uhlig (1996), Cheney and Kincaid (1999), Burder and Faires (1997), Gerald and Wheatley (1994)]. In these direct methods, if the calculations are done exactly, the solution will be exact. However, for large number of linear equations, the inevitable roundoff error (by using computer) may completely ruin the results. Apart roundoff error, with capable of parallel processing, the situation may change in favor of iterative methods. Hence for set of linear equations, specially, for large sparse and structured coefficient (matrices) equations [Pissanetzky (1984) and Tewarson (1973)], iterative methods are preferable. As they are unaffected by roundoff error and truncation error to a large extent [Antia (1991), Wilkinson (1963), Gerald and Wheatley (1994), Varga (1962) and Young (1971)]. The well-known classical numerical iterative methods are Jacobi method and Gauss-Seidel method. The rate of convergence, as very slow for both cases, can be accelerated by using Successive Relaxation (SR) technique [Gerald and Wheatley (1994), Varga (1962), Engeln-Müllges and Uhlig (1996)]. But the speed of convergence depends on relaxation factor, \check{S} with a necessary condition for the convergence is $0 < \check{S} < 2$ [Young (1954), Hagaman and Young (1981), Stoer and Bulirsch (1991), Engeln-Müllges and Uhlig (1996) and Gourdin and Boumahrat (1996)]. However, it is often very difficult to estimate the optimal relaxation factor, which is a key parameter of SR technique [Young and Frank (1963), Varga (1962), Engeln-Müllges and Uhlig (1996), Gourdin and Boumahrat (1996)]. Also SR technique is very sensitive to the relaxation factor [Carre (1961), Krishnamurthy (1989)].

On the other hand the Evolutionary Computation (EC) techniques are stochastic algorithms whose search methods model some natural phenomena: genetic

inheritance and Darwinian strife for survival [Schoenauer and Michalewicz (1997), Bäck and Schwefel (1993) and Bäck et al. (1997)]. Nearly three decades of research and applications have clearly demonstrated that the simulated search process of natural evolution can yield very robust, direct computer algorithms, although these limitations are crude simplification of biological reality. The resulting evolutionary computation techniques are based on the collective learning process within a population of individuals, each of which represents a search point in the space of potential solutions to a given problem. The population is arbitrarily initialized, and it evolves towards better and better region of the search space by means of randomized process of selection (which is deterministic in some algorithms), mutation, and recombination (which is completely omitted in some algorithmic realizations). The environment (given aim of the search) delivers a quality information (fitness value) of the search points, and the selection process favors those individuals of higher fitness to reproduce more often than worse individuals. The recombination mechanism allows the mixing of parental information while passing it to their descendants, and mutation introduces innovation into the population [Bäck and Schwefel (1993), Schoenauer and Michalewicz (1997) and Bäck et al. (1997), Hashem (1999)]. According to this elucidation, the simulated evolutionary search cycle is abstractly depicted in **Fig. 1.1** [Hashem (1999)].

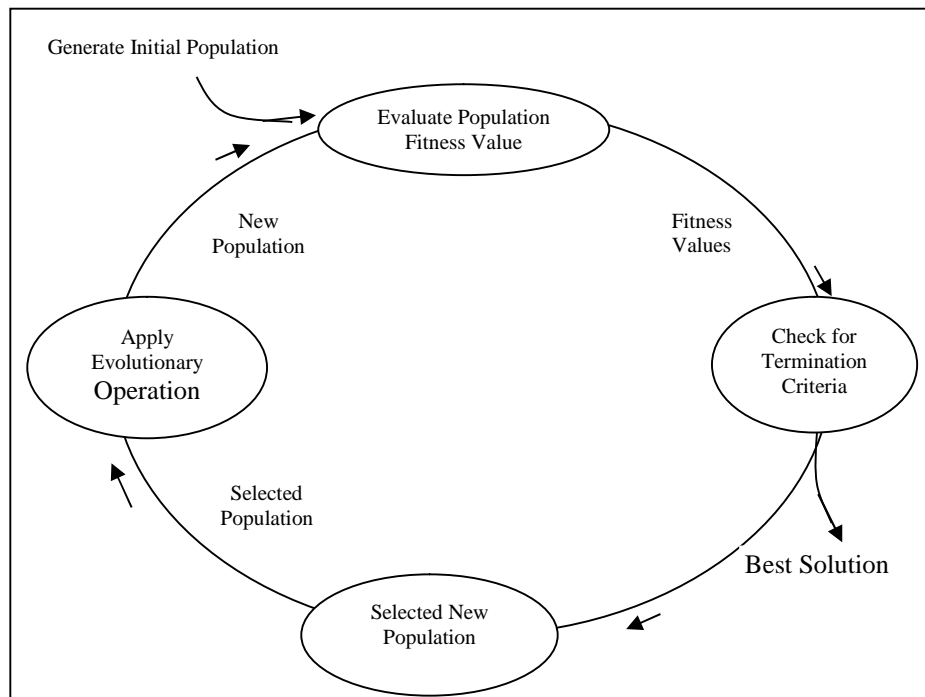


Figure 1.1: An abstract view of the Evolutionary Computation search cycle.

There is a general “agreement” that EC is “made up” of four main branches: (i) Evolutionary Programming (EP), [Fogel et al. (1966), Bäck and Schwefel (1993)], (ii) Evolution Strategies (ES), [Rechenberg (1994), Bäck and Schwefel (1993) and Hashem (1999)], (iii) Genetic Algorithms (GAs [Holland (1962), Bäck and Schwefel (1993)] and (iv) Genetic Programming (GP), [Koza (1994)]. There has been a huge increase in number of papers and successful applications of evolutionary computation techniques in a wide range of areas in recent years. Almost all of these works can be classified as evolutionary optimization (either numerical or combinatorial) or evolutionary learning [He et al. (2000), Salomon (1998)]. But Fogel and Atmar (1990) used linear equation solving as test problems for comparing recombination, inversion operations and Gaussian mutation in an evolutionary algorithm. However, they emphasized their study not on equation solving, but rather on comparing the effectiveness of recombination relative to mutation. No comparison with classical equation-solving methods was given. Recently a very different and novel application of evolutionary computation was used to solve system of linear equations and partial differential equations, which is presented by He et al. (2000). In that paper, a hybrid evolutionary algorithm (Gauss-Seidel Based Uniform Adaptive hybrid algorithm) is developed by integrating classical Gauss-Seidel based SR technique with evolutionary computation techniques to solve equations in which \tilde{S} is self-adapted by using Uniform Adaptation (UA) technique. The idea of self-adaptation was also applied in many different fields [Beyer and Deb (2001), Salomon and Hemmen (1996), Bäck (1997), Bäck (1992), Smith and Fogary (1996)].

Obvious biological evidence is that a rapid change is observed at early stages of life and a slow change is observed at later stages of life in all kinds of animals/plants. These changes more often occur dynamically depending on the situation exposed to them. By mimicking this emergent natural evidence, a special dynamic Time-Variant Mutation (TVM) operator is proposed by Hashem (1999), Michalewicz (1996) and Bäck et al. (1997), Bäck and Schwefel (1993) in global optimization problem. But Time-variant adaptive (TVA) parameter for solving linear equations has not been used yet.

1.2 Statement of The Problem

Though there are many numerical methods for solving linear equations (as Eqn.(1.1.1)) but they have some limitations. Some of those limitations are described, in brief, below:

- (a) **Limitations of direct methods:** For small and dense coefficients matrix of system of linear equations direct methods are efficient and easy to implement. But for large set of linear equations, especially for sparse and structured coefficient (matrix) equations, solutions using direct method become arduous and for the inevitable roundoff error, solutions may become completely useless [Young (1971), Antia (1991), Wilkinson (1963), Gerald and Wheatley (1994), Chapra and Canale (1990)]. Moreover, direct methods cannot be implemented in parallel processing environment efficiently.
- (b) **Limitations of iterative methods:** The well known iterative methods are Gauss-Seidel method and Jacobi method, but speed of convergence of both methods are slow. By using SR technique, the speeds of convergence of both the methods are accelerated. But SR technique needs to pre-estimate the optimal relaxation factor. Also SR technique is very much sensitive to the relaxation factor [Carre (1961), Krishnamurthy.and Sen (1989), Young and Frank (1963) and Young (1971)].
- (c) **Limitations of Gauss-Seidel Based Uniform Adaptive hybrid algorithm:** Though in Gauss-Seidel Based Uniform Adaptive (GSBUA) [He et. al. (2000)] hybrid algorithm, we need not pre-estimate the optimal relaxation factor but this algorithm cannot be implemented, inherently, in parallel processing environment efficiently. Also in this algorithm, as uniform adaptation technique is used for self-adaptation of relaxation factor, it has a tendency of oscillation. Consequently the rate of convergence becomes relatively slow [Jamali et. al. (Dec. 2003), Jamali et. al. (2003)].

1.3 Goals of The Thesis

After the invention of computer, one of the main issues is how to decrease the time to solve problems. If appropriate algorithms can be used in parallel processing

environment, then it can decrease a significant amount of time. Though each evolutionary algorithm can be implemented in parallel processing environment, but Gauss-Seidel based algorithm, inherently, cannot be implemented in parallel processing environment. On the other hand, Jacobi based algorithm, inherently, can be implemented in parallel processing environment. For example, if n^2 processors are available, then Jacobi algorithm reduces the time, for each iteration, to $\log_2 n$ time units. This is a significant speedup over the sequential algorithm, as Gauss-Seidel algorithm, which requires n^2 time units per iteration [Gerald and Wheatley (1994)]. Apart from this, Jacobi algorithm is more robust and stable [Engeln-Müllges and Uhlig (1996)]. *So, if a hybrid evolutionary algorithm will be developed by integrating Jacobi based SR technique with evolutionary computation techniques within framework of GSBUA hybrid evolutionary algorithm, it may be overcome the limitation of recently developed GSBUA hybrid algorithm.*

Recently developed GSBUA hybrid algorithm [He et. al. (2000)], relaxation factors are adapted uniformly by using uniform random number as adaptive sample space. For the cause of uniform adaptation, relaxation factors are not finely adapted and there is a tendency of relaxation factor to oscillate in any stages and there is no local fine-tuning in later stages. However, obvious biological evidence is that a rapid change is observed at early stages of life and a slow change is observed at later stages of life in all kinds of animals/plants. These changes more often occur dynamically depending on the situation exposed to them [Hashem (1999)]. *By mimicking this emergent natural evidence, a time variant adaptive (TVA) parameter has been developed. Also if a time-variant adaptive hybrid evolutionary algorithm based on classical Gauss-Seidel algorithm, within framework of GSBUA hybrid evolutionary algorithm will be developed then the limitation of GSBUA algorithm of tuning of relaxation factors in later stages may be overcome. Consequently the rate of convergence may be increased.*

Also by integrating Jacobi based SR technique with evolutionary computation techniques, *a time-variant adaptive hybrid evolutionary algorithm within the framework of JBUA hybrid evolutionary algorithm, will be developed, then the limitation regarding tuning of relaxation factor in later stages may be overcome. The speed of convergence may also be increased.*

1.4 Contributions

This thesis work makes the following contributions:

1. This thesis considerably extends the power of hybrid evolutionary computation by introducing Jacobi Based Uniform Adaptive (JBUA), Gauss-Seidel Based Time-Variant Adaptive (GSBTVA) and Jacobi Based Time-Variant Adaptive (JBTVA) hybrid evolutionary algorithms in the numerical area.
2. This thesis also extends the method of solving system of linear equations.
3. The proposed JBUA algorithm outperforms classical Jacobi based SR method and Gauss-Seidel based SR method. Also JBUA algorithm is parallel to GSBUA algorithm in sequential processing environment. And this algorithm outperforms the GSBUA algorithm in parallel processing environment.
4. The proposed GSBTVA algorithm overcomes the disadvantages of uniform adaptation technique of GSBUA algorithm. This hybrid algorithm uses two new Time-Variant Adaptive (TVA) parameters. The performance of GSBTVA hybrid algorithm is much better than that of GSBUA hybrid algorithm.
5. By introducing TVA parameter in JBUA algorithm instead of Uniform Adaptive (UA) parameter, a JBTVA hybrid algorithm is developed. The performance of JBTVA hybrid algorithm is much better than that of JBUA hybrid algorithm.

1.5 Structure of The Thesis

After the introduction which is in this **Chapter**, the remaining thesis is organized as follows:

Chapter 2 discusses the overview of classical methods for solving systems of linear equations and formulations of the systems of linear equations in matrix form. Also this chapter overviews the conditions of existence of solutions of a system. Some well-known classical methods for solving linear equations are also discussed in this chapter.

Chapter 3 discusses an overview of the basic constituents and commitments, comparisons, properties, and merits and demerits of major evolutionary algorithms in

terms of their canonical forms. This chapter also overviews the hybridization of classical numerical method with evolutionary computation techniques for solving linear equations.

Chapter 4 discusses and presents the development of a Jacobi Based Uniform Adaptive (JBUA) hybrid evolutionary algorithm within the framework of contemporary Gauss-Seidel Based Uniform Adaptive (GSUA) hybrid evolutionary algorithm. This proposed new algorithm has been tested and then the performance of the algorithm is compared with classical iterative methods as well as GSBUA hybrid evolutionary algorithm by some numerical experiments.

Chapter 5 discusses the weakness of uniform adaptation techniques, discusses the formulation of a Time-Variant adaptive (TVA) parameter and presents the development of Gauss-Seidel Based Time-Variant Adaptive (GSBTVA) hybrid evolutionary algorithm within the framework of GSBUA hybrid evolutionary algorithm. This newly proposed algorithm has been tested and then the performance of the algorithm is compared with classical iterative methods as well as GSBUA hybrid evolutionary algorithm by some numerical experiments.

Chapter 6 discusses and presents the development of Jacobi Based Time-Variant adaptive (JBTVA) hybrid evolutionary algorithm within the framework of JBUA hybrid evolutionary algorithm. This newly proposed algorithm has been tested and then the performance of the algorithm is compared with JBUA hybrid evolutionary algorithm by some numerical experiments.

Chapter 7 discusses the convergence theorems and the adaptation theorem of hybrid evolutionary algorithms and then proves the theorems.

Chapter 8 contains the detail discussions, concluding remarks and recommendations for possible future extensions of the present works.

Appendices overview the numerical errors coming from computer implements of the algorithms. Also some related definitions and theorems are given in appendix. Moreover some algorithms of classical methods and pseudo-code of evolutionary computation techniques are given in the appendix.



An Overview of Classical Numerical Methods

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 computer solution of system of linear equations. 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.

2.2 Problem

Consider the problem (see Eqn. 1.1.1) of finding values for n unknowns quantities x_j , $j = 1, 2, \dots, n$, so that m given linear equations

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
 \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots & \\
 a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m
 \end{aligned}
 \tag{2.2.1}$$

are simultaneously satisfied. In Eqn. (2.2.1), the coefficients $a_{ij} \in \mathfrak{R}$ and the right hand sides constant component $b_i \in \mathfrak{R}$ are given numbers for $i = 1, 2, 3, \dots, m$ and $j = 1, 2, 3, \dots, n$. Written in matrix notation [Antia (1991)], Eqn. (2.2.1) becomes

$$\mathbf{Ax} = \mathbf{b} \quad (2.2.2)$$

for $\mathbf{x} \in \mathfrak{R}^n$ and $\mathbf{b} \in \mathfrak{R}^m$,
where

$$\mathbf{A} = (a_{ik}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Note that the first index i of a_{ik} indicates the row in which the element a_{ik} occurs in \mathbf{A} , while the second index k of a_{ik} denotes the column in which a_{ik} occurs in \mathbf{A} . The matrix \mathbf{A} is called coefficient matrix of order $m \times n$, where m is indicated number of rows and n is indicated number of column of the matrix \mathbf{A} and vector \mathbf{b} is called right hand constant vector of order m .

A vector \mathbf{x} with components $x_j, j = 1, \dots, n$, that solves Eqn. (2.2.2) is called a solution vector of the linear systems.

If the number of unknowns are equal to the number of equations i.e. $m = n$ then Eqn. (2.2.1) i.e. Eqn. (2.2.2) becomes

$$\mathbf{Ax} = \mathbf{b} \quad (2.2.3)$$

where, $\mathbf{x} \in \mathfrak{R}^n$ and $\mathbf{b} \in \mathfrak{R}^n$, and $\mathbf{A} \in \mathfrak{R}^n \times \mathfrak{R}^n$

2.3 Consistency

When the system (2.2.1) has a solution, it is said to be consistent; otherwise, the system is said to be inconsistent.

Now for the system of Eqn. (2.2.1) the augmented matrix is given by

$$[\mathbf{A} \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{bmatrix} \quad (2.3.1)$$

So in matrix notation, if rank of the coefficient matrix and that of augmented matrix are identical then the system is said to be consistent; otherwise, the system is said to be inconsistent [Ayres (1997), Krishnamurthy and Sen (1989)]. Thus

- (a) A system $\mathbf{Ax} = \mathbf{b}$ of m linear equations in n unknowns is consistent if and only if the coefficient matrix and the augmented matrix of the system have the same rank [Krishnamurthy and Sen (1989)].
- (b) In a consistent system of Eqn. (2.2.2) if rank, r , is less than number of unknowns, n , i.e. $r < n$, the system has many solutions and $n-r$ of unknown may be chosen arbitrarily [Antia (1991)].
- (c) If the right hand side constant vector, \mathbf{b} , is a zero vector then the system of Eqn. (2.2.2) is called homogeneous and then system is always consistent.
- (d) If the right hand side constant vector, \mathbf{b} , is not a zero vector i.e. $\mathbf{b} \neq \mathbf{0}$, then the system of Eqn. (2.2.2) is called non-homogeneous.
- (e) A system of n non-homogeneous equations in n unknowns (i.e. Eqn. (2.2.3)) has **a unique solution** provided the rank of its coefficient matrix \mathbf{A} is n , that is, determinant of \mathbf{A} not equal to zero i.e. $|\mathbf{A}| \neq 0$ [Ayres (1997), Kreyszig (1993), Lang (1987), and Gantmacher (1990)]. Some theorem related to the classical methods are discussed in appendices.

2.4 Direct Methods

The term direct method indicates a method that solves a set of equations by techniques in which it need not 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 [Balagurusamy (2004)]. There are many direct methods to solve system of linear equations such as Gauss elimination method, Gauss-Jordan method, Crout method, Doolittle's method etc [Gerald and Wheatley

(1994), Jain (1985). Two well-known classical direct methods named Gauss Elimination and Crout LU decomposition methods are described below:

2.4.1 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 process a systematic strategy for deducing the system of equations to the upper triangular form using the forward elimination approach and then for a obtaining values of unknowns using the back substitution process. 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. (2.2.3) is reduced to triangular matrix. The relation for obtaining the coefficient of the k th derived system has the general form:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \quad (2.4.1)$$

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_{kk} is referred to as normalization [Balagurusamy (2004)].

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. (2.2.3), by the following way the relation for obtaining the k th unknown, x_k , has the general form:

$$x_k = \frac{1}{a_{kk}^{(k-1)}} \left[b_k^{(k-1)} - \sum_{j=k+1}^n a_{kj}^{(k-1)} x_j \right] \quad (2.4.2)$$

where $k = n - 1$ to 1 , and

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}} \quad (2.4.3)$$

2.4.2 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 \mathbf{L} or \mathbf{U} , and the ones on the diagonal of \mathbf{U} can also be omitted. The LU decomposition is produced by Crout reduction method [Gerald and Wheatley (1994)] as follow:

$$l_{ik} = a_{ik} - \sum_{j=1}^{k-1} l_{ij} u_{jk}, \quad k \leq i, \quad i = 1, 2, \dots, n, \quad (2.4.4)$$

$$u_{ik} = \frac{1}{l_{ii}} \left(a_{ik} - \sum_{j=1}^{i-1} l_{ij} u_{jk} \right), \quad i \leq k, \quad k = 1, 2, \dots, n. \quad (2.4.5)$$

(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 coefficient matrix $\mathbf{A} = [a_{ik}]$ from Eqn. (2.2.3), Lower triangular matrix $\mathbf{L} = [l_{ik}]$ and Upper triangular matrix $\mathbf{U} = [u_{ik}]$.

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

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & \cdots \\ a_{31} & a_{32} & a_{33} & \vdots & \vdots \\ \vdots & \vdots & \vdots & a_{n-1,n-1} & a_{n-,n} \\ a_{n1} & \cdots & \cdots & a_{n,n-1} & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} l_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & l_{22} & u_{23} & \vdots & u_{2n} \\ l_{31} & l_{32} & l_{33} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{(n-1)n} \\ l_{n1} & l_{n2} & \cdots & \cdots & l_{nn} \end{bmatrix} \quad (2.4.6)$$

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. (2.2.3) is readily obtained with the \mathbf{L} and \mathbf{U} matrices by the following formulas:

The general equation for the reduction of \mathbf{b}

$$\left. \begin{aligned} b'_1 &= \frac{b_1}{l_{11}} \\ b'_i &= \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} b'_{jk} \right), \quad i = 2, 3, \dots, n \end{aligned} \right\} \quad (2.4.7)$$

And the equations for the back-substitution are

$$\left. \begin{aligned} x_n &= b'_n, \\ x_k &= b'_k - \sum_{j=k+1}^n u_{kj} x_j, \quad k = n-1, n-2, \dots, 1 \end{aligned} \right\} \quad (2.4.8)$$

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.

2.5 Classical Iterative Methods

As opposed to the direct methods of solving a set of linear equations, 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 roundoff 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 roundoff 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)]. **Fig.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 – when the coefficient matrices are sparse (has many zeros). Then 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.

2.5.1 Iterative Techniques

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

$$\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{V} \quad (2.5.1)$$

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

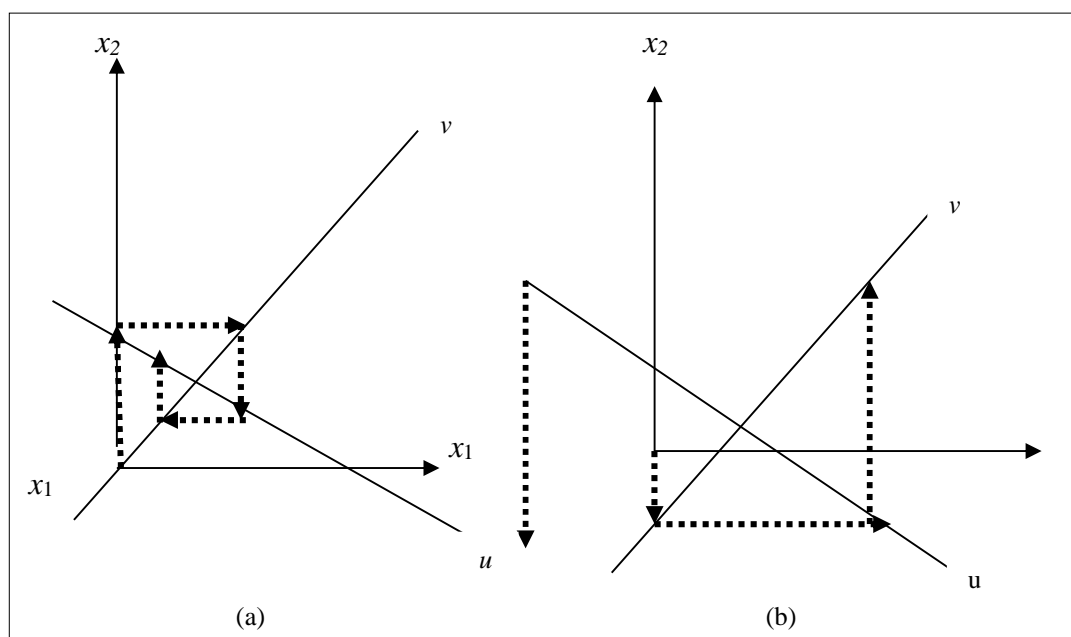


Figure 2.1: Illustration of (a) convergence and (b) divergence of iterative methods. Notices that the same functions (line u and v in the figure) are plotted in both cases.

$$\mathbf{x}^{(k)} = \mathbf{H}\mathbf{x}^{(k-1)} + \mathbf{V}, \quad \text{for each } k = 1, 2, \dots \quad (2.5.2)$$

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

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)} \quad (2.5.3)$$

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

Subtracting Eqn.(2.5.1) from Eqn. (2.5.2) and if the error is defined as

$$v^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}, \quad (2.5.4)$$

then

$$v^{(k+1)} = \mathbf{H}v^{(k)}, \quad k = 0, 1, 2, \dots \quad (2.5.5)$$

from which follows

$$v^{(k+1)} = \mathbf{H}^{(k)}v^{(0)}, \quad k = 0, 1, 2, \dots \quad (2.5.6)$$

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 bellow. SR technique also discussed bellow.

2.5.2 Jacobi Method

Assume that a linear system given in the form Eqn. (2.2.3) is

$$\mathbf{A}\mathbf{x} = \mathbf{b} \text{ with } / \mathbf{A} \neq 0$$

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

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

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

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

$$\text{or } \mathbf{x} = \mathbf{H}_j \mathbf{x} + \mathbf{V}_j \quad (2.5.7)$$

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.(2.2.3) for x_i , then an equivalent form for the system is [Antia (1991), Balagurusamy (2004)]

$$x_i = -\sum_{\substack{k=1 \\ k \neq i}}^n \frac{a_{ik}}{a_{ii}} x_k + \frac{b_i}{a_{ii}}, \quad i = 1, \dots, n \quad (2.5.8)$$

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

$$\begin{cases} \mathbf{x}^{(k+1)} = \mathbf{H}_j \mathbf{x}^{(k)} + \mathbf{V}_j & \text{with} \\ \mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})^t & \end{cases} \quad \text{for } k = 0, 1, 2, \dots \quad (2.5.9)$$

Expressed in component-wise, this Jacobi iteration becomes

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j \neq i}}^n \frac{a_{ij}}{a_{ii}} x_j^{(k)}, \quad i = 1, \dots, n, \text{ and } k = 0, 1, 2, \dots \quad (2.5.10)$$

The iteration matrix \mathbf{H}_j can be viewed as a correction on the last computed iteration as Eqn. (2.5.3) i.e.

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

where

$$\mathbf{z}^{(k)} = \mathbf{V}_j - (\mathbf{I} - \mathbf{H}_j) \mathbf{x}^{(k)} \quad (2.5.11)$$

Jacobi method is also known as the method of simultaneous displacement method [Antia (1991), Balagurusamy (2004)]. The algorithm of this method is given in Appendix C.

2.5.3 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. (2.2.3) is

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

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

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

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

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

$$\text{or } \mathbf{x} = \mathbf{H}_g \mathbf{x} + \mathbf{V}_g \quad (2.5.12)$$

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

$$\begin{cases} \mathbf{x}^{(k+1)} = \mathbf{H}_g \mathbf{x}^{(k)} + \mathbf{V}_g & \text{with} \\ \mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})^t & \text{for } k = 0, 1, 2, \dots \end{cases} \quad (2.5.13)$$

Expressed in component wise, this Gauss-Seidel iteration becomes

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} - \sum_{j=i+1}^n \frac{a_{ij}}{a_{ii}} x_j^{(k)}, \quad i = 1, \dots, n, \quad \text{and } k = 0, 1, \dots \quad (2.5.14)$$

The iteration matrix \mathbf{H}_g can be viewed as a correction on the last computed iteration as Eqn. (2.5.3) i.e

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

where

$$\mathbf{z}^{(k)} = \mathbf{V}_g - (\mathbf{I} - \mathbf{H}_g) \mathbf{x}^{(k)} \quad (2.5.15)$$

Gauss-Seidel method is also known as the method of successive displacement method. The algorithm of this method is given in Appendix C.

2.5.4 Successive Relaxation (SR) Technique

Relaxation 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

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

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

Then using SR technique the final solution is now written as

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \check{\mathbf{S}} \mathbf{z}^{(k)} \quad (2.5.16)$$

where $\mathbf{z}^{(k)}$ is the correction vector and $\check{\mathbf{S}}$ is a relaxation factor.

$$\text{or } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \check{\mathbf{S}} (\tilde{\mathbf{x}}^{(k+\Gamma)} - \mathbf{x}^{(k)})$$

$$\text{or } \mathbf{x}^{(k+1)} = (1 - \check{\mathbf{S}}) \mathbf{x}^{(k)} + \check{\mathbf{S}} \tilde{\mathbf{x}}^{(k+\Gamma)} \quad (2.5.17)$$

Here $\mathbf{x}^{(k+1)}$ is weighted mean of $\tilde{\mathbf{x}}^{(k+\Gamma)}$ and $\mathbf{x}^{(k)}$ and $\check{\mathbf{S}}$ 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 $\check{\mathbf{S}} = 1$, the Eqn. (2.5.16) reduced to the Jacobi/Gauss-Seidel method [Krishnamurthy and Sen (1989), Gerald and Wheatley (1994)].

- (ii) If ω 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 ω 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 ω 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 ω from 0 to 2, is called successive relaxation or SR technique [Gourdin and Boumahrat (1996), Engeln-Müllges and Uhlig (1996)].

The algorithms of SR technique are given in Appendix C.

2.6 Summary

Throughout this chapter an attempt is made to overview the basic properties of classical numerical methods for solving linear equations and also describes about the formulation of the problems in matrix notations and then consistency of the problems. Also overviews some well-known direct methods (Gauss elimination method, Crout decomposition method) as well as classical iterative methods (Jacobi method, Gauss-Seidel method) for solving equations. Successive Relaxation (SR) technique is also discussed here.

An Overview of Evolutionary Computations

3.1 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 to “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 to few (or single) region due to selection and the single 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)].

3.2 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) Evolution Strategies (ESs) [Rechenberg (1993), Bäck and Schwefel (1993) and Hashem (1999)], (iii) Evolutionary Programming (EP) [Fogel et al. (1966), Bäck and Schwefel (1993)] and (iv) Genetic Programming (GA) [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 even noisy or moving response surfaces of optimization problems. The variety of data-structures, variation of operators and selection mechanisms give possible ways of classifying EAs. 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 **Table3.1** [Hashem 1999].

Table 3.1: 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

3.3 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 Evolutionary Algorithms. The EAs can be classified as probabilistic search algorithms, which maintain a population of u individuals, $\Pi(t) = \{s_1(t), s_2(t), \dots, s_u(t)\}$ where $s_i(t) \in \mathbf{S}$ 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 with component $s_i \in \mathcal{R} \forall i \in \{1, 2, \dots, n\}$. Each solution $s_i(t)$ is evaluated to produce some measure of its “fitness” $f(s_i(t))$. After initialization of the population, a new population is formed by three main operators – crossover (recombination), mutation and selection operations. There is higher order transformation: \times (crossover operator), which creates new individuals (offspring) ($\times : (s \times s)^u \rightarrow s^v$) where v is the offspring population size and an unary transformation: ζ (mutation operator), which modifies these new individuals (offspring) by a small change ($\zeta : s \rightarrow s'$). A selection operation ($g : (s^v|_{\text{offspring}} \cup s^u|_{\text{parent}}) \rightarrow s^u$) 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)]. The pseudo-code of this process is given in **Fig. C.1** (see Appendix C).

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. Algorithm **Fig. C.1** (see Appendix C) gives an outline of the mechanism of an EA. The several instances of the EA differ in the way that 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. Back 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 f [Michalewicz (1994, 1994a), Michalewicz and Attia (1994), Kim and Myung (1997), Chellapilla et al. (1998), and Waagen et al. (1992)].

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).

3.3.1 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 [Bäck et. al. (1993), Schwefel et. al. (1995)] do – as $\forall i \in \{1, 2, \dots, n\}$

$$\langle_i^{(m)} = \langle_i + \dagger(t) \cdot N_i(0,1) \quad (3.3.1)$$

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

$$\dagger(t) = \left[1 - q^{\left(1 - \frac{1}{T}\right)^x} \right] \quad (3.3.2)$$

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

The function $\dagger(t)$ returns a value in the range $[0,1]$, that falls within so-call evolution window [Rechebberg (1994)] such that the probability of $\dagger(t)$ being closed to 0 as the generation t increases. This property of $\dagger(t)$ 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)].

3.4 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] 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, ω , 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 ω 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 ω . The fitness of an individual is evaluated by approximate solution. The relaxation factor is adapted after each generation, depending on how well an individual performs.

3.5 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 in dependence 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 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].

3.6 Merits and Demerits of Evolutionary Algorithms

3.6.1 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 an other, 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 “neighbors” A few individuals *migrate* between neighbors and form a loose coupling between the sub-populations [Hashem 1999].

3.6.2 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].

3.7 Summary

Throughout this chapter an attempt is made to overview the basic constituents, properties, merits and demerits of evolutionary algorithms in terms of their canonical forms. But in practical the borders between these approaches are much more fluid. This chapter also overviews the hybridization of classical numerical method with evolutionary computation techniques for solving linear equations.

Jacobi Based Uniform Adaptive Hybrid Algorithm

4.1 Introduction

Invent of easily accessible computers makes it possible and practical for us to solving large set of simultaneous linear algebraic equations. Now for appropriate decision of the physical problems, it is sometimes desired an appropriate algorithm which converged rapidly for solving physical problems. For example, short-term weather forecast, image processing, simulation to predict aerodynamics performance which of these applications involve the solution of very large set of simultaneous equations by numerical methods and time is an important factor for practical application of the results. If the algorithm of solving equations can be implemented in parallel processing environment, it can easily decrease a significance time to get the result. But as Gauss-Seidel based SR method cannot be implemented in parallel processing environment, so Gauss-Seidel Based Uniform Adaptive (GSBUA) hybrid algorithm cannot be implemented, inherently, in parallel processing environment efficiently.

To eliminate above-mentioned problem and to decrease the time of convergence (by using parallel processors) this chapter is devoted to develop a new hybrid evolutionary algorithm [Jamali et. al. (2003)]. This hybrid algorithm uses Jacobi based SR method instead of Gauss-Seidel based SR method. As Jacobi based SR method can be implemented in parallel processing environment [Gerald and Wheatley (1994)], so Jacobi based hybrid evolutionary algorithm can be implemented in parallel processing environment. Note that in hybrid evolutionary algorithm, individuals of population can be implemented in parallel processing environment explicitly.

4.2 Proposed Method

In this section, a new hybrid evolutionary algorithm is proposed in which evolutionary computation techniques and Jacobi based SR technique is used. The proposed Jacobi-Based Uniform Adaptive (JBUA) hybrid evolutionary algorithm does not require a user to guess or estimate the optimal relaxation factor ω . The proposed 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 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.

4.2.1 The Basic Equations of Jacobi Based SR Method

The system of linear equations (Eqn. (1.1.1)) can be written as

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad (i = 1, 2, \dots, n) \quad (4.2.1)$$

In Jacobi method by using SR technique [Engeln-Müllges, and Uhlig (1996)] Eqn. (4.2.1) is given by

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^n a_{ij}x_j^{(k)} \right), \quad (i = 1, 2, \dots, n) \quad (4.2.2)$$

In matrix form Eqn. (4.2.2) can be rewritten in matrix-vector equation as (see Chapter 2 Eqn. (2.4.18)):

$$\mathbf{x}^{(k+1)} = \mathbf{H} \mathbf{x}^{(k)} + \mathbf{V} \quad (4.2.3)$$

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

$$\mathbf{H}_S = \mathbf{D}^{-1} \{ (1 - \tilde{S}) \mathbf{I} - (\mathbf{L} + \mathbf{U}) \}, \quad (4.2.4)$$

and

$$\mathbf{V} = \mathbf{D}^{-1} \mathbf{b}. \quad (4.2.5)$$

4.2.2 The Algorithm

Similar to many other evolutionary algorithms, the proposed 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. (4.2.3). The mutation is stochastic since S used in the iteration is initially generated between \tilde{S}_L and \tilde{S}_U and S 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 follows:

Step-1: Initialization

Generate an initial population of approximate solution to the system of linear Eqn. (4.2.1) using different arbitrary relaxation factors. Denote the initial population as

$$\mathbf{X}^{(0)} = \{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)} \dots \mathbf{x}_N^{(0)}\} \quad (4.2.6)$$

Where each individual $\mathbf{x}_i \in \mathfrak{R}^n$; N is the population size. Let $k \leftarrow 0$ where k is the generation counter. Also initialize relaxation factor s_1, s_2, \dots, s_N randomly from (s_L, s_U) where s_L and s_U are lower and upper boundary of s '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:

$$\mathbf{X}^{(k+c)} = \mathbf{R}(\mathbf{X}^{(k)})' \quad (4.2.7)$$

Where

$$\mathbf{R} = (r_{ij})_{N \times N} \quad (4.2.8)$$

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. (4.2.3) as

$$\mathbf{x}_i^{(k+m)} = \mathbf{H}_i \mathbf{x}_i^{(k+c)} + \mathbf{V}_i, \quad i = 1, 2, \dots, N. \quad (4.2.9)$$

where \mathbf{H}_i is called Jacobi iteration matrix corresponding \mathbf{x}_i and given by

$$\mathbf{H}_i = \mathbf{D}^{-1} \{ (1 - \alpha_i) \mathbf{I} - \alpha_i (\mathbf{L} + \mathbf{U}) \}, \quad (4.2.10)$$

and

$$\mathbf{V}_i = \alpha_i \mathbf{D}^{-1} \mathbf{b}. \quad (4.2.11)$$

Here α_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 relaxation factors α_x and α_y and $\|e(\mathbf{x})\|$ and $\|e(\mathbf{y})\|$ are their corresponding errors (fitness value). Then the relaxation factors α_x and α_y are adapted as follows:

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

(i) then move α_x toward $\tilde{\alpha}_y$ (i.e. α_x is adapted to α_x^m) by using

$$\alpha_x^m = (0.5 + p_x)(\alpha_x + \alpha_y) \quad (4.2.12)$$

$$\text{where } p_x \in (-0.01, 0.01) \quad (4.2.13)$$

and p_x may be called a uniform adaptive probability parameter for x [He et. al. (2000)].

And (ii) move y away from x (i.e. y is adapted to y^m) by using

$$y^m = \begin{cases} y + p_y (U - y), & \text{when } y > x \\ y + p_y (L - y), & \text{when } y < x \end{cases} \quad (4.2.14)$$

$$\text{where } p_y \in (0.008, 0.012) \quad (4.2.15)$$

and p_y may be called a uniform adaptive probability parameter for y [He et. al. (2000)].

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

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

$$\check{S}_x^m = \check{S}_x \text{ and } \check{S}_y^m = \check{S}_y.$$

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 then the algorithm terminates; otherwise, go to **Step -2**.

The pseudo-code structure of this hybrid algorithm is shown in **Fig.C.2** (Appendix C), where iteration matrix, \mathbf{H}_{S_i} , \mathbf{V}_{S_i} and random functions p_x and p_y are defined by Eqn.'s (4.2.10), (4.2.11), (4.2.13) and (4.2.15) respectively.

4.3 Performance of the Proposed Algorithm

In order to evaluate the effectiveness and efficiency of the proposed JBUA hybrid algorithm, numerical experiments had been carried out on a number of problems to solve the systems of linear Eqn. (2.2.3) of the form:

$$\mathbf{Ax} = \mathbf{b} \quad (4.3.1)$$

The proposed JBUA hybrid algorithm, used in all of our experiments, was very simple and had population size two ($N = 2$). That is, only two individuals were used. The recombination matrix (see Eqn. (4.2.8)), in all through the experiments, was chosen as follows:

Since only two individuals were used in a population in our experiment, if the fitness of the first individuals was better then the second (using Eqn. (4.2.7), let

$$\begin{pmatrix} \mathbf{x}_1^{(k+c)} \\ \mathbf{x}_2^{(k+c)} \end{pmatrix} = \begin{pmatrix} 1.0 & 0 \\ 0.99 & 0.01 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^{(k)} \\ \mathbf{x}_2^{(k)} \end{pmatrix} \quad (4.3.2)$$

else let

$$\begin{pmatrix} \mathbf{x}_1^{(k+c)} \\ \mathbf{x}_2^{(k+c)} \end{pmatrix} = \begin{pmatrix} 0.01 & 0.99 \\ 1.0 & 0.0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^{(k)} \\ \mathbf{x}_2^{(k)} \end{pmatrix} \quad (4.3.3)$$

The following settings were also valid all through the experiments:

The dimension of unknown variables was $n = 100$, each individual \mathbf{x} of population \mathbf{X} was initialized from the domain $\mathfrak{R}^{100} \in (-30, 30)$ randomly and uniformly. For each experiment, a total of ten independent runs were conducted and the average results are reported here.

Now first problem was to solve linear equations, Eqn. (4.3.1), where $a_{ii} = 2n$ and $b_i = i$ for $i = 1, \dots, n$ and $a_{ij} = j$ for $i \neq j$, $i, j = 1, \dots, n$ (see problem P₁). The problem was to be solved with an error smaller than $= 10^{-12}$.

Table 4.1 and **Table 4.2** show the numerical results achieved by the classical Jacobi-based SR method and proposed JBUA hybrid evolutionary algorithm with initial relaxation factors, $S = 0.5, 1.50$ and $S = -1.0, 1.0$ respectively. Four experiments were carried out using classical Jacobi based SR method with relaxation factors $S = 0.5, 1.50$ and $S = -1.0, 1.0$. And Two experiments were carried out using the proposed algorithm, one with initial relaxation factors $\alpha_1 = 0.5$ and $\alpha_2 = 1.5$ and the other with initial relaxation factors $\alpha_1 = -1.0$ and $\alpha_2 = 1.0$. It is very clear from the **Tables 4.1** and **4.2** that the proposed algorithm performed much better than the classical Jacobi based SR method. Proposed JBUA algorithm with different initial relaxation factors, α_1 and α_2 , have all found approximate solutions with an error smaller than $y = 10^{-12}$ within 1000 generations, while none of the classical Jacobi based SR method could find an approximate solution with an error smaller than $= 10^{-12}$ after 1000 generations, no matter which relaxation factors had been used. After 1000 generations, there was at least eight orders of magnitude difference between the error generated by the classical Jacobi-based SR method and that produced by the proposed JBUA hybrid algorithm.

Table 4.1: Comparison of Jacobi-based SR method and proposed JBUA hybrid algorithm

Iteration	Jacobi based SR method with initial		JBUA hybrid algorithm with initial	
	$\alpha_1 = 0.50$	$\alpha_2 = 1.50$	$\alpha_1 = 0.50$	and $\alpha_2 = 1.50$
01	1.83716e+04	2.36221e+04	4.3972e+04	2.26891e+04
100	3.97643e+03	2.05325e+04	9.42877e+00	3.10284e+00
200	3.14860e+01	1.83852e+02	3.52673e-03	1.02643e-03
300	1.06612e+00	4.23743e+01	2.78793e-04	1.19089e-04
400	4.43217e-02	9.54315e+00	1.23254e-05	1.02244e-06
500	1.04843e-02	6.08937e+00	1.80543e-06	1.03781e-06
600	7.55472e-03	4.28310e+00	7.15730e-08	2.43217e-08
700	2.35390e-03	2.61748e+00	3.98569e-09	1.01475e-09
800	1.02362e-03	2.12982e+00	2.25191e-10	1.03283e-10
900	7.27216e-04	1.63231e+00	8.44612e-11	5.00851e-11
1000	1.32542e-04	9.76833e-01	6.96453e-12	3.74284e-12

Table 4.2: Comparison of Jacobi-based SR method and proposed JBUA hybrid algorithm

Iteration	Jacobi based SR method with initial		JBUA hybrid algorithm with initial	
	$\alpha = -1.0$	$\alpha = 1.0$	$\alpha_1 = -1.0$	and $\alpha_2 = 1.0$
01	8.372113e+13	1.18508e+04	1.372113e+12	4.3972e+04
100	Diverge	3.89678e+03	5.22573e+2	2.36573e+2
200	Diverge	1.26643e+02	1.35920e-00	1.01321e-00
300	Diverge	1.75359e+01	2.19745e-02	1.79832e-02
400	Diverge	2.34710e+00	5.66802e-04	3.23152e-04
500	Diverge	9.83765e-01	3.47889e-05	1.89475e-05
600	Diverge	3.26554e-01	2.22358e-07	1.39126e-07
700	Diverge	5.06362e-02	5.89688e-09	3.26786e-09
800	Diverge	1.03243e-02	8.74730e-11	4.82132e-11
900	Diverge	8.68931e-03	3.57647e-12	1.32256e-12
1000	Diverge	1.23574e-03	1.23741e-13	1.19243e-13

There is another interesting observation from the **Table 4.2** that when it was used initial relaxation factor $\alpha_1 = -1.0$ and $\alpha_2 = 1.0$, the proposed hybrid algorithm adapted relaxation factors and converged rapidly. Whereas in classical Jacobi based SR methods, at relaxation factor $\alpha = -1.0$, diverged very rapidly.

Table 4.3: The dynamical change of relaxation factors, α , for corresponding individuals at different generations for proposed JBUA hybrid algorithm

Iteration	Value of α 's for 1st experiment of JBUA algorithm		Value of α 's for 2nd experiment of JBUA algorithm	
	-1.0	1.0	0.5	1.5
1	-1.0	1.0	0.5	1.5
100	0.869122	0.871368	1.039819	1.05214
200	0.972992	0.97667	1.08041	1.08407
300	1.039424	1.043368	1.094001	1.096638
400	1.057982	1.057956	1.086654	1.098117
500	1.060547	1.059654	1.072153	1.085534
600	1.072739	1.068253	1.08393	1.080362
700	1.080413	1.068221	1.082872	1.088507
800	1.085379	1.093159	1.07965	1.070871
900	1.089493	1.090912	1.087312	1.076113
1000	1.082053	1.098993	1.051892	1.054571

Table 4.3 shows how relaxation factors changed dynamically as proposed JBUA hybrid algorithm progressed.

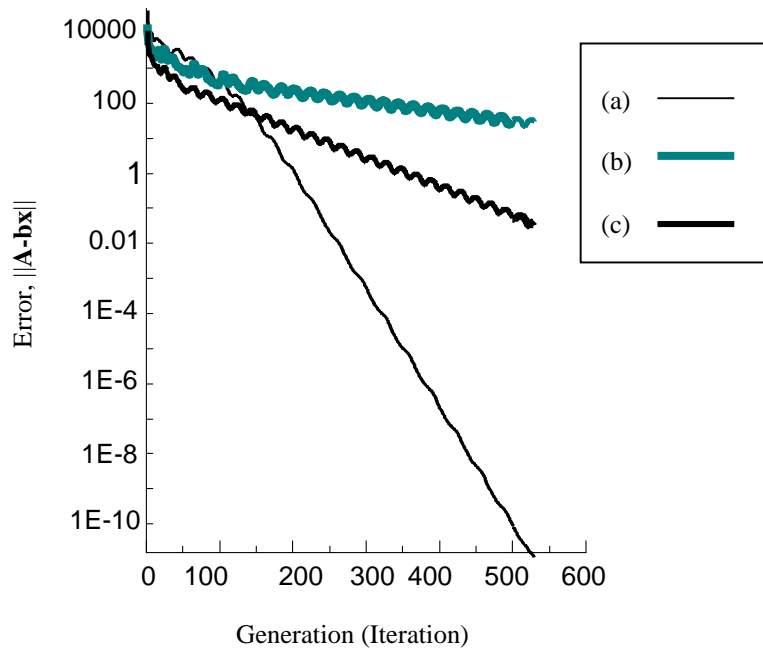


Figure 4.1: Curve (a) represents proposed JBUA hybrid generation history, curve (b) represents classical Jacobi-SR iteration history and curve (c) represents classical Jacobi iteration history.

Fig. 4.1 represents the decreasing of error, for the problem P_1 , produced by the proposed JBUA hybrid algorithm with initial relaxation factors $\omega_1 = 0.5$ and $\omega_2 = 1.5$, Classical Jacobi based SR method with relaxation factor $\omega = 1.5$ and Classical Jacobi method (i.e. relaxation factor $\omega = 1.0$). It is clear, from **Fig. 4.1**, that proposed JBUA hybrid algorithm outperformed the classical Jacobi based SR method ($\omega = 1.0$) as well as Classical Jacobi method ($\omega = 1.0$). It also shows that classical SR technique was extremely sensitive to the relaxation factor ω , while the proposed JBUA hybrid algorithm was very less sensitive against initial values of relaxation factors. This indicates that the simple adaptation scheme for relaxation factors had worked quite effectively in the proposed JBUA hybrid algorithm.

To evaluate the proposed JBUA hybrid algorithm further, eleven test problems, labeled from P_1 to P_{11} , with dimension, $n=100$ were considered. For each test problem $P_i : i = 1, 2, \dots, 11$, the elements of the coefficient matrix \mathbf{A} and elements of

the constant vector \mathbf{b} were all generated uniformly and randomly within given boundaries (shown in 2nd column with corresponding rows of **Table 4.4**). But each coefficient matrix \mathbf{A} , constant vector \mathbf{b} and initial population \mathbf{X} were identical for each comparison. Initial relaxation factors are set at $\alpha_1=0.5$ and $\alpha_2=1.5$ for all the cases. For different problems $P_1 - P_{11}$ different threshold of errors, η , were allowed. **Table 4.4** shows the comparison of the number of generation (iteration) and relative elapsed time needed by the GSBUA hybrid algorithm and that needed by proposed JBUA hybrid algorithm to solve the linear equations (4.3.1) to the given preciseness, γ (see column three of **Table 4.4**). One observation can be made immediately from the **Table 4.4** that, the numbers of generations are comparable in each case in the both hybrid algorithms (proposed JBUA and existing GSBUA algorithms).

Table 4.4: Comparison between existing GSBUA and proposed JBUA hybrid algorithms for several randomly generated test problems

Label of Test Function	Domain of the elements of the coefficient matrix \mathbf{A} & right side constant vector \mathbf{b} of test Problems			Threshold Error, η	GSBUA Hybrid Algorithm	Proposed JBUA Hybrid Algorithm
					Number of Generation	Number of Generation
P ₁	$a_{ii}=2n$;	$a_{ij}=j$;	$b_i=i$	10^{-12}	530	580
P ₂	$a_{ii}=70$;	$a_{ij} \in (-10,10)$,	$b_i \in (-70,70)$	10^{-12}	166	190
P ₃	$a_{ii} \in (50,100)$;	$a_{ij} \in (-10,10)$;	$b_i \in (-100,100)$	10^{-12}	85	91
P ₄	$a_{ii} \in (1,100)$;	$a_{ij} \in (-2,2)$;	$b_i = 2$	10^{-12}	559	586
P ₅	$a_{ii} = 200$;	$a_{ij} \in (-30,30)$;	$b_i \in (-400,400)$	10^{-11}	156	175
P ₆	$a_{ii} \in (-70,70)$;	$a_{ij} \in (0,4)$;	$b_i \in (0,70)$	10^{-08}	801	816
P ₇	$a_{ii} \in (-200,200)$;	$a_{ij} \in (-10,10)$;	$b_i \in (-100,100)$	10^{-11}	189	200
P ₈	$a_{ii} \in (-100,100)$;	$a_{ij} \in (-10,10)$;	$b_i \in (-200,200)$	10^{-06}	5683	5711
P ₉	$a_{ii} \in (10,50)$;	$a_{ij} \in (5,8)$;	$b_i \in (-200,200)$	10^{-11}	618	655
P ₁₀	$a_{ii} \in (100,300)$;	$a_{ij} \in (-50,50)$;	$b_i \in (-100,100)$	10^{-11}	1508	1832
P ₁₁	$a_{ii} \in (200,300)$;	$a_{ij} \in (-100,100)$;	$b_i \in (-100,100)$	10^{-11}	798	870

4.4 Parallel Processing

Parallel searching is one of the main properties of Evolutionary Computational (EC) techniques. As computers can be used for parallel searching by using parallel processors, so EC techniques can be used to solve various kinds of complex problems. For available of parallel processors, recently, evolutionary algorithms are well developed and successfully used to solve so many real world problems. Though individuals of population can be implemented in parallel processing environment for both GSBUA hybrid algorithm and JBUA hybrid algorithm. But GSBUA hybrid algorithm cannot be implemented in parallel processing environment inherently. Since classical Gauss-Seidel method, cannot be implemented in parallel processing environment efficiently. Whereas, proposed JBUA hybrid algorithm can be implemented in parallel processing environment inherently and efficiency of proposed JBUA is near to one. Since Jacobi method can be implemented in parallel processing environment [Gerald and Wheatley (1994)]. As a result by using parallel processors it can be reduced large amount of times for each iteration of JBUA algorithm. For example, if n^2 processors are available, then JBUA hybrid algorithm reduces the time for each iteration to $\log_2 n$ time units. This is a significant speedup over the sequential algorithm (as GSBUA hybrid algorithm inherently), which requires n^2 time units per iteration [Gerald and Wheatley (1994)].

4.5 Summary

In this chapter, a new hybrid evolutionary algorithm called JBUA hybrid evolutionary algorithm has been proposed to solve system of linear equations and have been tested on various randomly generated test problems. This hybrid algorithm is developed on the based of classical Jacobi-based SR method. The effectiveness of this hybrid algorithm is compared with that of classical Jacobi based SR method and GSBUA hybrid algorithm. This preliminary investigation has showed that this algorithm outperforms the classical numerical methods as well as GSBUA hybrid algorithm. Another significant property of this proposed algorithm is that this algorithm can be implemented in parallel processing environment inherently.

Gauss-Seidel Based Time Variant Adaptive Hybrid Algorithm

5.1 Introduction

In last three decades, effectiveness of evolutionary algorithms (EAs) has induced many people to believe that they are the methods of choice for hard real-life problems superseding traditional search techniques. However, they are not without their limitations. In particular, the choice of a good evolutionary operator can make a considerable difference to the exploration and exploitation, and often the feasibility of the evolutionary search. Moreover, the success and progress of an evolutionary search algorithm mostly depends upon the balance between population diversity and selective pressure [Michakewicz (1996)]. To meet these requirements, contemporary EAs [Bäck and Schwefel (1993), Bäck et. al. (1993), Rechenberg (1994), Schwefel et. al. (1995), Bäck et. al. (1996), Bäck et. al. (1997)] usually apply self-adaptation in so-called strategy parameters (or internal model) of the object variables. While optimizing the objective function, the self-adaptation techniques require optimization of the strategy parameters as well [Poil and Logan (1996)].

But In GSBUA hybrid algorithm [He. et. al. (2000)], relaxation factors (i.e. strategy parameters) are self-adapted by Uniform Adaptation (UA) technique and so there is no guaranty of optimization of the strategy parameters i.e. relaxation factors. On the other hand, empirical studies [Chellapilla and Fogel (1997), Yao et. al. (1997)] showed that Cauchy mutations are effective for early stages of the evolution and Gaussian mutations are essential for final stages of the evolution. Advantages of these mutations are utilized by a linear combination of them [Chellapilla (1998a)]. But this method is somewhat cumbersome to implement by general people.

To eliminate above-mentioned problems, this chapter is devoted to developing a Gauss-Seidel based Time-Variant Adaptive hybrid evolutionary algorithm. This algorithm uses a time-variant adaptation scheme based on observed natural phenomena. The mutation strategy parameter i.e. relaxation factors are adapted by time-variant function.

5.2 Development of Time-Variant Adaptive Parameters

5.2.1 Basic Notion

The inherent strength of EA – towards convergence and high precision results – lies in the choice of the mutation steps, i.e. standard deviation [Rechnberg (1994), Fogel (1995a), Bäck et.al. (1997), Yao et. al. (1997)]. An obvious biological evidence is that a rapid change is observed at early stages of life and a slow changes 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. By mimicking this emergent natural evidence, a special dynamic time variant adaptive operator is proposed aiming at both improving the fine local tuning and reducing the disadvantages of uniform adaptation [Hashem (1999), Michalewicz (1996), Bäck et.al. (1997)]. 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. But Hashem (1999), Michalewicz (1996), Bäck et.al. (1997) discussed about time variant mutation in global optimization problems. In this section, a new Time-Variant Adaptive (TVA) parameter is introduced 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.

5.2.2 Formulas

The time variant adaptive (TVA) parameters are defined as

$$p_x = E_x \times N(0,0.25) \times T \quad (5.2.1)$$

and is denoted as adaptive (TVA) probability parameter of x , and

$$p_y = E_y \times | N(0,0.25) | \times T \quad (5.2.2)$$

and is denoted as adaptive (TVA) probability parameter of p_y

$$\text{Where } T_{\check{S}} = \left\lceil \ln\left(1 + \frac{1}{t + \check{S}}\right) \right\rceil, \quad t > 10 \quad (5.2.3)$$

$$\text{or } T = \left(1 - \frac{t}{T}\right)^x \quad (5.2.4)$$

Here \check{S}_x and \check{S}_y 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 \check{S}_x and \check{S}_y respectively. And if \check{S}^* is denoted as the optimal relaxation factor then

$$E_x = p_x |_{\max} = \frac{\check{S}_y \sim \check{S}_x}{2(\check{S}_x + \check{S}_y)}, \text{ so that } p_x^m = (0.5 + p_x |_{\max})(\check{S}_x + \check{S}_y) \approx \check{S}_y \text{ and}$$

$$E_y = p_y |_{\max} = \frac{\check{S}^* \sim \check{S}_y}{\check{S}_y - \check{S}_L} \quad \text{or} \quad \frac{\check{S}^* \sim \check{S}_y}{\check{S}_U - \check{S}_y}, \text{ so that}$$

$$\check{S}^* \approx p_y^m = \begin{cases} p_y |_{\max} (\check{S}_U - \check{S}_y), & \text{when } \check{S}_y > \check{S}_x \\ p_y |_{\max} (\check{S}_L - \check{S}_y), & \text{when } \check{S}_y < \check{S}_x \end{cases}$$

5.2.3 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 for Eqn. (5.2.3) i.e. $T_{\xi} = \lambda \ln(1 + 1/(t + \lambda))$ (denoted as Lambda based TVA parameter) characteristics of this equation for $\lambda = 20, 50, 200$ are shown in **Fig. 5.1**. It is observed in the **Fig. 5.1** that when the value of λ is small then, initially, rate of change of the function $T_{\xi} = \lambda \ln(1 + 1/(t + \lambda))$ is very rapid; on the other hand when the value of λ is relatively large, then initially, rate of change of the this function is relatively slow. For all the cases, in later stages, the rate of change is slow. The variation of p_x and p_y respectively for $\lambda = 50$ are relatively fine for this function and the graphical representations of characteristics of this function are shown in **Fig.5.3** and **Fig.5.4** respectively.

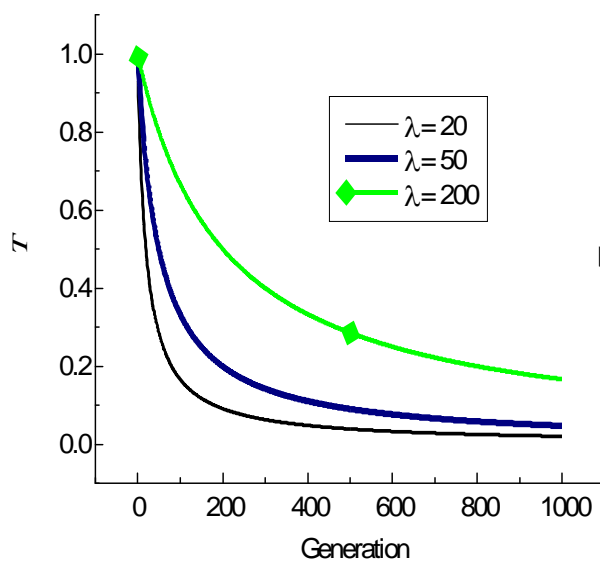


Figure 5.1: Rate of change of the variation of T for various λ

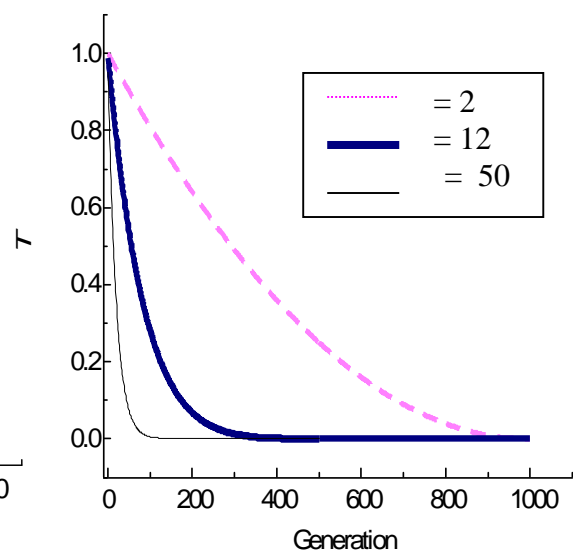


Figure 5.2: Rate of change of the variation of T for various γ

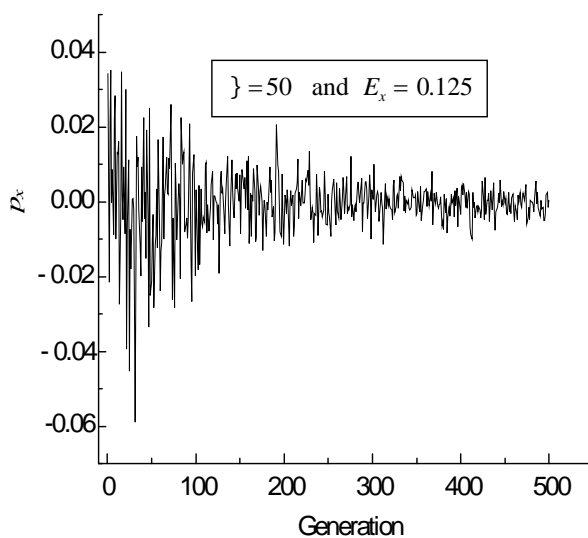


Figure 5.3: Rate of change of the variation of p_x for $\lambda = 50$ and $E_x = 0.125$

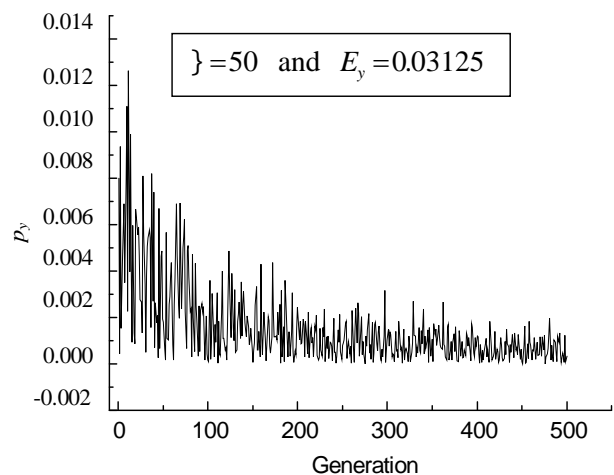


Figure 5.4: Rate of change of the variation of p_y for $\lambda = 50$; $E_y = 0.03125$

Again for Eqn. (5.2.4) i.e. $T = (1-t/T)^\gamma$ (denoted as Gamma based TVA parameter) the characteristics of this equation for $\gamma = 2, 12, 50$ are shown in **Fig. 5.2** where T is set at 3000. It is shown in the **Fig. 5.2** that when the value of γ is large then initially, rate of change of this function is very rapid; on the other hand when the value of γ 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 γ . The variation of p_x and p_y for $\gamma = 12$, are relatively fine for this function and graphical representations are shown in **Fig.5.5** and **Fig.5.6** respectively.

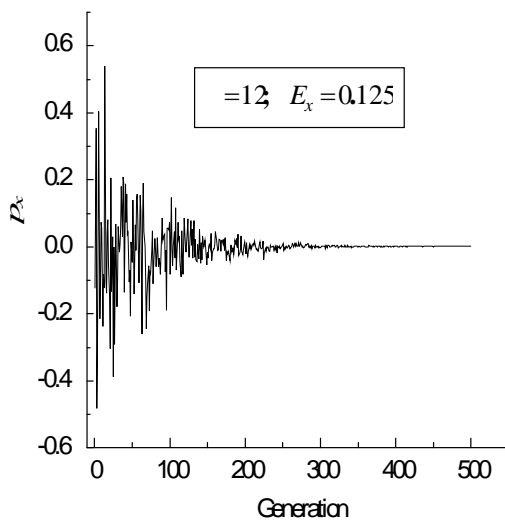


Figure 5.5: Rate of change of the variation of p_x for $\gamma=12$ and $E_x=0.0315$

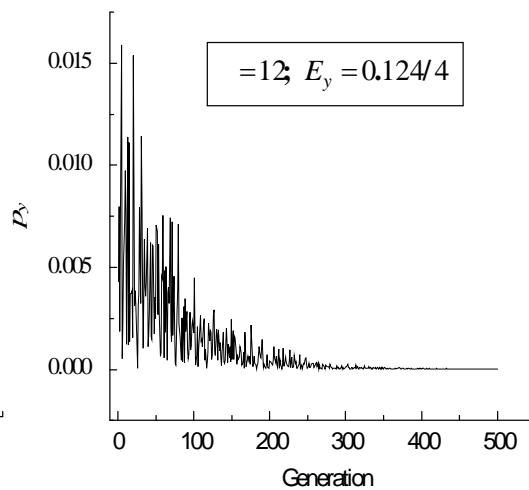


Figure 5.6: Rate of change of the variation of p_y for $\gamma=12$ and $E_y=0.03145$

5.3 Proposed Method

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) (see Chapter 4). In sequel, it is described here elaborately.

5.3.1 The Basic Equations of Gauss-Seidel Based SR Method

The system of linear Eqn. (1.1.1) can be rewrite as

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad (i = 1, 2, \dots, n) \quad (5.3.1)$$

In Gauss-Seidel based SR method (see App. C) [Engeln-Müllges, and Uhlig (1996)] Eqn. (5.3.1) is given by

$$x_i^{(k+1)} = (1 - \check{S})x_i^{(k)} + \frac{\check{S}}{a_{ii}} \left(- \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} + b_i \right), \quad (5.3.2)$$

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

In matrix form Eqn. (5.3.2) can be rewrite in matrix-vector equation as (see §2.5, Eqn. (2.5.18)):

$$\mathbf{x}^{(k+1)} = \mathbf{H}_S \mathbf{x}^{(k)} + \mathbf{V}_S, \quad (5.3.3)$$

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

$$\mathbf{H}_S = (\mathbf{I} + \mathbf{D}^{-1}\mathbf{L})^{-1} \{ (1 - \check{S})\mathbf{I} - \mathbf{D}^{-1}\mathbf{U} \} \quad (5.3.4)$$

and

$$\mathbf{V} = (\mathbf{I} + \mathbf{D}^{-1}\mathbf{L})^{-1} \mathbf{D}^{-1}\mathbf{b} \quad (5.3.5)$$

5.3.2 The Algorithm

Similar to many other evolutionary algorithms, the proposed GSBTVA hybrid algorithm also always maintains a population of approximate solution to linear equations. The initialization of population and recombination mechanisms of this proposed 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. (5.3.3). The mutation is stochastic since 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 as described in § 4.2.3. 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 follows:

Step 1: Initialization

Generate, randomly from \mathfrak{R}^n , an initial population of approximate solutions to the linear Eqn. 5.3.1 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 $k \leftarrow 0$ where k is the generation counter. And initialize corresponding relaxation factor \check{S} as:

$$\check{S}_i = \begin{cases} L + \frac{d}{2} & \text{for } i = 1 \\ \check{S}_{i-1} + d & \text{for } 1 < i \leq N \end{cases} \quad (5.3.6)$$

Where $d = \frac{U - L}{N}$

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:

$$\mathbf{X}^{(k+c)} = \mathbf{R}(\mathbf{X}^{(k)})' \quad (5.3.7)$$

where \mathbf{R} is a stochastic matrix (§ 4.2.3). 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. (5.3.3))

$$\mathbf{x}_i^{(k+m)} = \mathbf{H}_i \mathbf{x}_i^{(k+c)} + \mathbf{V}_{S_i}, \quad i = 1, 2, \dots, N. \quad (5.3.9)$$

where \mathbf{H}_i is called Gauss-Seidel iteration matrix corresponding \check{S}_i and given by

$$\mathbf{H}_{S_i} = (\mathbf{I} + \check{S}_i \mathbf{D}^{-1} \mathbf{L})^{-1} \{ (1 - \check{S}_i) \mathbf{I} - \check{S}_i \mathbf{D}^{-1} \mathbf{U} \} \quad (5.3.10)$$

and

$$\mathbf{V}_i = \check{S}_i (\mathbf{I} + \check{S}_i \mathbf{D}^{-1} \mathbf{L})^{-1} \mathbf{D}^{-1} \mathbf{b} \quad (5.3.11)$$

Here \check{S}_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 relaxation factors \check{S}_x and \check{S}_y and $\|e(\mathbf{x}^m)\|$ and $\|e(\mathbf{y}^m)\|$ are their corresponding errors (fitness value).

Then the relaxation factors \check{S}_x and \check{S}_y are adapted as follows:

(a) If $\|e(\mathbf{x}^m)\| > \|e(\mathbf{y}^m)\|$, (i) then move x toward \check{S}_y by using

$$x^m = (0.5 + p_x)(x + \check{S}_y) \quad (5.3.12)$$

and (ii) move y away from x using

$$y^m = \begin{cases} y + p_y(\check{S}_x - y), & \text{when } y > x \\ y + p_y(\check{L}_x - y), & \text{when } y < x \end{cases} \quad (5.3.13)$$

Where $p_x = E_x \times N(0, 0.25) \times T$, and $p_y = E_y \times |N(0, 0.25)| \times T$, as define above (Eqn. (5.2.1) and Eqn. (5.2.2)).

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

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

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 parents 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.

The pseudo-code structure of this hybrid algorithm is shown in **Fig.C.2** (Appendix C), where \mathbf{H}_{ξ_i} , \mathbf{V}_{ξ_i} and adaptive probability parameters p_x and p_y are defined by Eqn.'s (5.3.10), (5.3.11), (5.2.1) and (5.2.2) respectively.

5.4 Performance of the Proposed Algorithm

In order to evaluate the effectiveness and efficiency of the proposed GSBTVA hybrid

algorithm, numerical experiments had been carried out on a number of problems to solve the systems of linear Eqn. (2.2.3) of the form:

$$\mathbf{Ax} = \mathbf{b} \quad (5.4.1)$$

The hybrid algorithms (GSBUA and GSBTVA) used in all of our experiments were very simple and had population size N is two. That is, only two individuals were used. Boundary of Relaxation factors α_L and α_U were set at 0 and 2 respectively so that initial α 's became $\alpha_L = 0.5$ and $\alpha_U = 1.5$ respectively in all of the experiments. Also the approximate initial boundary, E_x and E_y were set at 0.125 and 0.03125 respectively in all of the experiments. Moreover for time variant parameter (TVA) $T_S = \ln\{1+1/(t+1)\}$, was set at 50 and for time variant parameter $T = (1-t/T)^x$, was set at 12.0 and maximum generation, T , was set at 2500 in all of the experiments. The stochastic matrix \mathbf{R} was same as Eqn.'s (4.3.2 – 3) (see § 4.3). The following settings were also valid all through the experiments:

The dimension of unknown variables was $n = 100$, each individual \mathbf{x} of population \mathbf{X} was initialized from the domain $\mathfrak{R}^{100} \in (-30, 30)$ randomly and uniformly. For each experiment, a total of ten independent runs were conducted and the average results are reported here.

First problem was to solve linear equations, Eqn. (5.4.1), where $a_{ii} = 2n$ and $b_i = i$ for $i = 1, \dots, n$ and $a_{ij} = j$ for $i \neq j$, $i, j = 1, \dots, n$ (i.e. problem P_1). The problem was to be solved with an error smaller than $\gamma = 10^{-12}$ (threshold error). **Fig. 5.7** shows the numerical results (in graphical form) achieved by the proposed time variant adaptive (TVA) based hybrid algorithm (GSBTVA) and the Uniform Adaptation (UA) based hybrid algorithm (GSBUA) – (A) proposed GSBTVA hybrid algorithm where TVA parameter was defined by $T_S = \ln\{1+1/(t+1)\}$, (B) proposed GSBTVA hybrid

algorithm where TVA parameter was defined by $T = (1-t/T)^x$ and (C) GSBUA hybrid algorithm [He et. al. (2000)].

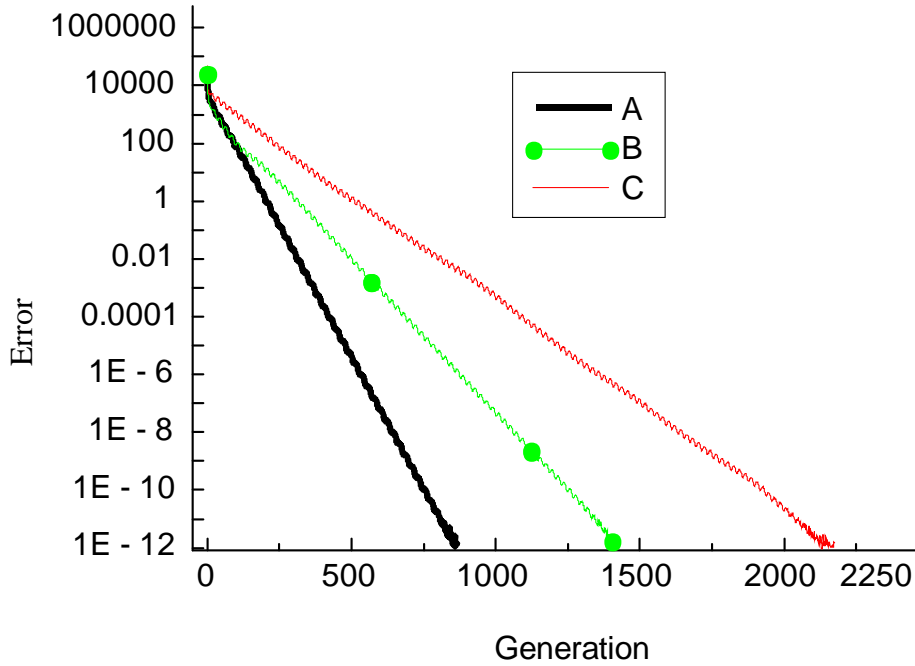


Figure 5.7: Curve (A) represents the evolution history of proposed GSBTVA (used Lambda based TVA parameter) hybrid algorithm, curve (B) represents the evolution history of proposed GSBTVA (used Gamma based TVA parameter) hybrid algorithm and curve (C) represents the evolution history of GSBUA hybrid algorithm.

It is observed in **Fig. 5.7** that the rate of convergence of proposed algorithm with both TVA parameters is much better than that of UA-based algorithm and TVA-based algorithm exhibits a fine local tuning. Another observation is that the TVA parameter $T_{\xi} = \lfloor \ln\{1+1/(t+\xi)\} \rfloor$ make algorithm faster than TVA parameter $T = (1-t/T)^x$. Also for former TVA parameter, algorithm needs not pre-estimate maximum generation, T. On the other hand for later TVA parameter, algorithm need pre-estimate T.

Table 5.1 presents six test problems, labeled from P_1 to P_6 , with dimension, $n = 100$. For each test problem P_i : $i = 1, 2 \dots 6$, the coefficient matrix \mathbf{A} and constant vector \mathbf{b} were all generated uniformly and randomly within given domains (shown in 2nd column with corresponding rows of **Table 5.1**). For different problems (P_1 – P_6) corresponding threshold errors, η , is shown in the **Table 5.1**. Also note that the TVA parameter was defined as Eqn. (5.2.3). **Table 5.1** shows the comparison of the number

of generation (iteration) and relative elapsed time used by the GSBUA hybrid algorithm and by proposed GSBTVA hybrid algorithm to the given preciseness, y (see column three of the **Table 5.1**). One observation can be made immediately from the **Table 5.1**, except for problem P_3 where the GSBUA algorithm performed near to same as proposed GSBTVA algorithm, proposed TVA-based hybrid algorithm (GSBTVA) performed much better than the UA-based hybrid algorithm (GSBUA) for all other problems.

Table 5.1
Comparison of existing GSBUA and proposed GSBTVA hybrid algorithms for several randomly generated test problems

Label of Test Problems	Domain of the elements of the coefficient matrix \mathbf{A} & right side constant vector \mathbf{b} of test Problems			Threshold Error, η	GSBUA Alg.		GSBTVA Alg.	
					Elapsed Time	Generation (elapsed)	Elapsed Time	Generation (elapsed)
P_1	$a_{ii} = 2n$;	$a_{ij} = j$;	$b_i = i$	10^{-12}	107	1812	60	910
P_2	$a_{ii} \in (-70, 70)$;	$a_{ij} \in (-2, 2)$;	$b_i \in (-2, 2)$	10^{-12}	18	297	06	108
P_3	$a_{ii} \in (-70, 70)$;	$a_{ij} \in (0, 4)$;	$b_i \in (0, 70)$	10^{-12}	29	465	27	434
P_4	$a_{ii} \in (1, 100)$;	$a_{ij} \in (-2, 2)$;	$b_i = 2$	10^{-12}	75	1260	39	445
P_5	$a_{ii} = 200$;	$a_{ij} \in (-30, 30)$;	$b_i \in (-400, 400)$	10^{-11}	35	596	21	359
P_6	$a_{ii} \in (-70, 70)$;	$a_{ij} \in (0, 4)$;	$b_i \in (0, 70)$	10^{-09}	350	5719	191	3236

- Elapsed Times are shown, in column four and in column six, just for relative comparison of two algorithms. It is measure in second Fraction time of second is not counted.

Fig. 5.8 – 5.9 show the nature of self-adaptation of relaxation factors used in the GSBUA hybrid algorithm and **Fig. 5.10 – 5.11** show the nature of self-adaptation of relaxation factors used in the proposed GSBTVA hybrid algorithm for $\alpha_1 = 0.5$ and $\alpha_2 = 1.5$ respectively. It is observed in **Fig. 5.8 – 5.9** and **Fig. 5.10 – 5.11** that the self-adaptation process of relaxation factors in TVA-based hybrid algorithm is much better than that of in UA-based hybrid algorithm. **Fig. 5.10 – 5.11** show that how $\alpha_1 = 0.5$ and $\alpha_2 = 1.5$ were adapted to its near optimum value by the time variant adaptive process and reached to near optimal position for which rate of convergence was accelerated. On the other hand, **Fig. 5.8 – 5.9** show that initially $\alpha_1 = 0.5$ and $\alpha_2 = 1.5$, by uniform self-adaptation process, did not gradually reach to the near optimal position.

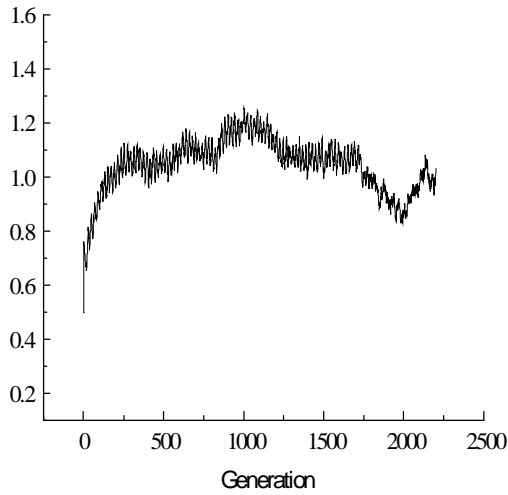


Figure 5.8: Self-adaptation of $\check{S}_1 = 0.5$ in the UA-based Algorithm

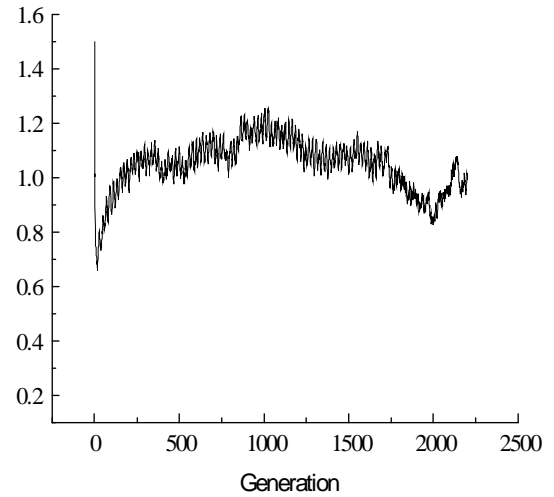


Figure 5.9: Self-adaptation of $\check{S}_2 = 1.5$ in the UA-based Algorithm

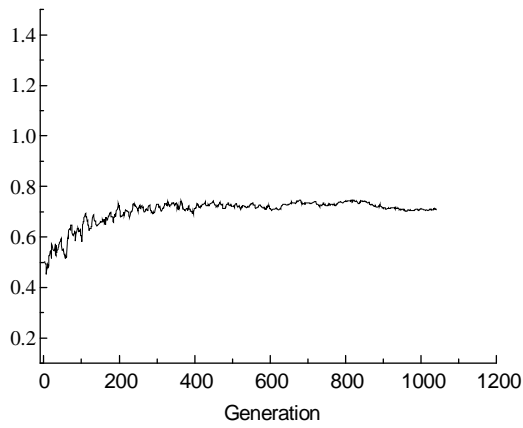


Figure 5.10: Self-adaptation of $\check{S}_1 = 0.5$ in the TVA-based Algorithm

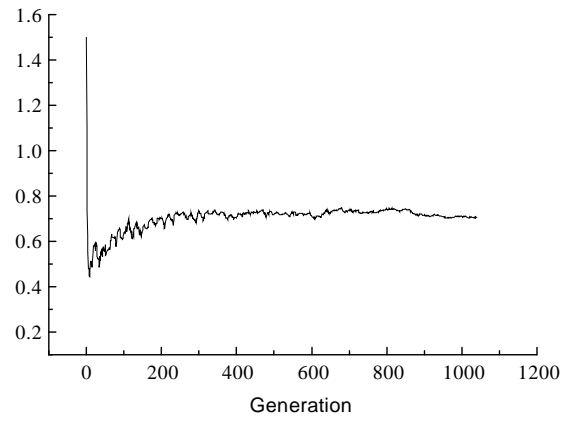
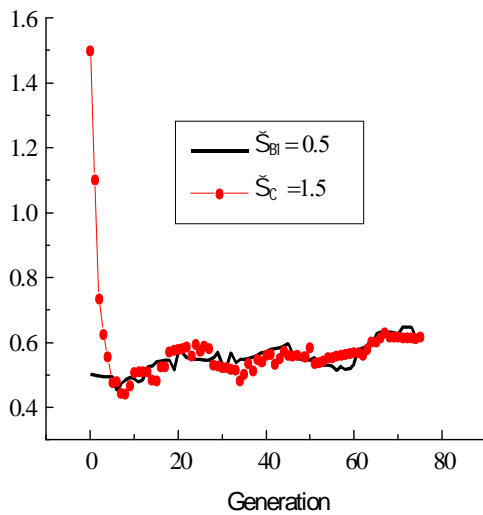
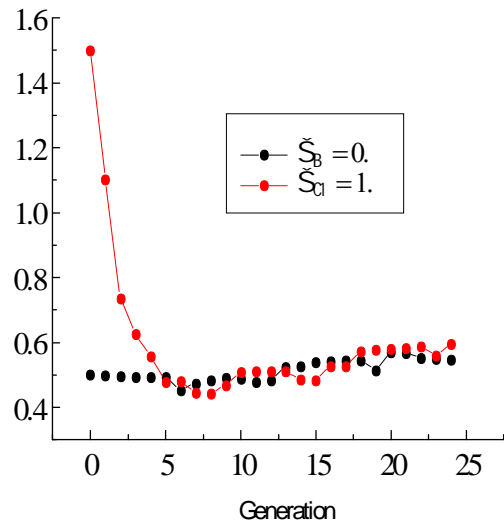


Figure 5.11: Self-adaptation of $\check{S}_2 = 1.5$ in the TVA-based Algorithm.



(a)



(b)

Figure 5.12: A graphical view of self-adaptation process of relaxation factors $\check{S}_1 = 0.5$ and $\check{S}_2 = 1.5$ in the GSBTVA-Algorithm

Fig. 5.12 shows also the graphical representation of self- adaptation process of the relaxation factors $\alpha_1 = 0.5$ and $\alpha_2 = 1.5$ in the proposed GSBTVA algorithm explicitly. It is observed that α 's were adapted gradually to its approximate optimal relaxation factor 0.7

5.5 Summary

In this paper, two time-variant adaptive (TVA) parameters are developed. By these two parameters, a Gauss-Seidel based time-variant adaptive hybrid evolutionary algorithm has been proposed for solving linear equations. The proposed GSBTVA hybrid algorithm integrates the classical Gauss-Seidel based SR method with evolutionary computation techniques. The time-variant based adaptation is introduced for adaptation of relaxation factors, which makes the algorithm more natural and accelerates its rate of convergence. The recombination operator in the algorithm mixed two parents by a kind of averaging, which is similar to the intermediate recombination often used in evolution strategies. The mutation operator is equivalent to one iteration in the classical Gauss-Seidel based SR method. The mutation is stochastic and time variant since the relaxation factors used in mutation are adapted stochastically and the time variant adaptation is used to adept relaxation factor α . The proposed TVA-based relaxation factor adaptation technique acts as a local fine tuner and helps to escape from the disadvantage of uniform adaptation. This preliminary investigation has showed that the proposed GSBTVA hybrid algorithm performs better than the GSBUA hybrid algorithm. Also GSBTVA hybrid algorithm is more efficient and robust than the GSBUA hybrid algorithm.

Jacobi Based Time Variant Adaptive Hybrid Algorithm

6.1 Introduction

In Chapter 4, a Jacobi based hybrid evolutionary algorithm has been proposed where a uniform adaptation technique is used. On the other hand in chapter 5, a Gauss-Seidel based hybrid evolutionary algorithm has been proposed where a time variant adaptation is used. In chapter 5, it is observed that a time-variant adaptive based algorithm outperforms the uniform adaptive based algorithm. And it is also discussed in chapter 4, that in parallel environment Jacobi based uniform adaptive algorithm outperforms the Gauss-Seidel based uniform adaptive algorithm.

This chapter is devoted to developing a new hybrid evolutionary algorithm. This hybrid algorithm uses Jacobi based time variant adaptive (JBTVA) algorithm instead of Gauss-Seidel based time variant adaptive (GSBTVA) algorithm. As Jacobi based SR method can be implemented in parallel processing environment efficiently, so proposed JBTVA hybrid evolutionary algorithm can be implemented in parallel processing environment efficiently.

6.2 Proposed Method

In this section, a new hybrid evolutionary algorithm is proposed in which Jacobi based SR method, evolutionary computation techniques and time variant adaptation techniques are used. This proposed Jacobi-based hybrid evolutionary algorithm also does not require a user to guess or estimate the optimal relaxation factor ω . The proposed 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.

6.2.1 The Algorithm

The main steps of the Jacobi Based Time-Variant Adaptive (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 GSBTVA algorithm (see § 5.3). And Mutation mechanism is same as that of JBUA algorithm (see § 4.2). The pseudo-code structure of this hybrid algorithm is shown in **Fig.C.2** (Appendix C), where \mathbf{H}_{S_i} , \mathbf{V}_{S_i} are defined by Eqn.’s (4.2.10) and (4.2.11) (see § 4.2), and adaptive parameters p_x and p_y are defined by Eqn.’s (5.2.1) and (5.2.2) (see § 5.2) respectively.

6.3 Performance of the Proposed Algorithm

In order to evaluate the effectiveness of the proposed JBTVA hybrid algorithm, numerical experiments had been carried out on a number of problems to solve the systems of linear Eqn. (2.2.3) of the form:

$$\mathbf{Ax} = \mathbf{b} \tag{6.3.1}$$

The hybrid algorithms (JBUA and JBTVA) used in all of our experiments were very simple and had population size N is two. That is, only two individuals were used. Boundary of Relaxation factors L and U were set at 0 and 2 respectively so that initial α ’s became $\alpha_1 = 0.5$ and $\alpha_2 = 1.5$ respectively in all of the experiments. Also the approximate initial boundary, E_x and E_y were set at 0.125 and 0.03125 respectively in all of the experiments. Moreover for time variant parameter (TVA)

$T = \ln\{1+1/(t +)\}$ (see Eqn. 5.2.3), was set at 50 in all of the experiments. Each individual was initialized from the domain $\mathfrak{R}^{100} \in (-30,30)$ randomly and uniformly. The stochastic matrix \mathbf{R} was same as Eqn.'s (4.3.2 – 3). The following settings were also valid all through the experiments:

The dimension of unknown variables was $n = 100$. For each experiment, a total of ten independent runs were conducted and the average results are reported here.

First problem was to solve linear equations, Eqn. (6.3.1), where parameters were randomly selected from the domain $a_{ii} \in (-70,70)$ and $b_i \in (0,70)$ for $i = 1, \dots, n$ and $a_{ij} \in (0,7)$; for $i \neq j$, $i, j = 1, \dots, n$ (i.e. problem P_2). The problem was to be solved with an error smaller than 10^{-6} (threshold error). **Fig.** 6.1 shows the numerical results (in graphical form) achieved by the proposed JBTVA (Jacobi based time-variant adaptive) hybrid algorithm and JBUA (Jacobi based Uniform Adaptation) hybrid algorithm. It is observed in **Fig.** 6.1 that the rate of convergence of TVA-based algorithm are better than that of UA-based algorithm.

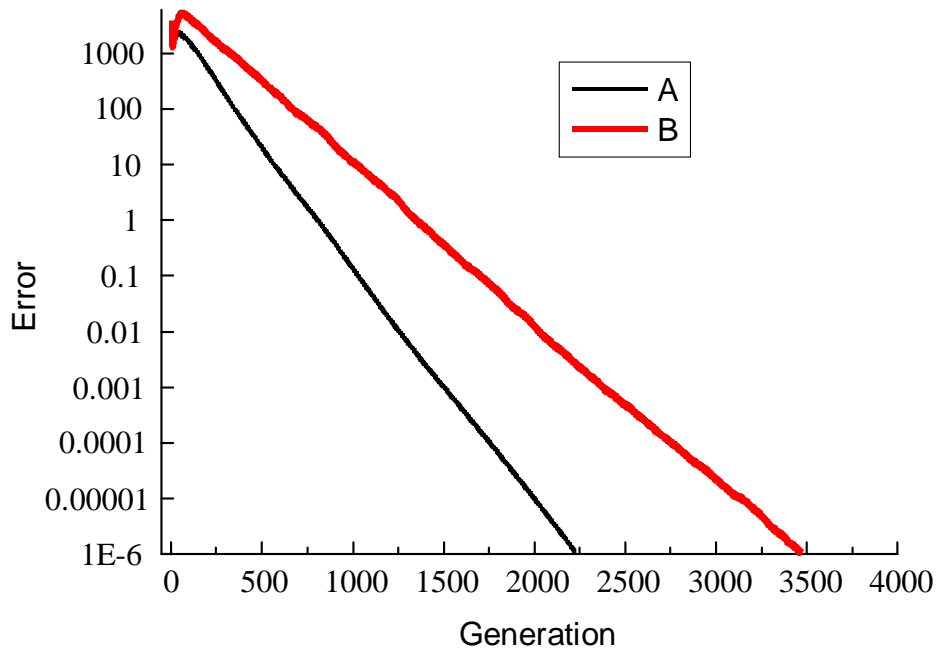


Figure 6.1: Curve (A) represents the evolution history of proposed JBTVA hybrid algorithm and curve (B) represents the evolution history of JBUA hybrid algorithm.

Table 6.1 presents ten test problems, labeled from P_1 to P_{10} , with dimension, $n = 100$. For each test problem P_i : $i = 1, 2, \dots, 10$, the coefficient matrix \mathbf{A} and constant vector \mathbf{b} were all generated uniformly and randomly within given domains (shown in 2nd column with corresponding rows of **Table 6.1**). For different problems (P_1 – P_{10}) corresponding threshold errors, η , is shown in the **Table 6.1**. This table shows the comparison of the number of generation (iteration) and relative elapsed time used by the JBUA and proposed JBTVA hybrid algorithms to the given preciseness, γ (see column three of the **Table 6.1**). One observation can be made immediately from this table, except for problem P_{10} where the JBUA algorithm performed near to same as JBTVA algorithm, proposed JBTVA hybrid algorithm performed much better than the JBUA hybrid algorithm for all other problems.

Table 6.1
Comparison between JBUA and proposed JBTVA hybrid algorithms for several randomly generated test problems

Label of Test	Domain of the elements of the coefficient matrix \mathbf{A} & right side constant vector \mathbf{b} of test Problems	Threshold Error, η	JBUA Alg.		JBTVA Alg.	
			Elapsed Time (in Second)	Generation (elapsed)	Elapsed Time (in Second)	Generation (elapsed)
P_1	$a_{ii} \in (50, 100)$; $a_{ij} \in (-20, 20)$; $b_i \in (-100, 100)$	10^{-11}	10.556	1765	3.068	666
P_2	$a_{ii} \in (-70, 70)$; $a_{ij} \in (0, 7)$; $b_i \in (0, 70)$	10^{-6}	15.797	3468	10.177	2224
P_3	$a_{ii} \in (-100, 100)$; $a_{ij} \in (-10, 10)$; $b_i \in (-100, 100)$	10^{-6}	20.76	5052	14.009	3060
P_4	$a_{ii} \in (2, 10)$; $a_{ij} \in (-2, 2)$; $b_i \in (-5, 5)$	10^{-6}	2.236	484	2.021	435
P_5	$a_{ii} \in (2, 3)$; $a_{ij} \in (-1, 1)$; $b_i \in (2, 5)$	10^{-6}	26.111	5743	18.896	4112
P_6	$a_{ii} = 5$; $a_{ij} = \{1, 0\}$; $b_i = 2$	10^{-12}	2.089	439	1.52	330
P_7	$a_{ii} = 20n$; $a_{ij} = j$; $b_i = i$	10^{-12}	0.891	67	0.73	47
P_8	$a_{ii} = 2i^2$; $a_{ij} = j$; $b_i = i$	10^{-12}	1.77	277	1.248	119
P_9	$a_{ii} = 70$; $a_{ij} \in (-10, 10)$; $b_i \in (70, 70)$	10^{-12}	0.794	53	0.641	28
P_{10}	$a_{ii} \in (1, 100)$; $a_{ij} \in (-2, 2)$; $b_i = 2$	10^{-12}	2.390	230	2.272	225

- Elapsed Times are shown, in column four and in column six, just for relative comparison of two algorithms.

6.4 Parallel Processing

As mention earlier (see Chapter 4) that parallel searching is one of the main properties of evolutionary computational techniques. And since classical Jacobi based SR method can be implemented in parallel processing environment efficiently [Gerald and Wheatley (1994)]. So JBTVA, as like as JBUA, can also be implemented, inherently, in parallel processing environment efficiently.

6.5 Summary

In this chapter, a new hybrid evolutionary algorithm called JBTVA hybrid evolutionary algorithm has been proposed to solve system of linear equations and have tested on various test problems. This hybrid algorithm is developed on the based of classical Jacobi-based SR method. The effectiveness of this hybrid algorithm is compared with that of JBUA hybrid algorithm. This preliminary investigation has showed that proposed JBTVA algorithm outperforms JBUA hybrid algorithm. Another significant property of this proposed algorithm is that parallel processors can be implement efficiently and inherently to this algorithm.

Validity of the Algorithms

7.1 Introduction

As mention earlier a solvable problem, solving by iterative method, may converge or diverge depend on the order of system arranged (see Fig.2.1). There is no any necessary condition of a solvable system of linear equations to solve by iterative methods. But there are some sufficient conditions, for iterative methods, for which system of linear equations must be converged [Chapra and Canale (1990)]. Proposed hybrid evolutionary algorithms (JBUA, GSBTVA and JBTVA) are also one kind of iterative method. So these hybrid algorithms follows some theorems so that the algorithms are converged to the required solutions. This chapter is devoted to proposed two theorems and then proved and verified the theorems. One theorem is related to the convergence of the algorithms and another is related to the adaptation of the relaxation factors of the algorithms. These verifications may establish the rapid convergence of the proposed hybrid algorithms.

7.2 Theorems

The following theorem establishes the rapid convergence of the hybrid algorithms.

7.2.1 Convergence Theorem

Theorem-1: If there exist an γ ($0 < \gamma < 1$) such that, for the norm of \mathbf{H} ,

$$\|\mathbf{H}_\zeta\| < \gamma < 1, \text{ then } \lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x}^*,$$

where \mathbf{x}^* is the solution vector to the system of linear equations i.e., $\mathbf{A}\mathbf{x}^* = \mathbf{b}$

Proof: Let $\mathbf{X}^{(k)}$ be the population at k th iteration so that

$$\mathbf{X}^{(k)} = \{\mathbf{x}_i^{(k)} : i = 1, 2, \dots, N\}$$

Let $\mathbf{e}_i^{(k)}$ be the error between the approximate solution $\mathbf{x}_i^{(k)}$ and exact solution \mathbf{x}^* i.e.

$$\mathbf{e}_i^{(k)} = \mathbf{x}_i^{(k)} - \mathbf{x}^*$$

Then according to the recombination

$$\mathbf{x}_i^{(k+c)} = \sum_{j=1}^N r_{ij} \mathbf{x}_j^{(k)}, \quad i = 1, 2, \dots, N$$

$$\therefore \|\mathbf{e}_i^{(k+c)}\| = \|\mathbf{x}_i^{(k+c)} - \mathbf{x}^*\|$$

$$= \left\| \sum_{j=1}^N r_{ij} \mathbf{x}_j^{(k)} - \mathbf{x}^* \right\|$$

$$= \left\| \sum_{j=1}^N r_{ij} \mathbf{x}_j^{(k)} - \sum_{j=1}^N r_{ij} \mathbf{x}^* \right\|$$

$$\therefore \sum_{j=1}^N r_{ij} = 1 \text{ and } r_{ij} \geq 0, \quad i = 1, 2, \dots, N$$

$$\begin{aligned} \therefore \|\mathbf{e}_i^{(k+c)}\| &= \left\| \sum_{j=1}^N r_{ij} (\mathbf{x}_j^{(k)} - \mathbf{x}^*) \right\| \\ &\leq \sum_{j=1}^N \|r_{ij} (\mathbf{x}_j^{(k)} - \mathbf{x}^*)\| \leq \sum_{j=1}^N \|r_{ij}\| \|\mathbf{x}_j^{(k)} - \mathbf{x}^*\| \end{aligned}$$

$$\therefore \|\mathbf{e}_i^{(k+c)}\| \leq \sum_{j=1}^N r_{ij} \|\mathbf{x}_j^{(k)} - \mathbf{x}^*\|$$

$$\therefore \|\mathbf{e}_i^{(k+c)}\| < \max\{\|\mathbf{e}_j^{(k)}\|; j = 1, \dots, N\}$$

Again according to mutation for $i = 1, 2, \dots, N$

$$\mathbf{x}_i^{(k+m)} = \mathbf{H}_i \mathbf{x}_i^{(k+c)} + \mathbf{V}_i$$

and also since $\mathbf{x}^* = \mathbf{H}_i \mathbf{x}^* + \mathbf{V}_i$ then

$$\mathbf{x}_i^{(k+m)} - \mathbf{x}^* = \mathbf{H}_i (\mathbf{x}_i^{(k+c)} - \mathbf{x}^*)$$

$$\therefore \|\mathbf{x}_i^{(k+m)} - \mathbf{x}^*\| = \|\mathbf{H}_i (\mathbf{x}_i^{(k+c)} - \mathbf{x}^*)\|$$

$$\leq \|\mathbf{H}_{\mathcal{S}_i}\| \|\mathbf{x}_i^{(k+c)} - \mathbf{x}^*\|$$

$$\leq \|\mathbf{H}_i\| \|\mathbf{e}_i^{(k+c)}\|$$

$$< \|\mathbf{H}_i\| \max\{\|\mathbf{e}_j^{(k)}\|; j = 1, \dots, N\}$$

$$\therefore \|\mathbf{e}_i^{(k+m)}\| < \gamma \max\{\|\mathbf{e}_j^{(k)}\|; j = 1, \dots, N\}$$

Now according to the selection mechanism, we have for $i = 1, \dots, N$

$$\| \mathbf{e}_i^{(k+1)} \| \leq \| \mathbf{e}_i^{(k+m)} \| < \gamma \max\{ \| \mathbf{e}_j^{(k)} \| ; j = 1, \dots, N \}$$

This implies that the sequence $\{ \max\{ \| \mathbf{e}_j^{(k+1)} \| ; j = 1, \dots, N \} ; k = 0, 1, 2, \dots \}$ is strictly monotonic decreasing and thus convergent. Note that, here, $\mathbf{e}_i^{(k+m)}$ denotes error of the i th individuals at k th-mutated generation. Also note that symbol m is used just as an indicator of mutation.

7.2.2 Adaptation Theorem

The following theorem justifies the adaptation technique for relaxation factors used in proposed hybrid evolutionary algorithms.

Theorem -2: Let $\rho(\cdot)$ be the spectral radius of matrix \mathbf{H} , α^* be the optimal relaxation factor, and let α_x and α_y are any two relaxation factors. Assume $\rho(\cdot)$ is monotonic decreasing when $\tilde{\alpha} < \tilde{\alpha}^*$ and $\rho(\cdot)$ is monotonic increasing when $\tilde{\alpha} > \tilde{\alpha}^*$. Also consider $\rho(\alpha_x) > \rho(\alpha_y)$. Then

$$(i) \quad \rho(\alpha_x^m) < \rho(\alpha_x), \text{ when } \alpha_x^m = (0.5 + \frac{1}{2}) (\alpha_x + \alpha_y)$$

$$\text{where } \alpha \in (-E_x, E_x), E_x = \frac{\tilde{\alpha}_y - \tilde{\alpha}_x}{2(\tilde{\alpha}_x + \tilde{\alpha}_y)}$$

$$(ii) \quad \rho(\alpha_y^m) < \rho(\alpha_y) \text{ when } \alpha_y^m = \alpha_y + u \text{ sign}(\alpha_y - \alpha_x), \text{ where } 0 < u \leq | \alpha^* - \alpha_y |,$$

and $\alpha^* < \alpha_y < \alpha_x$ or $\alpha_x < \alpha_y < \alpha^*$ and

$$(iii) \quad \rho(\alpha_y^m) < \rho(\alpha_y) \text{ when } \alpha_y^m = \alpha_y + u \text{ sign}(\alpha^* - \alpha_y), \text{ where } 0 < u \leq | \alpha^* - \alpha_y |.$$

Proof: We first assume that $\rho(\cdot)$ is monotonic decreasing when $\alpha < \alpha^*$ and $\rho(\cdot)$ is monotonic increasing when $\alpha > \alpha^*$. Also let α_x and α_y are any two relaxation factors and let $\rho(\alpha_x) > \rho(\alpha_y)$. Then there will be four cases:

Case-1 Both $\alpha_x, \alpha_y < \alpha^*$ (see Fig. 7.1):

Since $\rho(\cdot)$ is monotonic decreasing when $\alpha < \alpha^*$ and as assume $\rho(\alpha_x) > \rho(\alpha_y)$;

so $x < y$.

Now since $x^m = x + (0.5 + \dagger)(x + y)$ where $\dagger \in [-E_x, E_x]$, so x^m must go away from x and lies between x and y ; i.e. $x < x^m < y$ (see Fig.7.1). Now as (\cdot) is monotonic decreasing when $\cdot < *$, so $(y) < (x^m) < (x)$

i.e. $(x^m) < (x)$ (Proved the part (i))

Again since $x < y$ and $y^m = y + u \text{sign}(y - x)$ where $0 < u \leq |* - y|$ and $x < y < *$. So $\text{sign}(y - x)$ is positive and therefore y^m go away from x and y to $*$. That is $y < y^m \leq *$ (see Fig.7.1). Now since (\cdot) is monotonic decreasing when $\cdot < *$, therefore $(*) < (y^m) < (y)$

i.e. $(y^m) < (y)$. (Proved the part (ii))

Also if $y^m = y + u \text{sign}(* - y)$, where $0 < u \leq |* - y|$. Then, since in this case, $x, y < *$, so $\text{sign}(* - y)$ is positive and therefore y^m go away from y (as well as x) to $*$. That is $y < y^m \leq *$ (see Fig.7.1). Now since (\cdot) is monotonic decreasing when $\cdot < *$,

$\therefore (y^m) < (y)$ (Proved the part (iii))

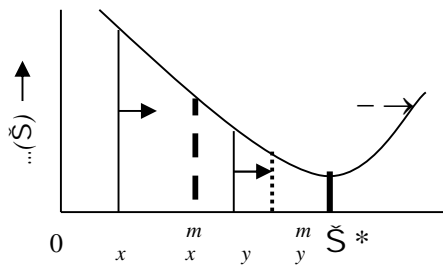


Figure 7.1: Decrease both spectral radii of \tilde{S}_x, \tilde{S}_y when $\tilde{S}_x, \tilde{S}_y < \tilde{S}^*$

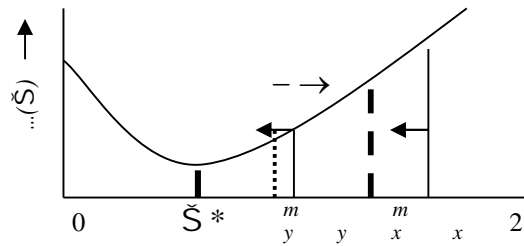


Figure 7.2: Decrease both spectral radii of \tilde{S}_x, \tilde{S}_y when $\tilde{S}_x, \tilde{S}_y > \tilde{S}^*$

Case-2 Both $x, y > *$ (see Fig. 7.2):

Since (\cdot) is monotonic increasing when $\cdot > *$ and as assuming $(x) > (y)$

so $x > y$.

Now since $\tilde{m}_x = x + (0.5 + \dagger)(x + y)$ where $\dagger \in [-E_x, E_x]$ so \tilde{m}_x must go away from x and lies between y and x ; i.e. $y < \tilde{m}_x < x$. (see **Fig. 7.2**). Now as (\cdot) is monotonic increasing when $> *$, so $(y) < (\tilde{m}_x) < (x)$ i.e. $(\tilde{m}_x) < (x)$. (Proved the part (i))

Again since $\tilde{m}_y = y + u \text{sign}(y - x)$ where $0 < u \leq |* - y|$ and $x < y < *$. So $\text{sign}(y - x)$ is negative and therefore \tilde{m}_y go away from y (as well as x and \cdot) to $*$. Therefore we have $* \leq \tilde{m}_y < y$ (see **Fig.7.2**). Now since (\cdot) is monotonic increasing when $> *$, therefore $(*) \leq (\tilde{m}_y) < (y)$ i.e. $(\tilde{m}_y) < (y)$. Hence proved the theorem of part (ii) completely.

Also if $\tilde{m}_y = y + u \text{sign}(* - y)$, where $0 < u \leq |* - y|$. Then, since in this case $x, y > *$ (and $x > y$), so $\text{sign}(* - y)$ is negative and therefore \tilde{m}_y go away from y to $*$. That is $* \leq \tilde{m}_y < y$ (see **Fig.7.2**). Now since (\cdot) is monotonic increasing when $> *$, therefore as above $(*) < (\tilde{m}_y) < (y)$ i.e. $(\tilde{m}_y) < (y)$ (Proved the part (iii))

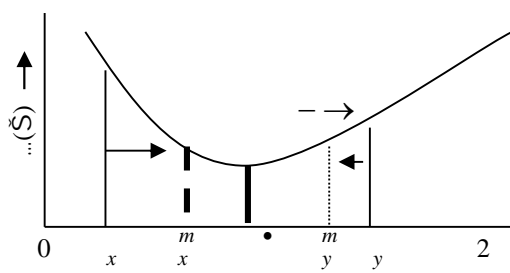


Figure 7.3: Decrease spectral radius of \tilde{S}_x and \tilde{S}_y when $\tilde{S}_x < \tilde{S}^* < \tilde{S}_y$

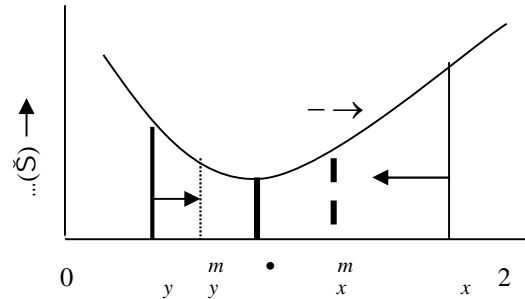


Figure 7.4: Decrease spectral radius of \tilde{S}_x and \tilde{S}_y when $\tilde{S}_x > \tilde{S}^* > \tilde{S}_y$

Case-3 If $x < * < y$ (see **Fig. 7.3**):

Given (\cdot) is monotonic decreasing when $< *$ and monotonic increasing when $> *$ and $(x) > (y)$.

Now since $\lambda_x^m = \lambda_x + (0.5 + \dagger)(\lambda_x + \lambda_y)$ where $\dagger \in [-E_x, E_x]$ so λ_x^m must go away from λ_x and lies between λ_x and λ_y ; i.e. $\lambda_x < \lambda_x^m < \lambda_y$ (see **Fig. 7.3**). Now as λ_x must be the largest spectral radius, in this case, within the range $\lambda_x < \lambda_x^m < \lambda_y$

$\therefore \rho(\lambda_x^m) < \rho(\lambda_x)$ (Proved the part (i))

Again as $\lambda_y^m = \lambda_y + u \text{sign}(\lambda^* - \lambda_y)$, where $0 < u \leq |\lambda^* - \lambda_y|$. Then, since in this case $\lambda_x < \lambda^* < \lambda_y$ and so $\text{sign}(\lambda^* - \lambda_y)$ is negative and therefore λ_y^m go away from λ_y to λ^* . That is $\lambda^* \leq \lambda_y^m < \lambda_y$ (see **Fig.7.3**). Now since $\rho(\lambda)$ is monotonic increasing when $\lambda > \lambda^*$, therefore

$$\rho(\lambda^*) \leq \rho(\lambda_y^m) < \rho(\lambda_y) \text{ i.e. } \rho(\lambda_y^m) < \rho(\lambda_y) \quad (\text{Proved the part (iii)})$$

Case-4 If $\lambda_y < \lambda^* < \lambda_x$ (see **Fig. 7.4**):

Given $\rho(\lambda)$ is monotonic decreasing when $\lambda < \lambda^*$ and monotonic increasing when $\lambda > \lambda^*$ and $\rho(\lambda_x) > \rho(\lambda_y)$.

Now since $\lambda_x^m = \lambda_x + (0.5 + \dagger)(\lambda_x + \lambda_y)$ where $\dagger \in [-E_x, E_x]$ so λ_x^m must go away from λ_x and lies between λ_y and λ_x ; i.e. $\lambda_y < \lambda_x^m < \lambda_x$ (see **Fig. 7.4**). Now as λ_x must be the largest spectral radius, in this case, within the range

$$\lambda_y < \lambda_x^m < \lambda_x$$

$$\therefore \rho(\lambda_x^m) < \rho(\lambda_x) \quad (\text{Proved the part (i)})$$

Again as $\lambda_y^m = \lambda_y + u \text{sign}(\lambda^* - \lambda_y)$, where $0 < u \leq |\lambda^* - \lambda_y|$. Then, since in this case $\lambda_y < \lambda^* < \lambda_x$ and so $\text{sign}(\lambda^* - \lambda_y)$ is positive and therefore λ_y^m go away from λ_y to λ^* . That is $\lambda^* \leq \lambda_y^m < \lambda_y$ (see **Fig.7.4**). Now since $\rho(\lambda)$ is monotonic decreasing when $\lambda < \lambda^*$, therefore $\rho(\lambda^*) \leq \rho(\lambda_y^m) < \rho(\lambda_y)$

i.e. $(\frac{m}{y}) < (\frac{m}{y})$

(Proved the part (iii))

Hence proved the theorem.

7.3 Summary

In this chapter two main theorems of hybrid evolutionary algorithms are stated and verified. The Convergence Theorem states the sufficient condition of the convergence of the algorithms; whatever be the initial (approximate) solution chosen and Adaptation Theorem justifies the adaptation of relaxation factors. Considering all the above situations, it may conclude that for a diagonally dominant coefficient matrix, any initial solution converges to true solution within some generations. Due to adaptation, it may also conclude that the comparatively worse relaxation factor $(\frac{m}{x})$ is always improved and the comparatively better relaxation factor $(\frac{m}{y})$ is also improved with a very high probability in each generation. Hence, the proposed adaptation mechanism increases the rate of convergence.



Discussions, Conclusions and Recommendations

8.1 Introduction

The main goal of this thesis was to develop and analyze some new hybrid evolutionary algorithms, which can be applied successfully to real-world problems for solving linear equations. The understanding and applications of evolutionary algorithms are, thereby, extended substantially by this work. Although a brief summary for each chapter is provided at the end of respective chapter, this chapter deals with the comprehensive discussions of the overall work in chapter basis. Details conclusions are drawn form the perspective of hybridization of evolutionary algorithm with classical numerical methods for solving linear equations. Possible future works, which can be extended from this study, are also mention in this chapter.

8.2 Discussions

Evolutionary algorithms have mostly been used to solve various optimization (either numerical or combinatorial) and evolutionary learning (supervised, reinforcement, or unsupervised) problem. In this work, some new hybrid evolutionary algorithms have been developed by incorporating evolutionary computational techniques in several classical numerical methods. The algorithms came into view as novel application of evolutionary computational techniques in equation solving by simulated evolution. Some large sets of linear equations have been solved evolutionary by using these new hybrid algorithms. In each of the experiment, a total of ten independent runs were conducted and the average results are reported in this thesis. Also all the algorithms were implemented in Borland C++ environment using Pentium IV PC (1.28GHz).

In **Chapter 2**, an attempt has been made to overview the basic constituents and properties of system of linear equations. Some well-known classical direct methods for solving linear equations are given in this chapter. Classical Iterative techniques, SR technique and some important iterative methods are also discussed in this chapter some elaborately.

In **Chapter 3**, another attempt has been made to overview the basic constituents and commitments, comparisons, properties, and merits and demerits of major evolutionary algorithms in terms of their canonical forms. Also attempt has been made to overview the hybrid evolutionary algorithm. The Gauss-Seidel Based Uniform Adaptive hybrid algorithm [He et. al (2000)] for solving linear equations also discussed in brief in this chapter.

In **Chapter 4** a new hybrid evolutionary algorithm called Jacobi Based Uniform Adaptive (JBUA) hybrid algorithm has been proposed and tested on various test problems. This algorithm can be implemented in parallel processing environment whereas GSBUA [He et. al (2000)] hybrid algorithm cannot be implemented, inherently, efficiently. The effectiveness of this algorithm is compared with other available hybrid algorithm (GSBUA) as well as classical Jacobi-SR method. This preliminary investigation has showed that this hybrid algorithm outperforms classical Jacobi-SR method and comparable with GSBUA with respective to convergence reliability as well as to solution precision.

In **Chapter 5**, another new hybrid evolutionary algorithm called Gauss-Seidel Based Time-Variant Adaptive (GSBTVA) hybrid algorithm has been proposed and tested on various test problems. This algorithm has utilized two new genetic operations, which are closely resembled to natural evolved systems. The effectiveness of this algorithm is compared with contemporary Gauss-Seidel Based Uniform Adaptive (GSBUA) hybrid evolutionary algorithm. This elementary investigation has showed that this hybrid algorithm is more efficient compare to contemporary GSBUA algorithm.

In **Chapter 6**, another new hybrid evolutionary algorithm called Jacobi Based Time-Variant Adaptive (JBTVA) has been proposed and tested by various test problems. This algorithm has also utilized two genetic operations developed in chapter 5, which are closely resembled to natural evolved systems as state earlier. The

effectiveness of this algorithm is compared with contemporary Jacobi Based Uniform Adaptive (GSBUA) hybrid evolutionary algorithm. The proposed hybrid algorithm outperforms JBUA with respect to evaluation time, solution precision and convergence reliability.

In **Chapter 7** two theorems related to the proposed hybrid algorithms are stated and proved. The convergence theorem justifies the convergence of the hybrid algorithms. And Adaptation theorem conforms that relaxation factors are adapted to reducing spectral radius, so that rate of convergence are accelerated.

8.3 Concluding Remarks

Essentially, the credibility of evolutionary algorithms relies on their ability to solve difficult, real-world problems with the minimal amount of human effort. If it cannot make the successful transition from academic exercise to physical application it will be abandoned in favor of other techniques. The overall goal of this thesis was to develop and analyze a class of hybrid evolutionary algorithms, which can be applied to real-world problems of equation solving successfully. Thus the understanding and applications of evolutionary algorithms are clearly extended by this thesis work.

Here, three hybrid evolutionary algorithms are proposed for solving system of linear equations. The significance of these works lies in the novel used of evolutionary computational techniques in the area where they had seldom been used. The proposed hybrid algorithms integrate the classical iterative (Jacobi, Gauss-Seidel) methods with evolutionary computation techniques. The recombination operator in the hybrid algorithms mixed two parents by a kind of averaging, which is a similar to the intermediate recombination often used in evolution strategies [Bäck et al. (1997)]. The mutation operator is equivalent to one iteration in the classical iterative (Jacobi, Gauss-Seidel) methods. However, the mutation is stochastic as a result of stochastic self-adaptation of the relaxation factor.

In GSBTVA and JBTVA (**Chapter 5 and Chapter 6**) time variant adaptive operators is proposed abstracted on some natural metaphor and biological observations. The time-variant based adaptation is introduced for adaptation of relaxation factors, which makes the algorithm more natural and accelerates its rate of convergence. The proposed time-variant adaptation technique of relaxation factor acts as a local fine tuner and helps to escape from the disadvantage of uniform adaptation. Numerical experiments with the test problems have shown that the proposed TVA-based hybrid algorithms perform better than the UA-based hybrid algorithms.

There are three hybrid evolutionary algorithms are proposed, two of them are developed based on classical Jacobi method and one of them developed based on classical Gauss-Seidel method. As Jacobi-based hybrid algorithms can be easily implemented in parallel computing environment efficiently [Gerald, and Wheatley (1994)]. So, if more parallel processors are available, then former (Jacobi based) algorithms may be implemented in parallel processing environment efficiently and reduce a significant amount of time compared to later (Gauss-Seidel-based) hybrid algorithms. But if there are unavailable sufficient parallel processors then Gauss-Seidel based hybrid algorithm is preferable.

There are two theorems, related to the proposed hybrid algorithms, are stated and proved. The first theorem justifies the convergence of the hybrid algorithms. This theorem is a sufficient for convergence of the algorithms. But this is not the necessary condition for convergence of the algorithms. The second theorem establishes that the relaxation factors are adapted dynamically to the near optimal relaxation factor for which the rate of convergence is increased. Therefore, these theorems justify the rapid convergence of the proposed hybrid algorithms whatever the relaxation factor are initially chosen. It may be also concluded that the time variant adaptive (TVA) based adaptation technique is better than uniform adaptive (UA) based adaptation technique.

Finally it may be concluded that all the proposed hybrid algorithms are more efficient and robust than the classical methods. The hybrid algorithms are much less sensitive on the initial values of the relaxation factor. There is no need for any preliminary experiments to estimate the relaxation factor. They are also very simple and easy to implement.

8.4 Recommendations for Future Research

There are many ways in which the work in this thesis can be extended. The following possible areas are recommended to extend the present work:

1. The number of population, all of the experiments done in this thesis, was only two. By increasing the size of the population, the impact of population size on performance of hybrid algorithms should be studied.
2. The recombination in all the experiments was simple and deterministic. The impact of different recombination parameters should be investigated. The recombination can be made stochastic by introducing random recombination. And then performance of the algorithms should be studied.
3. The self-adaptation scheme for relaxation factors should be studied further.
4. The impact of different selection mechanisms on performance of hybrid algorithms should be studied.
5. Comparison among the proposed with other classical methods should be studied.
6. Parallel hybrid algorithms should be investigated.
7. The concept of hybridization may be extended to solve non-linear equations and differential equations.

Errors

A.1 Errors in Numerical Computation

The process of solving physical problems can be roughly divided into three phases. The first consists of constructing a mathematical model for the corresponding physical problem. In most cases, this mathematical model cannot be solved analytically and hence a numerical solution is required. In which case, the second phase in the solution process usually consists of constructing an appropriately numerical model or approximate to the mathematical model. A numerical model is one where every thing in principle can be calculated using a finite number of basic arithmetic operations. The third phase of the solution process is the actual implementation and solution of the numerical model. Each of these phases involves some approximation. In the first phase, the real world problem is approximated by a mathematical model, while in the second phase the mathematical model is approximated by a numerical model and finally in the third phase the numbers are approximated. The reliability of the final result will depend on these approximations. Hence estimating the **error in each of the phases** is an integral part of the solution process. Without an error estimate or bound, the solution is of little use. Each of these three phases of solution process introduces errors in the final result [Antia (1991), Krishnamurthy and Sen (1989)]. In this section, we give a brief introduction to various sources of errors in numerical computations.

- (a) **Modeling Error:** - In the first phase, the errors could be either due to our inadequate understanding of the physical problem, or else the system is so complicated that we have to introduce some approximations in order to make the problem tractable. This error is referred to as the modeling error [Antia (1991), Krishnamurthy and Sen (1989)].

(b) **Truncation Error:** - In the second phase, error could be introduced when an essentially infinite process like summing an infinite series or evaluating an integral is approximated by a finite numerical process. This error is referred to as **truncation error**, since it is usually due to truncating an infinite process. This error of course, depends on the mathematical model. The rate, at which the truncation error tends to zero as the parameters of the method vary, is usually referred to as the order of convergence. The truncation error $e(x)$ is said to be $O(f(x))$ (this is called **big Oh notation**) as x tends to L , if

$$\lim_{x \rightarrow L} \left| \frac{e(x)}{f(x)} \right| < \infty$$

Here x is the parameter of the method, which is usually selected to be the number of points or steps n , in which case the limit is taken as $n \rightarrow \infty$ [Antia (1991), Krishnamurthy and Sen (1989)].

(c) **Round off Error:** - In the third phase, involving the actual numerical computation, there are errors due to finite precision with which the calculations can be carried out. This type of error is called round off error. It is known that at most 2^s different real numbers can be represented using a s -bit word in computer [Antia (1991), Krishnamurthy and Sen (1989), Wilkinson (1963)]. However, between any two distinct real numbers, there are infinite real numbers. Obviously, all these numbers cannot be represented in the computer. Hence, the computer's number system is necessarily "quantized" in some sense. The quantization of number system necessarily introduces some uncertainty in numerical calculation, which is the round off error [Antia (1991), Krishnamurthy and Sen (1989)]. The round off error depends on the number and the bound on the magnitude of this error depends on the exponent part of the number being represented. For floating point representation, using a base b with s -digit fraction part, the largest fraction part that can be represented in the machine is given by

$$\left| \frac{ux}{x} \right| < \frac{b}{2} b^{-s} = \hbar$$

where x is the round off error in representing x [Antia (1991), Krishnamurthy and Sen (1989)]. For a machine using binary arithmetic the largest fraction part $\hbar = 2^{-s}$. Thus, in quantum arithmetic \hbar is not a universal constant, but varies

from machine to machine. Note that h is either the smallest positive number which if added to one gives a result greater than one or it is the largest positive number which if added to one gives a result equal to one [Antia (1991), Krishnamurthy and Sen (1989)].

The effect of round off error is significant in various arithmetic operations. It may be noted that an iterative method for numerical computation does not usually converge to an accuracy better than that permitted by round off error. Hence, such methods give ready estimate of round off error. Probably the best way of estimating error in numerical calculation is to repeat the computations using a different algorithm.

- (d) **Blunder:** - In the first phase, we may overlook some basic assumption required to obtain the mathematical model, or there could be an error in deriving the mathematical equations, or the input data could be completely wrong. Similar error could occur in the second phase of the solution process also. These errors are known as blunder [Antia (1991)]. The blunders mainly occur at the programming stage.

- (e) **Bugs:** - The programming errors are usually referred to as bugs [Antia (1991)].

Definitions and Theorems

B.1 Some Definition of Matrices

Those definitions for real matrices are assembled that are needed or associated with this thesis.

(a) **Square Matrix:** - A matrix $\mathbf{A} = [a_{ik}]$ is said to be square matrix if it has equal number of rows and column i.e. $m = n$. If a square matrix has n rows then it is called a square matrix of order n or n -square matrix. The elements a_{ii} , $i = 1, 2, \dots, n$ are called main diagonal elements. Lower adjacent elements of main diagonal are called sub-diagonal elements. And upper adjacent elements of main diagonals are called super-diagonals elements [Ayres (1997, Anton and Rorres (1994))].

(b) **Singular Matrix:-** a square matrix $\mathbf{A} = [a_{ik}]$ is said to be singular if its determinant is zero [Ayres (1997)]i.e.

$$|\mathbf{A}| = 0 \tag{B.1.1}$$

(c) **Non-singular Matrix:-** a square matrix $(\mathbf{A})_{nn}$ (i.e. n -square matrix \mathbf{A}) is said to be non-singular if its determinant is not equal to zero [Ayres (1997, Anton and Rorres (1994))].i.e.

$$|\mathbf{A}| \neq 0 \tag{B.1.2}$$

(d) **Strictly Non-singular:-** a square matrix $(\mathbf{A})_{nn}$ (i.e. n -square matrix \mathbf{A}) is said to be strictly non-singular if its all leading principle diagonal minors are different from zero i.e.

$$|\mathbf{A}_{kk}| \neq 0, \quad k = 1, 2, \dots, n$$

(e) **Zero Matrix:-** Matrix \mathbf{A} , every element of which is zero, is called a zero matrix [Ayres (1997, Anton and Rorres (1994))].i.e.

$$\mathbf{A} = \mathbf{0} \tag{B.1.3}$$

(f) **Upper Triangular Matrix:** - A square matrix $\mathbf{U} = [u_{ik}]$ whose elements $u_{ik} = 0$ for $i > k$ is called upper triangular matrix. Thus

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & u_{nn} \end{bmatrix} \quad (\text{B.1.4})$$

is an upper triangular matrix [Ayres (1997, Anton and Rorres (1994))].

(g) **Strictly Upper Triangular Matrix:** - A square matrix $\mathbf{U} = [u_{ik}]$ whose elements $u_{ik} = 0$ for $i \geq k$ is called strictly upper triangular matrix. Thus

$$\mathbf{U} = \begin{bmatrix} 0 & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & 0 & u_{23} & \cdots & u_{2n} \\ 0 & 0 & 0 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \quad (\text{B.1.5})$$

is strictly upper triangular matrix [Burder and Faires (1997), Jain et. al. (1985)].

(h) **Lower Triangular Matrix:** - A square matrix $\mathbf{L} = [l_{ik}]$ whose elements $l_{ik} = 0$ for $i < k$ is called lower triangular matrix. Thus

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \cdots & \cdots & l_{nn} \end{bmatrix} \quad (\text{B.1.6})$$

is lower triangular matrix [Ayres (1997, Anton and Rorres (1994))].

(i) **Strictly Lower Triangular Matrix:** - A square matrix $\mathbf{L} = [l_{ik}]$ whose elements $l_{ik} = 0$ for $i \leq k$ is called strictly lower triangular matrix. Thus

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ l_{21} & 0 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 0 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \cdots & \cdots & 0 \end{bmatrix} \quad (\text{B.1.7})$$

is strictly lower triangular matrix [Ayres (1997, Anton and Rorres (1994))].

- (j) **Diagonal Matrix:** - A square matrix $\mathbf{A} = [a_{ik}]$ whose elements $a_{ik} = 0$ for $i \neq k$ is called diagonal matrix. Thus

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & a_{nn} \end{bmatrix} \quad (\text{B.1.8})$$

is diagonal matrix[Ayres (1997)].

- (k) **Identity Matrix:**- A square matrix $\mathbf{A} = [a_{ik}]$ whose elements $a_{ik} = 1$ for $i = k$ and $a_{ik} = 0$ for $i \neq k$ is called identity matrix and is denoted by \mathbf{I} . Thus

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \quad (\text{B.1.9})$$

is identity matrix[Ayres (1997)].

- (l) **The Transpose of a Matrix:** - The matrix of order $n \times m$ obtained by interchanging the rows and columns of a matrix $m \times n$ is called the transpose of \mathbf{A} and is denoted by \mathbf{A}' [Ayres (1997)].
- (m) **The Inverse of a Matrix:** -If \mathbf{A} and \mathbf{B} are square matrices such that $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$, then \mathbf{B} is called the inverse of \mathbf{A} and express $\mathbf{B} = \mathbf{A}^{-1}$ (\mathbf{B} equals \mathbf{A} inverse). The matrix \mathbf{B} also has \mathbf{A} as its inverse and we may write $\mathbf{A} = \mathbf{B}^{-1}$. Necessary condition of a matrix \mathbf{A} to be invertible [Ayres (1997)]. So it is nonsingular i.e. $|\mathbf{A}| \neq 0$.
- (n) **Symmetric Matrix:** - A square matrix \mathbf{A} , such that $\mathbf{A}' = \mathbf{A}$, is called symmetric. Thus a square matrix $\mathbf{A} = [a_{ik}]$ is symmetric provided $a_{ik} = a_{ki}$ for all value of k and i [Ayres (1997)].
- (o) **Band Matrix:** - An n -square matrix $\mathbf{A} = [a_{ik}]$, $i, k = 1, 2, \dots, n$ is called a band matrix, if its entries vanish outside of a band parallel to the main diagonal. Let m_l be the number of the sub-diagonals of \mathbf{A} and m_u be the number of super-diagonals of \mathbf{A} that are not all zero. Then $a_{ik} = 0$ for $i - k > m_l$ and $k - i > m_u$ where $0 \leq m_l \leq n - 2$ and $0 \leq m_u \leq n - 2$. The number $m = m_l + m_u + 1$ is the

bandwidth of \mathbf{A} . A matrix \mathbf{A} of bandwidth m can have almost $m+1$ non-zero elements in any one row. The following are the special band matrices.

- (i) Diagonal matrices for $m_l = m_u = 0$
- (ii) Bi-diagonal matrices for $m_l = 1, m_u = 0$ or $m_l = 0, m_u = 1$
- (iii) Tri-diagonal matrices for $m_l = m_u = 1$
- (iv) Five-diagonal matrices for $m_l = m_u = 2$

Matrices of this kind appear naturally in discretizations of elliptic boundary value problems [Burder and Faires (1997), Jain et. al. (1985)].

- (p) **Cyclically Tri-diagonal Matrix:** An n -square matrix $\mathbf{A} = [a_{ik}]$, $i, k = 1, 2, \dots, n$ is called cyclically tri-diagonal [Burder and Faires (1997), Jain et. al. (1985)] if

$$a_{ik} = 0 \text{ for } 1 < i - k < n - 1$$

- (q) **Sparse Matrix:** A matrix $\mathbf{A} = [a_{ik}]$ is called sparse matrix if maximum entries a_{ik} are fill up with zeroes [Burder and Faires (1997), Jain et. al. (1985)].

Matrices of this kind appear naturally, when finite-difference techniques are used to solve boundary value problems, a common application in the numerical solution of partial-differential equations and in circuit analysis.

- (r) **Diagonally Dominant Matrix:-** An n -square matrix $\mathbf{A} = [a_{ik}]$, $i, k = 1, 2, \dots, n$ is called diagonally dominant if

$$|a_{ii}| \geq \sum_{\substack{k=1 \\ k \neq i}}^n |a_{ik}| \text{ for all } i = 1, 2, \dots, n$$

and at least one index i the strictly inequality holds [Burder and Faires (1997), Jain et. al. (1985)].

- (s) **Strictly Diagonally Dominant Matrix:-** An n -square matrix $\mathbf{A} = [a_{ik}]$, is called strictly diagonally dominant [Burder and Faires (1997), Jain et. al. (1985)] if

$$|a_{ii}| > \sum_{\substack{k=1 \\ k \neq i}}^n |a_{ik}| \text{ for all } i = 1, 2, \dots, n$$

- (t) **Positive Definite and Positive Semi-definite Matrices:-** A symmetric n -square matrix $\mathbf{A} = [a_{ik}] = \mathbf{A}^t$, $i, k = 1, 2, \dots, n$ is called positive definite if

$$\mathbf{Q}(\mathbf{x}) := \mathbf{x}^t \mathbf{A} \mathbf{x} > 0 \text{ for all } \mathbf{x} \neq \mathbf{0}, \mathbf{x} \in \mathfrak{R}^n$$

And it is called positive semi-definite, if

$$Q(\mathbf{x}) := \mathbf{x}' \mathbf{A} \mathbf{x} \geq 0 \text{ for all } \mathbf{x} \in \mathfrak{R}^n$$

If \mathbf{A} is positive definite then $a_{ii} > 0$ for all i [Burder and Faires (1997), Jain et. al. (1985)].

Equivalent conditions for positive definiteness of a symmetric matrix $\mathbf{A} = \mathbf{A}^t$ if:

- (i) It is strictly diagonally dominant with positive diagonal entries $a_{ii} > 0$ for all i .
- (ii) It is diagonally dominant with positive entries $a_{ii} > 0$ for all i and $a_{ik} < 0$ for all $i \neq k$.
- (iii) It is tri-diagonal and diagonally dominant with $a_{ii} > 0$ for all i and $a_{ik} \neq 0$ for $|i - k| = 1$.

B.2 Some Definitions Related to Iterative Methods

Some important definitions related to iterative methods are discussed below:

(a) **Vector Norms:** - Let \mathfrak{R}^n be the n -dimensional real vector space and \mathbf{x} a vector in \mathfrak{R}^n . The norm function $\|\cdot\|: \mathfrak{R}^n \rightarrow \mathfrak{R}^+$ assign a nonnegative real number to $\|\mathbf{x}\|$, $\mathbf{x} \in \mathfrak{R}^n$ and fulfill the following vector norm [Burder and Faires (1997), Jain et. al. (1985)].

- (i) $\|\mathbf{x}\| > 0$ for all $\mathbf{x} \in \mathfrak{R}^n$ with $\mathbf{x} \neq \mathbf{0}$ (positive define),
- (ii) $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$,
- (iii) $\|\alpha \mathbf{x}\| = |\alpha| \|\mathbf{x}\|$ for all $\mathbf{x} \in \mathfrak{R}^n$ and $\alpha \in \mathfrak{R}$ (homogeneous),
- (iv) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathfrak{R}^n$ (triangular inequality).

Example for vector norms

- (i) $\|\mathbf{x}\|_{\infty} = \max_{1 \leq i \leq n} |x_i|$ (maximum norm),
- (ii) $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$ (absolute norm) and
- (iii) $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$ (Euclidean norm).

(b) **Matrix Norms:** - For an n -square matrix \mathbf{A} , the matrix norm $\|\mathbf{A}\|$ of \mathbf{A} is a non negative real number which satisfy the following matrix norm axioms [Engeln-Müllges and Uhlig (1996), Burder and Faires (1997)]:

- (i) $\|\mathbf{A}\| \geq 0$ for all n -square matrices (positive define).
- (ii) $\|\mathbf{A}\| = 0$ if and only if $\mathbf{A} = \mathbf{0}$,
- (iii) $\|\alpha \mathbf{A}\| = |\alpha| \|\mathbf{A}\|$ for all matrices and $\alpha \in \mathfrak{R}$ (homogeneous),
- (v) $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$ for all n -square matrices \mathbf{A}, \mathbf{B} (triangular inequality), and
- (iv) $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$ for all n -square matrices \mathbf{A}, \mathbf{B} (sub-multiplication).

Examples of matrix norms are:

- (i) $\|\mathbf{A}\|_{\infty} = \max_{1 \leq i \leq n} \sum_{k=1}^n |a_{ik}|$ (row sum norm),
- (ii) $\|\mathbf{A}\|_1 = \max_{1 \leq k \leq n} \sum_{i=1}^n |a_{ik}|$ (column sum norm),
- (iii) $\|\mathbf{A}\|_2 = \sqrt{\sum_{i,k=1}^n |a_{ik}|^2}$ (Frobenious/ Spectral / Hilbert norm),

A matrix norm is compatible with the vector norm if for every matrix \mathbf{A} and vector \mathbf{x} the inequality

$$\|\mathbf{Ax}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \tag{B.2.1}$$

holds.

If $\|\cdot\|$ is a vector norm on \mathfrak{R}^n , then

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\| \tag{B.2.2}$$

is called the natural or induced , matrix norm associate with the vector norm [Burder and Faires (1997), Jain et. al. (1985)].

(c) **Conditional Number:** -The conditional number is normally defined as the product of two matrix norms [Burder and Faires (1997), Jain et. al. (1985)]:

$$Condition(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \tag{B.2.3}$$

(d) **Ill-conditioned Systems:** - A matrix \mathbf{A} is called ill-conditioned if there exists a vector \mathbf{b} for which small perturbations in the coefficients of \mathbf{A} or \mathbf{b} will produce large changes in $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. The system is ill conditioned when \mathbf{A} is ill conditioned [Mathews (2001), Engeln-Müllges and Uhlig (1996)]. When the condition number for the coefficient matrix \mathbf{A} , of a systems of linear equations, is very large then the

systems are ill conditioned. In ill-conditioned systems, numerical methods for computing an approximate solution are prone, due to round off, to have more errors. For well-conditioned systems of linear equations, the condition number for the coefficient matrix \mathbf{A} is small.

- (e) **Eigen Values and Eigen Vectors:** - For a given matrix $\mathbf{A} = [a_{ik}]$ the eigen value problems consist of finding non-zero vector \mathbf{x} so that \mathbf{Ax} is parallel to the vector \mathbf{x} . Such a vector is called eigen vector of \mathbf{A} . It satisfy the eigen value-eigen vector equation of a scalar λ , called eigen value:

$$\mathbf{Ax} = \lambda \mathbf{x} \quad (\text{B.2.4})$$

The equation (2.4.4) has nontrivial solution if matrix $\mathbf{A} - \lambda \mathbf{I}$ is singular, i.e. if

$$P(\lambda) = \det(\mathbf{A} - \lambda \mathbf{I}) = 0 \quad (\text{B.2.5})$$

The equation $P(\lambda) = 0$ is called the characteristic equation of the matrix \mathbf{A} .

- (f) **Spectral Radius:** - The spectral radius $\rho(\mathbf{A})$ of a matrix \mathbf{A} is defined by $\rho(\mathbf{A}) = \max |\lambda|$ [Burder and Faires (1997)], where λ is an eigen value of \mathbf{A} if \mathbf{A} is a square matrix, then

(i) $\|\mathbf{A}\|_2 = [\mathbf{A}^t \mathbf{A}]^{1/2}$,

(ii) $\rho(\mathbf{A}) \leq \|\mathbf{A}\|$, for any natural norm $\|\cdot\|$

- (g) **Definition of Convergence:** - A sequence $\{\mathbf{x}^{(k)}; k = 1, 2, \dots\}$ of vectors in \mathfrak{R}^n is said to be converge to \mathbf{x} with respect to norm $\|\cdot\|$ if, given any $\epsilon > 0$, there exist an integer $N(\epsilon)$ such that

$$\|\mathbf{x}^{(k)} - \mathbf{x}\| < \epsilon \quad \text{for all } k \geq N(\epsilon), \quad (\text{B.2.6})$$

- (h) **Theorem of Convergence:** - *The sequence of vectors $\{\mathbf{x}^{(k)}\}$ converges to \mathbf{x} with respect to norm $\|\cdot\|$ if and only if $\lim_{k \rightarrow \infty} x_i^{(k)} = x_i$ for each $i = 1, 2, \dots, n$.*

- (i) **Scaling:** - Scaling is the operation of adjusting the coefficients of a set of equations so that they are all of the same order of magnitude. It has utility in minimizing roundoff error for those cases where some of the equations in a system have much larger coefficients than others coefficients. Such situations are frequently encounter in engineering practices when widely different units are used in the development of simultaneous equations [Gerald and Wheatley (1994), Chapra. and Canale (1990)].

B.3 Some Theorems Related to Iterative Methods

Some important theorems related to iterative methods are discussed below:

- (a) **Sufficient Condition for Convergence of Iterative Process:** - The process of iteration will converge if in each equation of the system the absolute value of the largest coefficient is greater than the sum of the absolute values of all the remaining coefficients in that equation. Mathematically (if the coefficient matrix \mathbf{A} is rearranged so that diagonal entries are the largest in each row) then the sufficient condition may be express as

$$|a_{ii}| > \sum_{k=1}^n |a_{ik}|, \quad \text{for all } i = 1, 2, \dots, n \quad (\text{B.3.1})$$

i.e. the systems are diagonally dominant [Scarborough (1966), stoer and Bulirsch (1992), Gourdin and Boumahrat (1996), Burder and Faires (1997)]. Fortunately many engineering problems of practical importance fulfill this requirement.

- (b) **Theorem:** - For any $\mathbf{x}^{(0)} \in \mathfrak{R}^n$ the sequence $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ defined by $\mathbf{x}^{(k)} = \mathbf{H}\mathbf{x}^{(k-1)} + \mathbf{V}$, for each $k \geq 1$ converge to a the unique solution of $\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{V}$ if and only if $\rho(\mathbf{H}) < 1$ [Scarborough (1966), Jain et al. (1985), Gourdin and Boumahrat (1996), Burder and Faires (1997)].

- (c) **Theorem:** - If \mathbf{A} is strictly diagonally dominant, then for any choice of $\mathbf{x}^{(0)} \in \mathfrak{R}^n$, both the Jacobi and Gauss-Seidel methods give sequences $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ that converge to the unique solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ [Scarborough (1966), Jain et al. (1985), Gourdin and Boumahrat (1996), Burder and Faires (1997)].

- (d) **Theorem (Stein-Rosenberg):**- If $a_{ik} \leq 0$ for each $i \neq k$ and $a_{ii} > 0$ for each $i = 1, 2, \dots, n$ then one and only one of the following statement holds [Burder and Faires (1997)]:

(i) $0 \leq \rho(\mathbf{H}_g) < \rho(\mathbf{H}_j) < 1$

(ii) $1 < \rho(\mathbf{H}_j) < \rho(\mathbf{H}_g)$

(iii) $\rho(\mathbf{H}_j) = \rho(\mathbf{H}_g) = 0$

(iv) $\rho(\mathbf{H}_j) = \rho(\mathbf{H}_g) = 1$

where $\rho(\mathbf{H}_j)$ and $\rho(\mathbf{H}_g)$ are denoted as spectral radius measured from Jacobi method and Gauss-Seidel method respectively.

- (e) **Residual Vector/Correction Vector:** - Suppose $\tilde{\mathbf{x}} \in \mathcal{R}^n$ is an approximation to the solution of the linear system defined by $\mathbf{Ax} = \mathbf{b}$. The residual vector/correction vector for $\tilde{\mathbf{x}}$ with respect to this system is $\mathbf{z} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$ [Engeln-Müllges and Uhlig (1996), Jain et al. (1985), Burder and Faires (1997) Cheney and Kincaid (1999)].
- (f) **Theorem (Kahan):** - If $a_{ii} \neq 0$ for each $i = 1, 2, \dots, n$ then $(\mathbf{H}_S) \geq |\tilde{S} - 1|$. This implies that the SR technique can converge only if $0 < \omega < 2$ [Engeln-Müllges and Uhlig (1996), Gourdin and Boumahrat (1996), Burder and Faires (1997)].
- (g) **Theorem:** - A sufficient condition for convergence of the successive relaxation technique when \mathbf{A} is strictly diagonally dominant is $0 < \omega < 1$ [Gourdin and Boumahrat (1996)].
- (h) **Theorem:** - For symmetric and positive-defined matrix \mathbf{A} the relaxation technique converges if and only if $0 < \omega < 2$ [Gourdin and Boumahrat (1996)].
- (i) **Theorem: - (Ostrowski-Reich):** - If \mathbf{A} is a positive definite matrix and $0 < \omega < 2$, then SR technique converge in the both Jacobi and Gauss-Seidel methods for any choice of initial approximation vector $\mathbf{x}^{(0)}$ [Burder and Faires (1997)].
- (j) **Theorem:** - No eigen value of a matrix \mathbf{A} exceeds the norm of a matrix, i.e. $\|\mathbf{A}\| \geq \lambda(\mathbf{A})$ [Jain et al. (1985)].
- (k) **Theorem:** - A necessary and sufficient condition for convergence an iterative method is that the eigen values of the iteration matrix satisfy
- $$|\lambda_i(\mathbf{H})| < 1, \quad i = 1, 2, \dots, n \quad [\text{Jain et al. (1985)}].$$

C.1 Some Algorithms of Classical Iterative methods

C.1.1 Algorithm of Jacobi Method

INPUT: Given $\mathbf{A} = [a_{ik}]$, $\mathbf{b} = [b_i]$; set initial approximation solution $\mathbf{x}^{(0)} = [x_i]$; set threshold error ϵ , maximum number of iteration T .

Step 1: set $k=1$, // Here k is denoted as Iteration counter

Step 2: While ($k \leq T$) do steps 3-6

Step 3: For $i = 1, 2, \dots, n$

$$\text{set } x'_i = \frac{1}{a_{ii}} \left(- \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij} x_j + b_i \right)$$

Step 4: if $\|\mathbf{x}^{(k)}\| < \epsilon$

then OUTPUT: solution $\mathbf{x} = \mathbf{x}'$ (procedure completed successfully)

STOP.

else

Step 5: set $k=k+1$

Step 6: For $i = 1, 2, \dots, n$

$$\text{set } x_i = x'_i$$

Step 7: OUTPUT (maximum number of iteration exceeded; Procedure completed unsuccessfully)

STOP.

C.1.2 Algorithm of Gauss-Seidel Method

INPUT: Given $\mathbf{A} = [a_{ik}]$, $\mathbf{b} = [b_i]$; set initial approximation solution $\mathbf{x}^{(0)} = [x_i]$;

Set threshold error ϵ , maximum number of iteration T .

Step 1: set $k=1$, // Here k is denoted as Iteration counter

Step 2: While ($k \leq T$) do steps 3-6

Step 3: For $i = 1, 2, \dots, n$

$$\text{set } x'_i = \frac{1}{a_{ii}} \left(- \sum_{j=1}^{i-1} a_{ij} x'_j - \sum_{j=i+1}^n a_{ij} x_j + b_i \right)$$

Step 4: if $\|\mathbf{x}^{(k)}\| < \epsilon$

then OUTPUT: solution $\mathbf{x} = \mathbf{x}'$ (procedure completed successfully)

STOP.

else

Step 5: Set $k=k+1$

Step 6: For $i = 1, 2, \dots, n$

$$\text{set } x_i = x'_i$$

Step 7: OUTPUT (maximum number of iteration exceeded; Procedure completed unsuccessfully)

STOP.

C.1.3 Algorithm of Jacobi based SR Method

INPUT: Given $\mathbf{A} = [a_{ik}]$, $\mathbf{b} = [b_i]$; set initial approximation solution $\mathbf{x}^{(0)} = [x_i]$,

relaxation factor ω ; Set threshold error ϵ , maximum number of iteration T .

Step 1: set $k=1$, // Here k is denoted as Iteration counter

Step 2: While ($k \leq T$) do steps 3-6

Step 3: For $i = 1, 2, \dots, n$

$$\text{Set } x'_i = x_i + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^n a_{ij} x_j \right)$$

Step 4: if $\|\mathbf{x}^{(k)}\| < \epsilon$

then OUTPUT: solution $\mathbf{x} = \mathbf{x}'$ (procedure completed successfully)

STOP.

else

Step 5: Set $k=k+1$

Step 6: For $i = 1, 2, \dots, n$

set $x_i = x'_i$

Step 7: OUTPUT (maximum number of iteration exceeded; Procedure completed unsuccessfully)

STOP.

C.1.4 Algorithm of Gauss-Seidel based SR Method

INPUT: Given $\mathbf{A} = [a_{ik}]$, $\mathbf{b} = [b_i]$; set initial approximation solution $\mathbf{x}^{(0)} = [x_i]$, relaxation factor \check{S} ; Set threshold error γ , maximum number of iteration T .

Step 1: set $k=1$, // Here k is denoted as Iteration counter

Step 2: While ($k \leq T$) do steps 3-6

Step 3: For $i = 1, 2, \dots, n$

$$\text{set } x'_i = (1 - \check{S})x_i + \frac{\check{S}}{a_{ii}} \left(- \sum_{j=1}^{i-1} a_{ij}x'_j - \sum_{j=i+1}^n a_{ij}x_j + b_i \right)$$

Step 4: if $\|\mathbf{x}^{(k)}\| < \gamma$

then OUTPUT: solution $\mathbf{x} = \mathbf{x}'$ (procedure completed successfully)

STOP.

else

Step 5: Set $k=k+1$

Step 6: For $i = 1, 2, \dots, n$

set $x_i = x'_i$

Step 7: OUTPUT (maximum number of iteration exceeded; Procedure completed unsuccessfully)

STOP.

C.2 Some Evolutionary Algorithms

C.2.1 A Pseudo-code Structure of Evolutionary Algorithms

Algorithm_EA()

begin

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

Initialize population: $\Pi(0) = \{ \pi_1(0), \pi_2(0), \dots, \pi_n(0) \}$;

/* Here $\pi_i(0) \Rightarrow i$ -th individual at zeroth generation */

Evaluate population: $w(\Pi(0)) = \{ w(\pi_1(0)), w(\pi_2(0)), \dots, w(\pi_n(0)) \}$;

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

begin

Select individuals for reproduction

Apply operators:

Crossover: $\Pi^c(t) = '(\Pi(t))$; /* if exist */

/* Superscript c denotes Crossover */

Mutation: $\Pi^m(t) = '(\Pi^c(t))$; /* if exist */

/* Superscript m denotes Mutation */

Evaluate newborn offspring:

$w(\Pi^m(t)) = \{ w(\pi_1^m(t)), w(\pi_2^m(t)), \dots, w(\pi_n^m(t)) \}$

Selection: $\Pi(t++) = g(\Pi^m(t) \cup Q)$; /* Here, $Q \in \{0, \Pi(t)\}$ */

$t \leftarrow t + 1$; /* Increase the generation counter */

end

end

Figure C.1: A pseudo-code structure of evolutionary algorithms

C.2.2 A Pseudo-code Structure of Hybrid Evolutionary Algorithms

Algorithm_HYBRID_EA()

begin

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

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

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

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 denotes Crossover */

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

/* Superscript c denotes Crossover & \mathbf{H}_i is iteration matrix */

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 : $\tilde{S}_x = f_x(\tilde{S}_x, \tilde{S}_y, p_x)$ & $\tilde{S}_y = f_y(\tilde{S}_x, \tilde{S}_y, p_y)$

/* p_x and p_y are adaptive probability functions */

Selection and reproduced: $\mathbf{X}^{(k+1)} = \mathbf{g}(\mathbf{X}^{(k+m)})$

$k \leftarrow k + 1$; /* Increase the generation counter */

end

end

Figure C.2: A pseudo-code structure of hybrid evolutionary algorithms

1. Jamali, A R M J U, M. M. A. Hashem and M. B. Rahman (accepted), “*Solving Linear Equations Using a Jacobi Based Time-Variant Adaptive Hybrid Evolutionary Algorithm*”, The 7th International Conference on Computer and Information Technology (ICCIT-04), Bangladesh, 26-28 Dec. 2004.
2. Jamali, A R M J U, M. M. A. Hashem and M. B. Rahman (accepted), “*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 (2004).
3. 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.

References

1. Antia, H. M., (1991), “*Numerical Methods for Scientist and Engineers*”, Tata McGraw-Hill, New Delhi, pp. 01 – 109.
2. Anton, H. and C.Rorres (1994), “*Elementary linear Algebra, Application Version (7th edition.)*”, John Weley & Sons, New York, pp. 01 – 375.
3. Ayres, J. F (1997), “*Schaum’s Outline of Theory and Problems of Matrices*”, McGraw-Hill, London, pp. 01 – 90.
4. 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.
5. Bäck, T. (1997), “*Self-adaptation, in Handbook of Evolutionary Computation*”, Oxford University Press.
6. 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.
7. 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.
8. 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.
9. 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.

10. 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.
11. 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.
12. Burder R. L., and J. D. Faires (1997), “*Numerical Analysis (6th edition)*”, Brooks/Cole – Thomson Learning, USA, pp. 250-472.
13. Carre, B.A. (1961), “*The Determination of the Optimum Accelerating Factor for Successive Over-Relaxation*”, The Computer Journal, Vol 4, pp. 73-78.
14. Chapra, S. C. and Canale, R. P. (1990), “*Numerical Method for Engineers (2nd edition)*”, McGraw-Hill, New York.
15. Chellapilla, K (1998)], “*Combining Mutation Operations in Evolutionary Programming*”, IEEE Trans. On Evolutionary Computation, 2(3), pp. 91-96.
16. Chellapilla, K. and D. B. Fogel (1997), “*Two New Mutation Operators for Enhanced Search and Optimization in Evolutionary Programming*”, In: *Applications of Soft computing* (B. Bosacchi, L. C. Bezdek and D. B. Fogel, Eds), Procs. of the SPIE, Vol. 3165, pp. 260-269.
17. Chellapilla, K., H. Birro and S. S. Rao (1998), “*Effectiveness of Evolutionary programming*”, In: 3rd Annual conference on Genetic Programming (GP’98), July 22-25, Univ. of Wisconsin, Madison.
18. Cheney, W., and D. Kincaid (1999), “*Numerical Mathematics and computing (4th edition)*”, Brooks/Cole – Thomson Learning, USA pp. 240-316.
19. Dongarra, J. J., Bunch, J. R., Moler, C.B. and Stewart, G.W. (1997), “*LINPACK User’s Guide*”, SIAM, Philadelphia.
20. Engeln-Müllges, G. E., and F. Uhlig (1996), “*Numerical Algorithms with C*”, Springer-Verlag, Heidelberg, pp. 59 – 142.
21. Faddeev, D. K. and V. N. Faddeeva (1963), “*Computational Methods of Linear Algebra*”. (Trans. R. C. Williams & W. H. Freeman), San Francisco.
22. Fogel, D. B (1995), “*Evolutionary Computation: Towards a New Philosophy of Machine Inelegance*”, IEEE Press, Piscataway, N J.

23. Fogel, D. B. and W. J. Atmar (1990), “*Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Process Using Linear Systems*”, “Biol” Cybernet, 63 (2), pp.111-114.
24. Fogel, L. J., A. J. Owens and M. J. Walsh (1966), “*Artificial Intelligence through Simulated Evolution*”, Wiley, New York..
25. Forsythe, G. E. and C. B. Moler (1967), “*Computer Solution of Linear Algebraic Systems*”, Prentice-Hall, Englewood Cliffs, New Jersey.
26. Gantmacher, F. R. (1990), “*The Theory of Matrices (2nd edition.)*”, Vol. 2, Chelsea, New York.
27. Gerald, C. F., and P. O.Wheatley (1994), “*Applied Numerical Analysis (5th edition.)*”, Addison-Wesley, New York. pp. 102-209.
28. Gourdin, A. and M. Boumahrat (1996), “*Applied Numerical Methods*”, Prentice Hall of India, New Delhi, pp. 212-232.
29. Gregory, R. T. and D. L. Karney (1969), “*A collection of Matrices for Testing Computational Algorithms*”, Wiley- Inter-science, New York.
30. Hagaman, L. A. and D. M. Young (1981)’ “*Applied Iterative, Methods*”, Academic press, New York.
31. 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.
32. He, J., J. Xu, and X. Yao (2000), “*Solving Equations by Hybrid Evolutionary Computation Techniques*”, Transactions on Evolutionary Computation, 4(3), pp. 295-304.
33. Holland, J. H. (1962), “*Outline for a Logical Theory of Adaptive Systems*”, Journal of the Association for Computing Machinery, 3, pp. 297-314.
34. Jain, M. K., S. R. K. Iyengar and R. K. Jain. (1985), “*Numerical Methods for Scientific and Engineering Computation (2nd edition)*”, Wiley Eastern, India.
35. 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..

36. Kim, J. H. and H. Myung (1997), “*Evolutionary Programming Techniques for Constrained Optimization Problems*”, IEEE Trans. on evolutionary Computation, 1(2), pp. 129 – 140.
37. Koza, J. R. (1994), “*Genetic Programming on the Programming of Computers by Means of Natural Evolution*”, MIT Press, Massachusetts.
38. Kreyszig, E. (1993), “*Advanced Engineering Mathematics (7th edition)*”, John Wiley & Sons, New York, pp. 326 359.
39. Krishnamurthy, E. V. and S. K. Sen (1989), “*Numerical Algorithms computations in Science and Engineering*”, Affiliated East-West Press New Delhi, pp. 157-259.
40. Lang, S. (1987), “*Linear Algebra. (3rd edition)*”, Springer, New York..
41. Mathews, J. H. (2001), “*Numerical Methods for Mathematics, Science, and Engineering, (2nd edition. & 6th reprint)*”, Prentice-Hall of India, New Delhi.
42. Michalewicz, Z.(1994), *A Hierarchy of Evolution Programs*, “*An Experimental Study*, Evolutionary Computation”, 1(1), pp. 51 – 76.
43. Michalewicz, Z. (1994a), “*Evolutionary Computation Techniques for Nonlinear Programming Problems*”, International Trans. On Operation Research, 192), pp. 223- 240. ([http:// www.coe.uncc.edu/~zbyszek/papers.html](http://www.coe.uncc.edu/~zbyszek/papers.html)).
44. Michalewicz, Z. (1996), “*Genetic Algorithms + Data Structure = Evolution Programs, (3rd Rev., and extended edition)*”, Springer-Verlag, Berlin..
45. 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.
46. Pissanetzky , S. (1984), “*Sparse Matrix Technology*”, Academic press, London.
47. Press, W. H., B. P. Flannery, S. A. Teukoisky and W. T. Vetterling (1988), “*Numerical Recipes in C: The Art of Scientific Computing*”, Cambridge University Press Cambridge.

48. Rechenberg, I. (1973), "*Evolutions Strategies: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution*", Frommann-Holzbock Verlag, Stuttgart.
49. 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.
50. Salomon, R. (1998), "*Evolutionary Algorithms and Gradient Search: Similarities and Differences*", IEEE Trans. on Evolutionary Computation, **2** (2), pp. 45 – 55.
51. Salomon, R and J. L. V. Hemmen (1996), "*Accelerating Back Propagation Through Dynamic Self-adaptation*", Neural Networks, 9(4), pp. 589-601.
52. Scarborough, J. B. (1966), "*Numerical Mathematical Analysis (6th edition)*", Oxford &IBH, New Delhi.
53. Schoenauer, M and Z. Michalewicz. (1997), "*Evolutionary Computation*", Control and Cybernetics 26(3), pp. 303-388.
54. Schwefel, H.-P. (1995), "*Evolution and Optimum Seeking*", New York, Wiley.
55. 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.
56. Smith, J. and T. C. Fogarty (1996)), "*Self-adaptation Mutation Rates in a Steady State Genetic Algorithm*", In Proceedings of 1996 IEEE International Conference on Evolutionary Computation. Piscataway, NJ. IEEE Press, pp. 318-323.
57. Stoer, J. and R. Bulirsch. (1991/92), "*Introduction to Numerical Analysis (2nd edition)*", Springer, New York.
58. Tewarson, R. P. (1973), "*Sparse Matrices*", Academic Press, New York.
59. Varga, R. S. (1962), "*Matrix Iterative Analysis*", Prentice-Hall, Englewood Cliffs, New Jersey.
60. Wilkinson, J. H. (1963), "*Rounding Errors in Algebraic Processes*", HMSO, London.

61. Yao, X., and Y. Liu (1997), "*Fast Evolutionary Strategies*", Control and Cybernetics, Special Issue on Evolutionary Computation, 26(3), pp.467 – 497.
62. Young, D. (1954), "*Iterative Method for Partial Difference Equations of Elliptic Type*", Trans. American Math. Soc, Vol 7(6), pp. 92-111.
63. Young, D. (1971), "*Iterative Solution of Large Linear System*", Academic Press, Now York.
64. Young, D. M. and T. G. Frank (1963), "*A Survey of Computer Methods for Solving Elliptic and Parabolic Partial Differential Equation*", ICC Bulletin, Vol. 2, pp. 1-61.
65. Yuret, D. (1994), "*From Genetic Algorithm to Efficient Optimization*", MIT, A.I. Technical Report No. 1569.