International Journal of Mathematics and Statistics Studies

Vol.2, No.1, pp.55-69, March 2014

Published by European Centre for Research Training and Development (www.ea-journals.org)

SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS USING ACCELERATED GENETIC ALGORITHM

Dr. Eman A. Hussain
Al-Mustansiriyah University, College of Science , Department of Mathematics.
E-mail address: dr_emansultan@yahoo.com

Yaseen M. Alrajhi
Al-Mustansiriyah University, College of Science , Department of Mathematics.
E-mail address: yargmmn@yahoo.com

ABSTRACT: *This project introduced an accelerated method of Genetic Algorithms (GAs) to solve Partial differential equations. This new method for solving partial differential equations, based on grammatical evolution is presented. The method forms generations of trial solutions expressed in an analytical closed form and developed by inserting the boundary conditions, part of exact solution or exact solution as a vectors of trial solutions in the population of the problem. Several examples are worked out and in most cases the exact solution is recovered. When the solution cannot be expressed in a closed analytical form then our method produces an approximation with a controlled level of accuracy. We report results on several problems to illustrate the potential of this approach.*

KEYWORDS: Partial Differential Equations**,** Genetic Algorithms (Gas)**,**Nonlinear .

INTRODUCTION

Applying mathematics to a problem of the real world mostly means, at first modeling the problem mathematically, maybe with hard restrictions, idealizations ,or simplifications,[1] then solving the mathematical problem, and finally drawing conclusions about the real problem based on the solutions of the mathematical problem.Since about 60 years, a shift of paradigms has taken place in some sense, the opposite way has come into fashion. The point is that the world has done well even in times when nothing about mathematical modeling was known. More specifically, there is an enormous number of highly sophisticated processes and mechanisms in our world which have always attracted the interest of researchers due to their admirable perfection. To imitate such principles mathematically and to use them for solving a broader class of problems has turned out to be extremely helpful in various disciplines. Just briefly. The class of such methods will be the main object of study throughout this whole project Genetic Algorithms (GAs).

Generally speaking, genetic algorithms are simulations of evolution, of what kind ever. In most cases, however, genetic algorithms are nothing else than probabilistic optimization methods which are based on the principles of evolution. This idea appears first in 1967 in J. D. Bagley's thesis [2]. The theory and applicability was then strongly influenced by J. H. Holland, who can be considered as the pioneer of genetic algorithms [6, 7]. Since then, this field has witnessed a tremendous development. The purpose of this project is to give a comprehensive overview of this class of methods and their applications in optimization, program induction, and machine learning. Genetic Algorithm uses this idea of selection of the best fit to find

optimal solutions to problems. We will examine how to use genetic Algorithm to solve Partial differential equations (PDE's) and also how to approximate solutions of an PDE's with no exact solution.
In the following sections we give in Section 2 description of the The Evolutionary Computation in Section 3 The Basics of Genetic Algorithms in Section 4 Grammatical Evolution in Section 5 Technical of the accelerated method in section 6 Applications of the Algorithm in section 7 Compare the method in section 8 Convergence and in section 9 conclusion

Description of the Evolutionary Computation

Work on what is nowadays called evolutionary computation started in the sixties of the 20th century in the United States and Germany. There have been two basic approaches in computer science that copy evolutionary mechanisms: evolution strategies (ES) and genetic algorithms (GA). Genetic algorithms go back to Holland [6], an American computer scientist and psychologist who developed his theory not only under the aspect of solving optimization problems but also to study self - adaptiveness in biological processes. Essentially, this is the reason why genetic algorithms are much closer to the biological model than evolution strategies. The theoretical foundations of evolution strategies were formed by Rechenberg and Schwefel (see for example [8] or [9]), whose primary goal was optimization. Although these two concepts have many aspects in common, they developed almost independently from each other in the USA (where GAs were eve loped) and Germany (where research was done on ES). Both attempts work with a population model whereby the genetic information of each individual of a population is in general different.

Among other things this genotype includes a parameter vector which contains all necessary information about the properties of a certain individual. Before the intrinsic evolutionary process takes place, the population is initialized arbitrarily; evolution, i.e., replacement of the old generation by a new generation, proceeds until a certain termination criterion is fulfilled. The major difference between evolution strategies and genetic algorithms lies in the representation of the genotype and in the way the operators are used (which are mutation, selection, and eventually recombination). In contrast to As, where the main role of the mutation operator is simply to avoid stagnation, mutation is the primary operator of evolution strategies. Genetic programming (GP), an extension of the genetic algorithm, is a domain-independent, biologically inspired method that is able to create computer programs from a high-level problem statement.

In fact, virtually all problems in artificial intelligence, machine learning, adaptive systems, and automated learning can be recast as a search for a computer program, genetic programming provides a way to search for a computer program in the space of computer programs ,[10]. Similar to GAs, GP works by imitating aspects of natural evolution, but whereas GAs are intended to find arrays of characters or numbers, the goal of a GP process is to search for computer programs (or, for example, formulas) solving the optimization problem at hand. As in every evolutionary process, new individuals (in GP's case, new programs) are created. They are tested, and the fitter ones in the population succeed in creating children of their own whereas unfit ones tend to disappear from the population.

The Basics of Genetic Algorithms

Genetic Algorithms are inspired by Charles Darwin's principle of natural selection[1,4]. The basic algorithm starts with a 'population' of random parameter vectors or 'individuals' and uses these to evolve a particular individual that solves or partially solves some optimization problem. 'Evolution' is implemented by using an artificial selection mechanism at each generation and using the selected individuals to produce the next generation. New individuals can be produced from old individuals by a variety of means including random perturbation (asexual reproduction or 'mutation') or by combining parts from two or more individuals (parents) to make a new individual (hermaphroditic reproduction or 'crossover').

The process of constructing new individuals from the previous generation is called 'reproduction', and mutation and crossover are called reproduction 'operators'. An individual's actual encoding (i.e. the integers and reals , or just ones and zeros) is sometimes called its 'genetic' encoding and the form of the encoding is often described s consisting of 'chromosomes' or sometimes, 'genes'. The term 'representation' in GAs describes a higher level concept than the actual encoding of ones and zeros ,the 'representation' is the meaning attached to each of the parameter values in the parameter vector. The only domain-specific information available to the algorithm is implicitly built into the 'fitness function'. The representation only acquires the meaning we want if we build that assumption into the fitness function. The fitness function takes a single parameter vector (an 'individual') as its argument and returns (usually) a single 'fitness value', so called because it measures the 'fitness' of that individual in terms of its ability to solve the optimization problem at hand.

Selection occurs solely on the basis of these fitness values. This allows the neat partitioning of the genetic algorithm into three separate components:
1. The (initially random) parameter vectors
2. The (problem specific) fitness function
3. The evolution mechanism (consisting of selection and reproduction)

In GAs the evolution mechanism is often generalized to such an extent that the only change required to apply the algorithm to a new problem is to define a new fitness function. A number of crossover operators may be available, but many of these are identical in principle, but for a change in representation1. A common choice is 1-point crossover which chooses a point randomly along the parameter vector and concatenates that part of the vector to the left of this point in one parent's encoding with that part to the right in the other parent's encoding. This can be generalized to *n* point crossover by choosing *n* points and taking each section from alternate parents. It is important to simulate a sufficiently large population (~10^2-10^3), and to allow evolution to proceed for enough generations (~50-2000), however, the success of a genetic algorithm depends mainly on how appropriate the reproduction operators are for the particular representation, and how informative the fitness function is.

An 'appropriate' reproduction operator is one which produces new individuals in such a way that there is a reasonable probability that the fitness of the new individual will be significantly higher than that of its parents. An 'informative' fitness function is one that gives monotonically increasing fitness values to individuals as they get closer to solving the problem under consideration. The combination of these two properties opens the way to the successful evolution of a solution to the optimization problem at hand. Much of the theoretical work in the field of GAs is into finding the most appropriate reproduction operators and representations.

The range of behaviors generable by a GA (or GP) is limited explicitly by the fitness function and the meaning of the parameters it accepts. In GAs the meaning of the parameters is usually quite limited and the number of parameters fixed. In Genetic Programming, the representation of each individual is actually a program in some limited programming language which must be executed to determine the fitness of the individual. In this case the fitness function contains an interpreter that must decode the instructions of each individual to determine how well that individual solves the given problem.
Concerning its internal functioning, a genetic algorithm is an iterative procedure which usually operates on a population of constant size and is basically executed in the following way: [2]

An initial population of individuals (also called "solution candidates" or "chromosomes") is generated randomly or heuristically. During each iteration step, also called a "generation, the individuals of the current population are evaluated and assigned a certain fitness value. In order to form a new population, individuals are first selected (usually with a probability proportional to their relative fitness values), and then produce offspring candidates which in turn form the next generation of parents. This ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population. For producing new solution candidates genetic algorithms use two operators, namely crossover and mutation: [4]

- Crossover is the primary genetic operator : It takes two individuals, called parents, and produces one or two new individuals, called offspring, by combining parts of the parents. In its simplest form, the operator works by swapping (exchanging) substrings before and after a randomly selected crossover point.
- The second genetic operator, mutation, is essentially an arbitrary modification which helps to prevent premature convergence by randomly sampling new points in the search space. In the case of bit strings, mutation is applied by simply flipping bits randomly in a string with a certain probability called mutation rate.

Initial steps of GA's:

Step 1 : Represent the problem variable domain as a chromosome of a fixed length, choose the size of a chromosome population N, the crossover probability p_c and the mutation probability p_m.
Step 2 : Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.
Step 3 : Randomly generate an initial population of chromosomes of size N ,$x_1,\ldots,x_N$
Step 4 : Calculate the fitness of each individual chromosome: f (x_1), f (x_2), … , f (x_N)
Step 5 : Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness.
Step 6: Create pair of offspring chromosomes by applying genetic operators - crossover & mutation.
Step 7 : Place the created offspring chromosomes in the new population.
Step 8 : Repeat Step 5 until the size of the new chromosome population becomes equal to the size of the initial population, N.
Step 9 : Replace the initial (parent) chromosome population with the new (offspring).
Step 10 : Go to Step 4, and repeat the process until the termination criterion is satisfied.

Backus–Naur Form (BNF)

BNF is a notation for expressing the grammar of a language in the form of production rules [3,16]. BNF grammars consist of terminals , which are items that can appear in the language, e.g., +,- , etc., and non-terminals , which can be expanded into one or more terminals and non-terminals. A grammar can
be represented by the tuple {N , T , P , S }, where N is the set of non-terminals, T the set of terminals, P a set of production rules that maps the elements of N to T , and S is a start symbol that is a member of N . When there are a number of productions that can be applied to one particular N , the choice is delimited with the ' | ' symbol. Below is an example BNF, where

N = { <expr> , <op> ,<fun> , <digit> , x , y , z}
T = { sin , cos ,exp , log , + , - , / , * }
S = <expr>
and P can be represented as in Fig.1.

Grammatical Evolution

Grammatical evolution is an evolutionary algorithm that can produce code in any programming language,[5 ,16]. The algorithm requires as inputs the BNF grammar definition of the target language and the appropriate fitness function. Chromosomes in grammatical evolution, in contrast to classical genetic programming [2], are not expressed as parse trees, but as vectors of integers. Each integer denotes a production rule from the BNF grammar. The algorithm starts from the start symbol of the grammar and gradually creates the program string, by replacing non terminal symbols with the right hand of the selected production rule. The selection is performed in two steps:[5].

- We read an element from the chromosome (with value V).
- We select the rule according to the scheme

Rule = V mod NR

where NR is the number of rules for the specific non-terminal symbol. The process of replacing non terminal symbols with the right hand of production rules is continued until either a full program has been generated or the end of chromosome has been reached. In the latter case we can reject the entire chromosome or we can start over (wrapping event) from the first element of the chromosome. In our approach we allow at most two wrapping events to occur. In our method we used grammar as we can see in Table 1. The numbers in parentheses denote the sequence number of the corresponding production rule to be used in the selection procedure described above.

Table 1: The grammar of the proposed method

S::=<expr>	
<expr> ::= <expr> <op> <expr>	(0)
| (<expr>)	(1)
| <func> (<expr>)	(2)
|<digit>	(3)
|x	(4)
|y	(5)
|z	(6)
<op> ::= +	(0)
| -	(1)
| *	(2)
| /	(3)
<func> ::= sin	(0)
|cos	(1)
|exp	(2)
|log	(3)
<digit> ::= 0	(0)
| 1	(1)
| 2	(2)
| 3	(3)
| 4	(4)
| 5	(5)
| 6	(6)
| 7	(7)
| 8	(8)
| 9	(9)

Further details about grammatical evolution can be found in [12, 13, 14, 15]
The symbol S in the grammar denotes the start symbol of the grammar. For example, suppose we have the chromosome g = [7 9 4 14 28 10 12 2 17 15 6 11 10 24 11]. In Table 2 , we show how a valid function is produced from g. The resulting function is : *f(x,y) = cos(x) + sin(y).*

Table 2: Illustrate example of program construction

String	Chromosome	Operation
<expr>	*7 2 9 4 12 2 20 12 31 4*	*7mod7=0*
<expr><op><expr>	*2 9 4 12 2 20 12 31 4*	*2mod 7=2*
Fun(<expr>)< op><expr>	*9 4 12 2 20 12 31 4*	*9mod4=1*
cos(<expr>)< op><expr>	*4 12 2 20 12 31 4*	*4mod7=4*
cos(x)<op><expr>	*12 2 20 12 31 4*	*12mod4=0*
cos(x) + <expr>	*2 20 12 31 4*	*2mod7=2*
cos(x) + Fun(<expr>)	*20 12 31 4*	*20mod4=0*
cos(x) + sin(<expr>)	*12 31 4*	*12mod7=5*
cos(x) + sin(y)		

Technique of the algorithm

The algorithm has the following phases:
1. Initialization.
2. Fitness evaluation.
3. Genetic operations.
4. Termination control.

Initialization

In the initialization phase the values for mutation rate and selection rate are set. The selection rate denotes the fraction of the number of chromosomes that will go through unchanged to the next generation(replication). The mutation rate controls the average number of changes inside a chromosome. Every chromosome in the population is initialized at random. The initialization of every chromosome is performed by randomly selecting an integer for every element of the corresponding vector.

Fitness evaluation

We express the PDE's in the following form:

$$f\left(x,y,\frac{\partial u}{\partial x}(x,y),\frac{\partial u}{\partial y}(x,y),\frac{\partial^2 u}{\partial x^2}(x,y),\frac{\partial^2 u}{\partial y^2}(x,y)\right)=0\ ,\ x\in[x_0,x_1]\ \ ,\ \ y\in[y_0,y_1]$$

The associated Dirichlet boundary conditions are expressed as:

$$u(x_0\ ,y\ \ )=f_0(y)\ \ \ ,u(x_1\ ,y\ \ )=f_1(y)\ \ \ ,\ \ u(x\ \ \ ,y_0)=g_0(y)\ \ \ \ ,\ u(x\ \ \ ,y_1)=g_1(y)$$

The steps for the fitness evaluation of the population are the following:

1. Choose N^2 equidistant points in the box $[x_0,x_1]\times[y_0,y_1]$, N_x equidistant points on the boundary at $x = x_0$ and at $x = x_1$, N_y equidistant points on the boundary at $y = y_0$ and at $y = y_1$
2. For every chromosome i
   - *i*- Construct the corresponding model $M_i(x,y)$, expressed in the grammar described earlier.
   - *ii*- Calculate the quantity

$$E(M_i)=\sum_{j=0}^{N^2}(f(x_j,y_j,\frac{\partial}{\partial x}M_i(x_j,y_j),\frac{\partial}{\partial y}M_i(x_j,y_j),\frac{\partial^2}{\partial x^2}M_i(x_j,y_j),\frac{\partial^2}{\partial y^2}M_i(x_j,y_j))^2$$

   - *iii*- Calculate an associated penalty $P_i(M_i)$. The penalty function P depends on the boundary conditions and it has the form:

$$P_1(M_i)=\sum_{j=1}^{N_x}(M_i(x_0,y_j)-f_0(y_j))^2$$
$$P_2(M_i)=\sum_{j=1}^{N_x}(M_i(x_1,y_j)-f_1(y_j))^2$$
$$P_3(M_i)=\sum_{j=1}^{N_y}(M_i(x_j,y_0)-g_0(x_j))^2$$
$$P_4(M_i)=\sum_{j=1}^{N_y}(M_i(x_j,y_1)-g_1(x_j))^2$$

   - *iv*- Calculate the fitness value of the chromosome as:

$$v_i=E\ \ (M_i)+P_1(M_i)+P_2(M_i)+P_3(M_i)+P_4(M_i)$$

Genetic operators

The genetic operators that are applied to the genetic population are the initialization, the crossover and the mutation.

The initialization is applied only once on the first generation. For every element of each chromosome a random integer in the range [0..255] is selected.

The crossover is applied every generation in order to create new chromosomes from the old ones, that will replace the worst individuals in the population. In that operation for each couple of new chromosomes two parents are selected, we cut these parent - chromosomes at a randomly chosen point and we exchange the right-hand-side sub-chromosomes, as shown in Fig.1.

- *Parent 1:* *2 20 14 |5 25 18*
- *Parent 2:* *8 13 17 | 28 3 30*
- *Offspring 1:* *2 20 14 28 3 30*
- *Offspring 2:* *8 13 17 5 25 18*

Fig.1 : Crossover

The parents are selected via tournament selection, i.e. :

- First, we create a group of $K >= 2$ randomly selected individuals from the current population.
- The individual with the best fitness in the group is selected, the others are discarded.

The final genetic operator used is the mutation, where for every element in a chromosome a random number in the range [0 , 1] is chosen. If this number is less than or equal to the mutation rate the corresponding element is changed randomly, otherwise it remains intact.
In every generation the following steps are performed:

1. The chromosomes are sorted with respect to their fitness value, in a way that the best chromosome is placed at the beginning of the population and the worst at the end.
2. $c = (1 - s) * g$ new chromosomes are produced by the crossover operation, where s is the replication rate of the model and g is the total number of individuals in the population. The new individuals will replace the worst ones in the population at the end of the crossover.
3. The mutation operation is applied to every chromosome excluding those which have been selected for replication in the next generation.

Termination control

The genetic operators are applied to the population creating new generations, until a maximum number of generations is reached or the best chromosome in the population has fitness better than a preset threshold.

TECHNIQUES OF THE ACCELERATED METHOD

To make the method is faster to arrive the exact solution of the partial differential equations by the following :

1- insert the boundary conditions of the problem as a part of chromosomes in the our population of the problem, the algorithm gives the exact solution a few generations. i.e. y^3 *= [28 5 2 7 5 2 5 15 2]* represent the boundary condition of problem.
2- insert a part of exact solution (or particular solution) as a part of a chromosome in the population, we find the algorithm gives the exact solution in a few generations.
3- insert the vector of exact solution (if exist) as a chromosome in the our population of the problem, the algorithm gives the exact solution in the first generation . i.e. *cos(x)cos(y) = [7 2 9 4 14 2 21 12 31 4]* the solution of example 2.

Applications of the Algorithm

We describe several experiments performed on Partial differential equations .And we applied our method to PDE's that do not posses an analytical closed form solution and hence can not be represented exactly by the grammar. We used 25% for the replication rate (hence the crossover probability is set to 75%) and 1% for the mutation rate. We investigated the importance of these two parameters by performing experiments using sets of different values. Each experiment was performed 20 times . As one can see the performance is somewhat dependent on these parameters, but not critically. The population size was set to 100 and the length of each chromosome to 50. The size of the population is a critical parameter. Too small a size weakens the method's effectiveness. Too big a size renders the method slow. Hence since there is no first principals estimation for the the population size, we resorted to an experimental approach to obtain a realistic determination of its range. It turns out that values in the interval [0 , 255] are proper. We used fixed – length chromosomes instead of variable - length to avoid creation of unnecessary large chromosomes which will render the method inefficient. The length of the chromosomes is usually depended on the problem to be solved. The maximum number of generations allowed was set to 100 and the preset fitness target for the termination criteria was 10^{-4}. From the conducted experiments, we have observed that the maximum number of generations allowed must be greater for difficult problems .

The value of N for PDE's was set to 5 and $N_x = N_y = 50$. depending on the problem, Also, for some problems we present graphs of the intermediate trial solutions. In all experiments we use Mat lab R2010a ,And use the function (*randi*) to generate the random integers with normal distribution where used to generation the population . In each experiment we insert the the boundary conditions as apart of chromosome in the population to renders the method fast to arrives the exact solution of the problems.

Solutions to Elliptic PDE's

A general form of an elliptic PDE's (Poisson equation) are :

$\nabla^2(x,y) = \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y)$ on $R = \{(x,y): a < x < b, c < y < d\}$ with $u(x,y) = g(x,y)$, $for\ all\ (x,y) \in S$, where S is the boundary of R .

If f & g are continuous on there domains , Then there exist a unique solution to this equation .

Example 1:-

$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = 0$, for (x,y) in the set $R = \{(x,y): 0 < x < 0.5\,, 0 < y < 0.5\}$

With the boundary conditions $u(0,y) = 0\,, u(x,0) = 0\,, u(x,0.5) = 200x\,, u(0.5,y) = 200y$

The exact solution *u(x,y)* = 400xy recovered at the 10th generation. At generation 1 the trial solution was *GP2(x, y) = 200xy* with fitness value 8.3607e+004 . At the 2nd generation the trial solution was *GP3(x, y) = exp(y) + 350xy-1* with fitness value 3.9236*1e4 as shown in fig.2. The error = exact solution–trail solution show in Table 3.

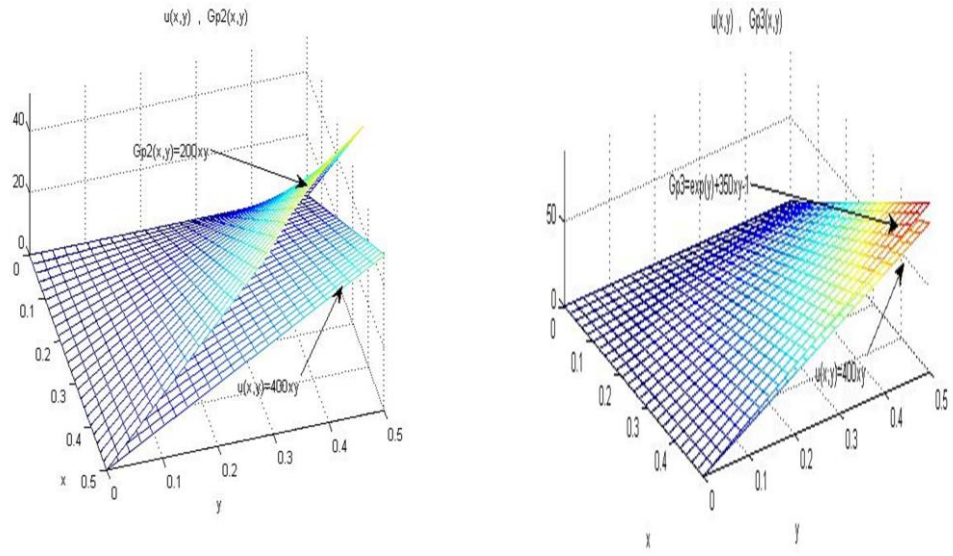

Fig.2 : Exact and trail solutions of example 1

Table 3: Error of Example 1

x	y	u= 400xy	Gp3=exp(y)+350xy-1	Error
0	0	0	0	0
0.0556	0.0556	1.2346	1.1374	0.0972
0.1111	0.1111	4.9383	4.0805	0.4998
0.1667	0.1667	11.1111	9.9036	1.2075
0.2222	0.2222	19.7531	17.5328	2.2203

Example 2 *:-*

$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = -(\cos(x+y) + \cos(x-y))$

for (x,y) in the set $R = \{(x,y): 0 < x < \pi\,, 0 < y < \frac{\pi}{2}\}$

With the boundary conditions $u(0,y) = cosy\,, u(x,0) = cosx\,, u\left(x, \frac{\pi}{2} = 0\right) = 200x\,, u(\pi,y) = -cosy$

The exact solution *u(x,y) = cosxcosy* . recovered at the 5th generation. with fitness value 0.7883*1e-30. as shown in Fig.3.

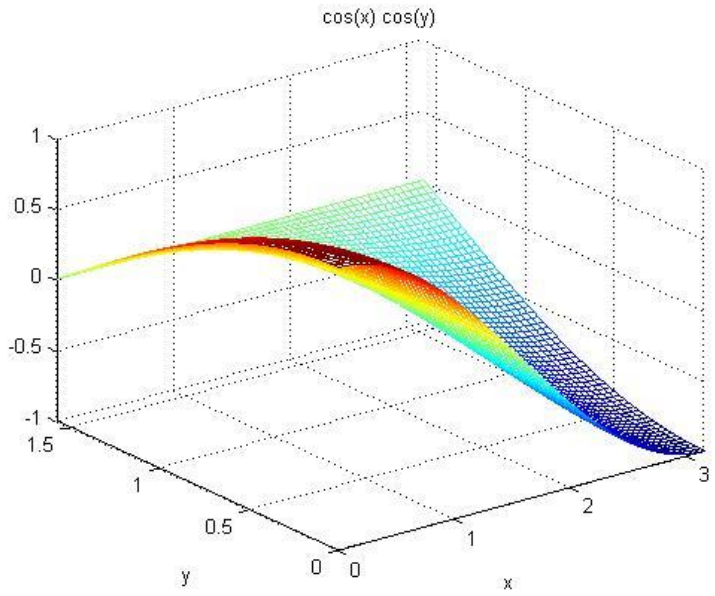

Fig.3 : Exact solution of example 2

Solutions to Parabolic PDE's

The Parabolic PDE's we study the heat equation of the form :
$\frac{\partial u}{\partial t}(x,t) = \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t)$, $0 < x < l$, $t > 0$ subject to the conditions :
$u(0,t) = u(l,t) = 0$, $u(x,0) = f(x)$, $t > 0$, $0 \le x \le l$

Example 3:-

$\frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 0$, $0 < x < 1$, $t > 0$ with boundary conditions
$u(0,t) = u(1,t) = 0$, $0 < t$ and initial condition $u(x,0) = sin\pi x$, $0 \le x \le 1$
The exact solution $u(x,t) = e^{-\pi^2 t}\sin(\pi x)$. recovered at the 50th generation. At generation 1 the trial solution was *GP1(x, t) = sin(x)/exp(t)* with fitness value 12.3667. as shown in Fig.4.

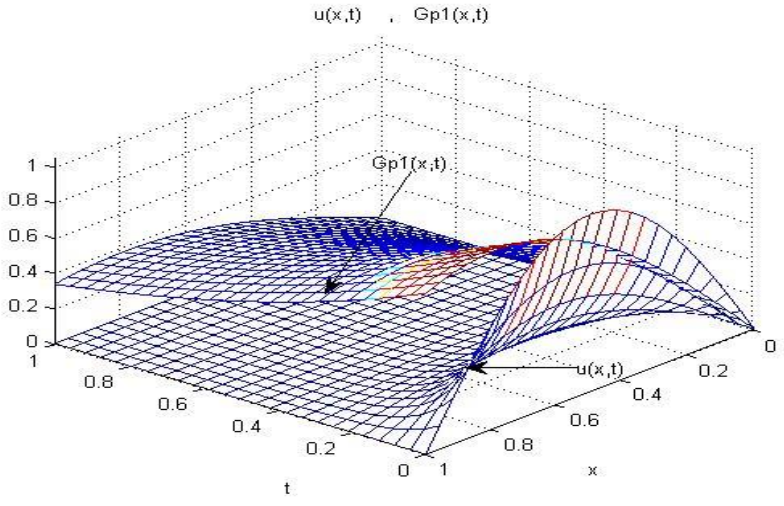

fig.4 : Exact and trail solutions of example *3*

Example 4:-

$\frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 2 \quad , \ 0 < x < 1 \quad , t > 0$ with boundary conditions
$u(0,t) = u(1,t) = 0 \ , 0 < t$ and initial condition $u(x,0) = \sin(\pi x) + x(1-x) \ , 0 \le x \le 1$
The exact solution $u(x,t) = e^{-\pi^2 t}\sin(\pi x) + x(1-x)$. recovered at the generation 58. At generation 1 the trial solution was *GP1(x, t) = x + sin(x)/exp(t) - x^2* with fitness value 12.3667 as shown in Fig. 5. The error = exact solution – trail solution shown in Table 4 .

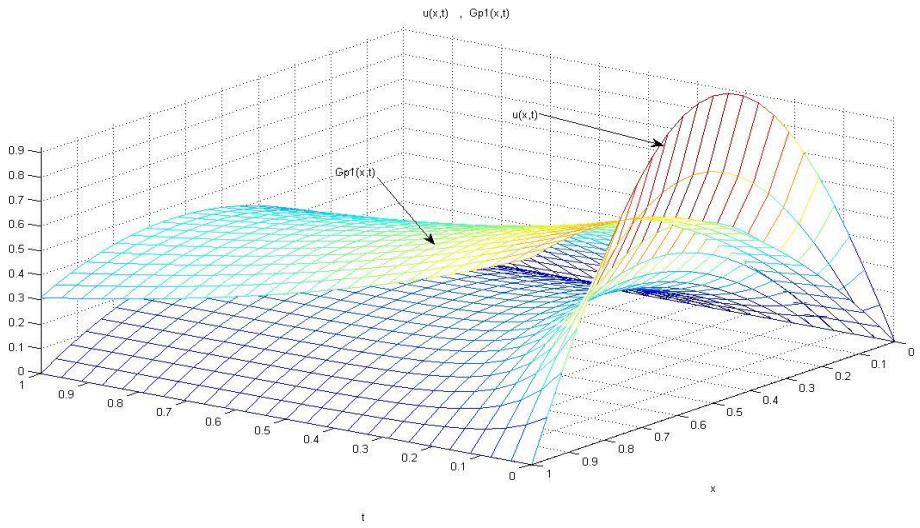

Fig.5 : Exact and trail solutions of example *4*

Table 4: Error of Example 4

x	t	u= $e^{-\pi^2 t}\sin(\pi x) + x(1-x)$	Gp1= *x + sin(x)/exp(t) - x^2*	Error
0	0	0	0	0
0.2500	0.2500	0.2475	0.3802	0.1327
0.5000	0.5000	0.2572	0.5408	0.2836
0.7500	0.7500	0.1879	0.5095	0.3216
1.0000	1.0000	0	0.3096	0.3096

Solution of Hyperbolic PDE's

We consider the wave equation
$\frac{\partial^2 u}{\partial t^2}(x,t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t) = 0 \ , 0 < x < l \ , t > 0$ subject to the conditions
$u(0,t) = u(l,t) = 0 \ , t > 0$ and $u(x,0) = f(x) \ \ and \ \ \frac{\partial u}{\partial t}(x,0) = g(x) \ , 0 \le x \le l$

Example 5:-

$\frac{\partial^2 u}{\partial t^2}(x,t) - 4\frac{\partial^2 u}{\partial x^2}(x,t) = 0 \ , 0 < x < 1 \ , t > 0$ subject to the conditions
$u(0,t) = u(1,t) = 0 \ , t > 0$ and $u(x,0) = \sin(\pi x) \ \ and \ \ \frac{\partial u}{\partial t}(x,0) = 0 \ , 0 \le x \le 1$.
The exact solution $u(x,t) = \sin(\pi x)\cos(2\pi t)$. recovered at the generation 30. At generation 1 the trial solution was *GP10(x,t) = cos(4t)sin(2x)* with fitness value 51.4329. as shown in fig.6. The error = exact solution – trail solution shown in table 4 .

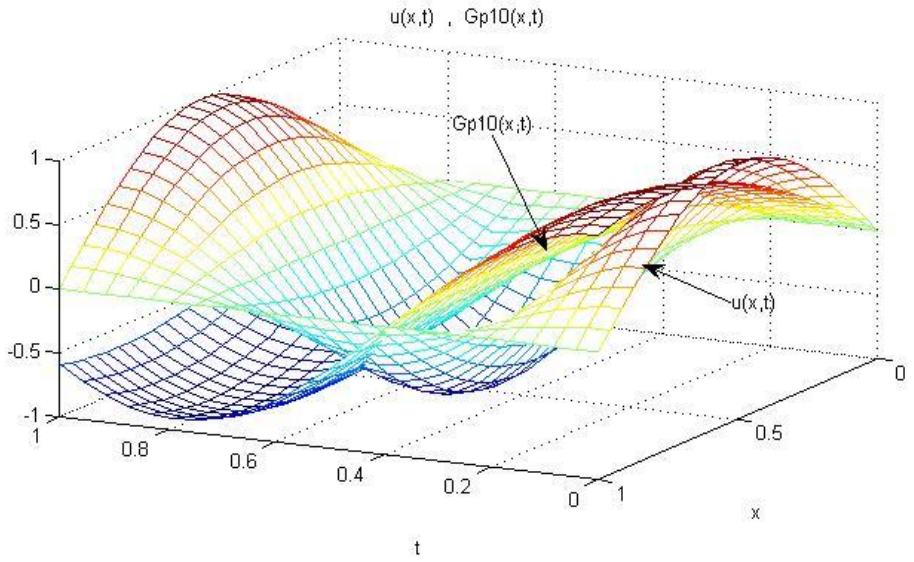

Fig.6 : Exact and trail solutions of example*5*

Table 5: Error of Example 5

x	t	u = $\sin(\pi x)\cos(2\pi t)$	Gp1 = $\sin(x)\cos(2t)$	Error
0	0	0	0	0
0.2500	0.2500	0.0000	0.2590	0.2590
0.5000	0.5000	-1.0000	-0.3502	0.6498
0.7500	0.7500	-0.0000	-0.9875	0.9875
1.0000	1.0000	0.0000	-0.5944	0.5944

Example 6:-

$\frac{\partial^2 u}{\partial t^2}(x,t) - \frac{1}{16\pi^2}\frac{\partial^2 u}{\partial x^2}(x,t) = 0 \;\; , 0 < x < 0.5 \;\; , t > 0$ subject to the conditions

$u(0,t) = u(0.5,t) = 0 \;\; , t > 0$ and $u(x,0) = 0 \;\; and \;\; \frac{\partial u}{\partial t}(x,0) = \sin(4\pi x) \;\; , 0 \le x \le 0.5$

The exact solution $u(x,t) = \sin(t)\sin(4\pi x)$. recovered at the generation 31. At generation 1 the trial solution was *GP1(x, t) = sin(5x)sin(t)* with fitness value 1.7290. as shown in Fig. 7.

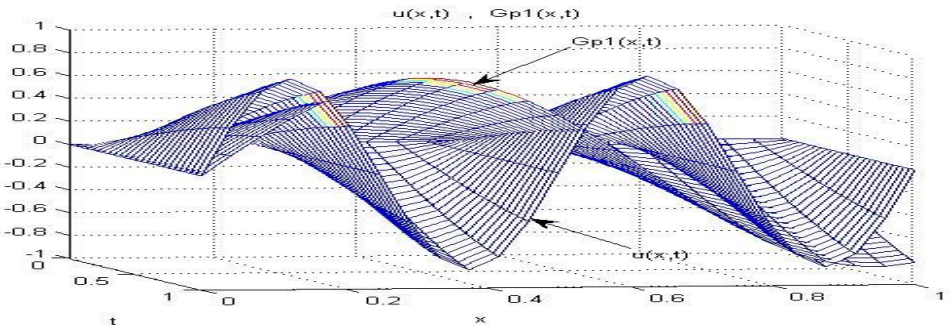

Fig.7 : Exact and trail solutions of example 6

Comparison the Method

The comparison of the present method with other grammatical evolutionary methods ,we found that this method is faster than others, because the technical of accelerated the method. In the Table.5 followed below ,we compared the current method with the study of [5] , where Min , Max , Avg ,means minimum ,maximum ,average of generations to find the exact solution of the PDE's .

Table 5 : comparison the current method with other methods

Previous method in [5]				Our method		
PDE	Min	Max	Avg	Min	Max	Avg
$\nabla^2\Psi(x,y) = e^{-x}(x-2+y^3+6y)$	159	1772	966	3	70	30
$\nabla^2\Psi(x,y) = -2\Psi(x,y)$	5	1395	203	5	76	42
$\nabla^2\Psi(x,y) = 4$	18	311	154	2	35	15
$\nabla^2\Psi(x,y) = -(x^2+y^2)\Psi(x,y)$	4	1698	207	8	82	20

Convergence of the Method

We denote by u_t the best adapted individual in the population, at the instance t, i.e. the individual in the population P (t) which has the minimum value of E . we have already stated that the sequence (u_t) $t>=0$ converges, its limit being the solution of the optimization problem inf E . While the solution is the limit of a convergent sequence, by applying the genetic algorithm, the following assertion is true:

For $\varepsilon > 0$, there is a chromosome $g=(g_1 ,g_2 , \ldots,g_n)$ such that:

$$E(M_i) = \sum_{j=0}^{N^2}\left(f\left(x_j, y_j, \frac{\partial}{\partial x}M_i(x_j,y_j), \frac{\partial}{\partial y}M_i(x_j,y_j), \frac{\partial^2}{\partial x^2}M_i(x_j,y_j), \frac{\partial^2}{\partial y^2}M_i(x_j,y_j)\right)^2 < \in$$

CONCLUSION

In this paper new accelerated method of GA was introduced, and applied for the purposes of solving PDE's . It is noted that this method has general utility for applications. GA is an evolutionary algorithm that can evolve ' rulesets '. In this study we found that insertion of boundary condition in population make the algorithm fast to approximate the exact solution. Our method for this initial study has included a number of simplifications, for example we only considered a small set of equations . The study could also be extended by constructing a another type of partial differential equations .

REFERENCES

[1] A. Beham, M. Affenzeller ,Genetic Algorithms and Genetic Programming, Modern Concepts and Practical Applications , Berlin, Germany, 2009.
[2] J. D. Bagley, The Behavior of Adaptive Systems Which Employ Genetic and Correlative Algorithms. PhD thesis, University of Michigan,1967.
[3] C. Ryan, M. O'Neill, and J.J. Collins, Grammatical Evolution: Solving Trigonometric Identities," In proceedings of Mendel 1998.
[4] D.E. Goldberg, Genetic algorithms in search, Optimization and Machine Learning, Addison Wesley, 1989.
[5] G. Tsoulos .I.E, Solving differential equations with genetic programming P.O.Box 1186 , Ioannina 45110, 2003.

[6] J. H. Holland , Adaptation in Natural and Artificial Systems, first MIT Press ed. The MIT Press, Cambridge, MA, 1992. First edition: University of Michigan Press, 1975.
[7] J. H. Holland, K. J. Holyoak, R.E.Nisbett, and P. R. Thagard, Induction: Processes of Inference, Learning, and Discovery. Computational Models of Cognition and Perception. The MIT Press, Cambridge, MA, 1986.
[8] H.P. Schwefel. ,Numerische Optimierung von Computer-Modellen mittels der Evolutions strategie, Birkhauser Verlag,Basel, Switzerland, 1994.
[9] I. Rechenberg . Evolutions strategies . Friedrich Frommann Verlag, 1973.
[10] J. R. Koza, Genetic Programming: On the programming of Computer by Means of Natural Selection. MIT Press: Cambridge, MA, 1992.
[11] R. Kruse, J. Gebhardt, and . Klawonn, Foundations of Fuzzy Systems. John Wiley & Sons, New York, 1994.
[12] M. O'Neill and C. Ryan, Under the hood of grammatical evolution,1999.
[13] M. O'Neill and C. Ryan, Evolving Multi-Line Compliable C Programs, Springer-Verlag, pp. 83-92, 1999.
[14] M. O'Neill and C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming. Kluwer Academic Publishers, 2003.
[15] M. O'Neill and C. Ryan, Grammatical Evolution, IEEE Trans. Evolutionary Computation, Vol. 5, pp. 349-358, 2001.
[16] P. Naur, "Revised report on the algorithmic language ALGOL 60,"Commun. CM, vol. 6, no. 1, pp. 1–17, Jan. 1963.
[17] R. H. J. M. Otten, , and L. P. P. P. van Ginneken, The Annealing Algorithm. Kluwer Academic Publishers, Boston, 1989.
[18] D. E Rumelhart, and J. L. McClelland, Parallel Distributed Processing Exploration in the Microstructures of Cognition, Volume I:Foundations. MIT Press, Cambridge, MA, 1986.
[19] P. J. M. van Laarhoven, , and E. H. L., Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, Dordrecht, 1987.
[20] P. Whigham, Grammatically-based Genetic Programming. In Proceeding of the Workshop on Genetic Programming : From Theory to Real-World Applications, pages 33-41. Morgan Kaufmann Pub. 1995.
[21] H.-J Zimmermann,. Fuzzy Set Theory and its Applications, second ed. Kluwer Academic Publishers, Boston, 1991.[32] Zurada, J. M. Introduction to Artificial Neural Networks. West Publishing ,St. Paul, 1992.