OPTIMIZATION USING DIFFERENTIAL EVOLUTION

By

VASAN ARUNACHALAM

Facility for Intelligent Decision Support
Department of Civil and Environmental Engineering
The University Of Western Ontario, London, Ontario, Canada
JULY 2008
ABSTRACT

The book explains in detail the working of Differential Evolution optimization algorithm. It also provides documentation for the use of Differential Evolution computer program to solve user-defined optimization problems. The computer program is written in C language for Windows environment. The book also demonstrates how to modify the program using an example optimization problem.

This source code is distributed for academic purposes only. It has no warranty implied or given, and the author assumes no liability for damage resulting from its use or misuse. The author can be contacted for any comments by email at vasan.arunachalam@gmail.com.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>ii</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 DIFFERENTIAL EVOLUTION</td>
<td>3</td>
</tr>
<tr>
<td>WORKING OF THE DE ALGORITHM</td>
<td>3</td>
</tr>
<tr>
<td>AN ILLUSTRATIVE EXAMPLE</td>
<td>7</td>
</tr>
<tr>
<td>3 HOW TO RUN THE DE PROGRAM?</td>
<td>10</td>
</tr>
<tr>
<td>4 CONCLUSION</td>
<td>19</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>20</td>
</tr>
<tr>
<td>APPENDIX A - SOURCE CODE OF THE DE ALGORITHM IN C</td>
<td>21</td>
</tr>
<tr>
<td>APPENDIX B - PREVIOUS REPORTS IN THE SERIES</td>
<td>35</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Optimization is a procedure through which the best possible values of decision variables are obtained under the given set of constraints and in accordance to a selected optimization objective function. The most common optimization procedure applies to a design that will minimize the total cost or maximize the possible reliability or any other specific objective. Fields of science and engineering, business decision-making and industry are all rich in problems that require the implementation of optimization approach. Since, most real world optimization problems seem to be both fundamentally and practically hard, research into better algorithms remains valuable and continues, so that, one can guarantee to find the best solution using an efficient optimization algorithm.

Nowadays, there exist a lot of optimization algorithms that work using gradient-based and heuristic-based search techniques in deterministic and stochastic contexts. In order to widen the applicability of the optimization approach to various problem domains, natural and physical principles are mimicked to develop robust optimization algorithms. Evolutionary algorithms, simulated annealing, ant colony optimization, memetic algorithms, particle swarm optimization are few examples of such algorithms.

Over the last decade, evolutionary algorithms have been extensively used in various problem domains and succeeded in effectively finding the near optimal solutions. The present book provides a detailed description of one such evolutionary algorithm, named, *Differential Evolution (DE)*. Since its inception in 1995, DE has earned a reputation of a very effective global optimizer (Storn and Price, 1995). Chapter 1 begins with an introduction to the optimization and Chapter 2 introduces the detailed working of DE.
algorithm with an illustrative example. Chapter 3 provides the instructions for using the DE computer program developed for Windows environment that accompanies this book on CD-ROM. Chapter 4 concludes with few practical suggestions for the users of the DE computer program. An Appendix to the text contains description of the source code of DE software developed in C programming language.
CHAPTER 2

DIFFERENTIAL EVOLUTION

WORKING OF THE DE ALGORITHM

Differential Evolution (DE) algorithm is a branch of evolutionary programming developed by Rainer Storn and Kenneth Price (Price and Storn, 1997) for optimization problems over continuous domains. In DE, each variable’s value is represented by a real number. The advantages of DE are its simple structure, ease of use, speed and robustness. DE is one of the best genetic type algorithms for solving problems with the real valued variables. Differential Evolution is a design tool of great utility that is immediately accessible for practical applications. DE has been used in several science and engineering applications to discover effective solutions to nearly intractable problems without appealing to expert knowledge or complex design algorithms. If a system is amenable to being rationally evaluated, DE can provide the means for extracting the best possible performance from it. Differential Evolution uses mutation as a search mechanism and selection to direct the search toward the prospective regions in the feasible region. Genetic Algorithms generate a sequence of populations by using selection mechanisms. Genetic Algorithms use crossover and mutation as search mechanisms. The principal difference between Genetic Algorithms and Differential Evolution is that Genetic Algorithms rely on crossover, a mechanism of probabilistic and useful exchange of information among solutions to locate better solutions, while evolutionary strategies use mutation as the primary search mechanism.

DE is a population based search technique which utilizes NP variables as population of D dimensional parameter vectors for each generation. The initial population is chosen
randomly if no information is available about the problem. In the case of the available preliminary solution, the initial population is often generated by adding normally distributed random deviations to the preliminary solution. The basic idea behind DE is a new scheme for generating trial parameter vectors. DE generates new parameter vectors by adding the weighted difference vector between two population members to a third member. If the resulting vector yields a lower objective function value than a predetermined population member, the newly generated vector replaces the vector with which it was compared. In addition, the best parameter vector is evaluated for every generation in order to keep track of the progress that is made during the optimization process. Extracting the distance and the direction information from the population to generate random deviations result in an adaptive scheme with excellent convergence properties (Price et al., 2005).

DE maintains two arrays, each of which holds a population size $NP$ and $D$ dimensional, real-valued vectors. The primary array holds the current vector population, while the secondary array accumulates vectors that are selected for the next generation. In each generation, $NP$ competitions are held to determine the composition of the next generation. Every pair of vectors $(X_a, X_b)$ defines a vector differential: $(X_a - X_b)$. When $X_a$ and $X_b$ are chosen randomly, their weighted differential is used to perturb another randomly chosen vector $X_c$. This process can be mathematically expressed as:

$$X'_c = X_c + F(X_a - X_b)$$  \hspace{1cm} (1)$$

The weighting, or scaling, factor $F$ is a user supplied constant in the optimal range between 0.5 and 1.0 (DE, 2008). In every generation, each primary array vector $X_i$ is targeted for crossover with a vector like $X'_c$ to produce a trial vector $X'_i$. Thus, the trial vector is the child of two parents, a noisy random vector and the target vector against which it must compete. Uniform crossover (that can take child vector parameters from
one parent more often than it does from others) is used with a crossover constant \((CR)\), in the optimal range of 0.5 to 1.0 (DE, 2008) which actually represents the probability that the child vector inherits the parameter values from the noisy random vector. When \(CR = 1\), for example, every trial vector parameter is certain to come from \(X_c\). On the other hand, if \(CR = 0\), all but one trial vector parameter comes from the target vector. To ensure that \(X_t\) differs from \(X_i\) by at least one parameter, the final trial vector parameter always comes from the noisy random vector even when \(CR = 0\). Then the objective function corresponding to the trial vector is compared with that of the target vector, and the vector that has the lower objective function value (for minimization) of the two would survive for the next generation. This process is continued until the termination criterion of a preset maximum number of generations \((MAXGEN)\) is met, and difference in objective function values between two consecutive generations reaches a small value. Figure 1 shows the working of the DE algorithm.

![Fig. 1. Working of Differential Evolution Algorithm](image-url)
Price & Storn (1997) gave the working principle of DE with single strategy. Later on, they suggested ten different strategies for DE. Different strategies can be adopted in the DE algorithm depending upon the type of problem to which DE is applied. The strategies can vary based on the vector to be perturbed, number of difference vectors considered for perturbation, and finally the type of crossover used. The following are the ten different working strategies:

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

The general convention used above is $DE/x/y/z$. $DE$ stands for Differential Evolution, $x$ represents a string denoting the vector to be perturbed, $y$ is the number of difference vectors considered for perturbation of $x$, and $z$ stands for the type of crossover being used (exp: exponential; bin: binomial). Hence the perturbation can be either in the best vector of the previous generation or in any randomly chosen vector. Similarly for perturbation either single or two vector differences can be used. For perturbation with a single vector difference, out of the three distinct randomly chosen vectors, the weighted vector differential of any two vectors is added to the third one. Similarly for perturbation with two vector differences, five distinct vectors, other than the target vector are chosen randomly from the current population. Out of these, the weighted vector difference of each pair of any four vectors is added to the fifth one for perturbation. In exponential crossover, the crossover is performed on the D variables in one loop until it is within the CR bound. The first time a randomly picked number between 0 and 1 goes beyond the CR value, no crossover is performed and the remaining D variables are left intact. In
**binomial crossover**, the crossover is performed on each of the D variables whenever a randomly picked number between 0 and 1 is within the CR value. So for high values of CR, the exponential and binomial crossover methods yield similar results.

A strategy that works out to be the best for a given problem may not work well when applied to a different problem. Also, the strategy and the key parameters to be adopted for a problem are to be determined by trial and error. However, strategy-7 (DE/rand/1/bin) appears to be the most successful and the most widely used strategy. In all, three factors control evolution under DE, the population size NP, the weight applied to the random differential F and the crossover constant CR. More details regarding DE are available in Price and Storn (1997), Onwubolu and Babu (2004) and Price and Storn (2005).

**AN ILLUSTRATIVE EXAMPLE**

A simple numerical example is presented to illustrate the DE algorithm. Let us consider the following objective function:

\[
\text{Minimize } f(x) = x_1 + x_2 + x_3
\]

The initial population is generated (chosen randomly between the bounds of decision variables, in this case within 0 and 1, for the three decision variables. The population along with its respective objective function values is shown in Table 1. The first member of the population “Individual 1” is set as the target vector.

In order to generate the noisy random vector, three individuals (Individual 2, Individual 4 and Individual 6) from the population size are selected randomly (ignoring “Individual 1”, since it is set as the target vector). The weighted difference between “Individual 2” and “Individual 4” is added to the third randomly chosen vector “Individual 6” to generate the noisy random vector. The weighting factor F is chosen as 0.80 and the weighted difference vector is obtained in Table 2 and the noisy random vector in Table 3.
Table 1. An illustrative example

<table>
<thead>
<tr>
<th>Population Size $NP = 6$ (user defined), $D = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual 1</td>
</tr>
<tr>
<td>$x_1$</td>
</tr>
<tr>
<td>$x_2$</td>
</tr>
<tr>
<td>$x_3$</td>
</tr>
<tr>
<td>$f(x)$</td>
</tr>
</tbody>
</table>

Table 2. Calculation of the weighted difference vector for the illustrative example

<table>
<thead>
<tr>
<th>Individual 2</th>
<th>Individual 4</th>
<th>Difference Vector</th>
<th>Weighted Difference Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.92</td>
<td>0.12</td>
<td>= 0.80</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.92</td>
<td>-</td>
<td>0.09</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.33</td>
<td>0.05</td>
<td>= 0.28</td>
</tr>
</tbody>
</table>

Table 3. Calculation of the noisy random vector for the illustrative example

<table>
<thead>
<tr>
<th>Weighted Difference Vector</th>
<th>Individual 6</th>
<th>Noisy Random Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.64</td>
<td>0.94</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.66</td>
<td>+</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.22</td>
<td>0.13</td>
</tr>
</tbody>
</table>

The noisy random vector does a crossover with the target vector to generate the trial vector as shown in Table 4. This is carried out by (1) generating random numbers equal to the dimension of the problem (2) for each of the dimensions: if random number $> CR$; copy the value from the target vector, else copy the value from the noisy random vector into the trial vector. In this example, the crossover constant $CR$ is chosen as 0.50.
Table 4. Generation of the trial vector for the illustrative example

<table>
<thead>
<tr>
<th>Target Vector</th>
<th>Noisy Random Vector</th>
<th>Trial Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0.68</td>
<td>Crossover $=$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.89</td>
<td>($CR = 0.50$)</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>$f(x)$</td>
<td>1.61</td>
<td>3.22</td>
</tr>
</tbody>
</table>

The objective function of the trial vector is compared with that of the target vector and the vector with the lowest value of the two becomes “Individual 1” for the next generation. To evolve “Individual 2” for the next generation, the second member of the population is set as target vector and the above process is repeated. This process is repeated $NP$ times till the new population set array is filled which completes one generation. Once the termination criterion is met, the algorithm ends.

Table 5. New population for next generation for the illustrative example

<table>
<thead>
<tr>
<th>New Population for Next Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual 1</td>
</tr>
<tr>
<td>$x_1$</td>
</tr>
<tr>
<td>$x_2$</td>
</tr>
<tr>
<td>$x_3$</td>
</tr>
<tr>
<td>$f(x)$</td>
</tr>
</tbody>
</table>
Single objective Differential Evolution optimization algorithm has been programmed in C language using Microsoft Visual C++ 6.0 for Windows. This Chapter explains how to use the DE computer program for solving any optimization problem. The source code of the DE program is available in Appendix I. The CD enclosed along with this book contains a folder named “Program”. The folder contains the C program file “DE.c” which requires two input files, “Limits.txt” and “ParameterLimits.txt”. The program generates two output files, “ReportDE.html” and “Convergence.txt”.

In order to learn how to use the DE program, consider the following optimization problem:

Minimize \( f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \)

subject to

\( (x_1 - 5)^2 + x_2^2 - 26 \leq 0 \)
\( 4x_1 - x_2 - 20 \leq 0 \)
\( x_1, x_2 \geq 0 \)

The algorithm is designed to accept optimization problem with an objective of minimization type. If the objective of the problem is of maximization type, it should be modified accordingly to use in the program. Each decision variable is represented from \( x[0], x[1], x[2] \) and so on. Accordingly, the objective function for the example is written as follows:

\( (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11)+(x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7) \)
To enter the constraints, the inequality constraints should be modified to the less or equal format, \( g(x) \leq 0 \). If the problem is an unconstrained optimization problem, the user need not enter anything in the space specified for the constraints coding. The constraints of the example are entered as follows:

/* Constraint 1*/
\[
temp = (x[0]-5) \times (x[0]-5) + x[1] \times x[1] - 26;
\]
val[1] = max(temp,0);

/* Constraint 2 */
\[
temp = 4 \times x[0] + x[1] - 20;
\]
val[2] = max(temp,0);

If the first constraint of the example is an equality constraint, the constraint should be entered as follows:

/* Constraint 1 (Equality Condition) */
\[
val[1] = abs((x[0]-5) \times (x[0]-5) + x[1] \times x[1] - 26);
\]

where val[1] and val[2] represent the amount of violation for each constraint. These values are multiplied by a penalty co-efficient penal[1] and penal[2] (entered by the user of the program), which is then added to the objective function to continue the process of optimization. This process is often termed as penalty function approach. A simple additive penalty function approach is used in order to convert the constrained problem into unconstrained problem. Due to this conversion, the solution falling outside the feasible region is penalized and forced to fall into the feasible solution space after a few generations. However, the penalty function method has certain weaknesses that are fatal when the penalty parameters are large. Penalty functions tend to be ill conditioned near the boundary of the feasible domain and that may result in a local optimal solution or an
infeasible solution. Careful selection of the penalty parameters is required for the proper convergence to a feasible optimal solution.

The main C program has many user-defined functions. The user needs to enter objective function and constraints in the user-defined function evaluate( ). The program code is commented, so the user needs to follow the comments to enter the number of constraints, ncons (in this example, ncons = 2), objective function, constraints (if any) and the penalty coefficient for each constraint (if any), at the appropriate places. The “evaluate( )” function for the example problem is displayed below:

double evaluate(double *x)
{
    int ncons, i;
    double *val, *penal, temp;
    long double value=0;
    feval++;

    /* DO NOT CHANGE ANYTHING ABOVE */
    /* Enter the Number of Constraints to the variable ncons*/
    ncons = 2;
    /*-----------------------------------------------*/
    val   = (double *)malloc(sizeof(double)*ncons);
    penal = (double *)malloc(sizeof(double)*ncons);
    penal[0]=1;

    /*-------------------------------CODE YOUR OBJECTIVE FUNCTIONS HERE-------------------*/
    /*All functions must be of minimization type*/
    /*====================================Start Coding Your Objective Function=================*
    val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) +
            (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7);
    /*================================END YOUR CODING UPTO THIS POINT======================*/
/*----------------------CODE YOUR CONSTRAINTS HERE--------------*/
/*All functions must be of minimization type, negate maximization functions */
/*Constraints must be of the following type: g(x) <= 0. Enter the constraint
equation to temp variable and don't change the expression for val[1].
Similarly, do the same for other constraints. */
/*=============Start Coding Your Constraints============*/
/* Constraint 1*/
 temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
 val[1] = max(temp,0);

/* Constraint 2*/
 temp = 4*x[0] + x[1] -20;
 val[2] = max(temp,0);

/*Enter the Penalty Coefficient for Each constraint*/
 penal[1] = 1000;
 penal[2] = 1000;
/*========END YOUR CODING UPTO THIS POINT==========*/
 for (i=0;i<=ncons;i++) value = (value + penal[i]*val[i]);
 return value;
}

Similarly, the objective function without constraints needs to be entered in the function
“evaluateWOpenalty( )”. The “evaluateWOpenalty( )” function for the example problem
is displayed below:

double evaluateWOpenalty(double *x)
{
    double value;
    /* DO NOT CHANGE ANYTHING ABOVE */
/*------------------------CODE YOUR OBJECTIVE FUNCTIONS HERE----------------*/
/*All functions must be of minimization type*/
/*====================Start Coding Your Objective Function====================*/

value = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) +
(x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7);

/*====================END CODING UPTO THIS POINT====================*/

return value;
}

After inserting the objective function and constraints, the user needs to prepare the two input files (“Limits.txt” and “ParameterLimits.txt”). In the “Limits.txt” input file, the lower and upper bound for each decision variable separated by a tab, is entered. The number of decision variables for the example is two. The lower and upper bounds for each variable is assumed as 0 and 10 respectively. The “Limits.txt” input file for the example is as follows:

<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

The “ParameterLimits.txt” input file requires the number of decision variables (in the first row), maximum number of generations (in the second row), minimum and maximum number of population (NP), crossover constant (CR), weighting factor (F) along with their step length for sensitivity analysis in the third, fourth and fifth rows respectively. The input file for the example problem is as follows:

<table>
<thead>
<tr>
<th>Decision Variables</th>
<th>NP</th>
<th>CR</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Minimum</td>
<td>Maximum</td>
<td>Step Length</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>30</td>
<td>20 20 10</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.6</td>
<td>0.1</td>
</tr>
</tbody>
</table>
This completes the preparation of inputs files. To generate the optimal set of solutions for the optimization problem, DE program is compiled and executed. The program can be compiled and execution using any standard C compiler for Windows environment. After the execution starts, a DOS window pops up (as shown in Figure 2), displaying the progress of the optimization process.

When the program is run for different combinations of $NP$, $CR$ and $F$, the optimal set of parameters is determined based on two factors i.e., minimum objective function value and lower CPU time requirement. In any given situation, if minimum objective function values are the same for any given combination(s), the next criteria that is chosen for selecting optimal combination is lower CPU time requirement. In this program, these two factors are considered for choosing optimal set of parameters. Two output files (“ReportDE.html” and “Convergence.txt”) are generated after the successful completion of the program’s execution.

“ReportDE.html” output file prints the detailed record of the objective function value for all combinations of $NP$, $CR$ and $F$ for all ten strategies. It also records the best ever combination for each strategy along with its objective function value, constraint
violation, number of function evaluations and computational time. The optimal value of objective function and decision variables for the optimization problem is also recorded. The output file for the example is as follows:

## Optimization by Differential Evolution

<table>
<thead>
<tr>
<th>Strategy 1</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>1.776225E-005 (31 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>1.954033E-006 (63 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 2</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>8.589940E-010 (78 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>3.381982E-008 (94 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 3</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>3.777920E-005 (78 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>4.402272E-007 (93 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 4</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>2.198994E-004 (78 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>7.713227E-006 (109 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 5</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>5.074285E-007 (94 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>5.250883E-005 (93 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 6</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>2.001744E-007 (110 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>8.234626E-008 (125 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 7</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>7.588175E-009 (94 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>7.055386E-009 (93 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 8</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>1.362700E-006 (109 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>4.150216E-008 (109 ms)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy 9</th>
<th>NP = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F = 0.50</td>
</tr>
<tr>
<td>CR = 0.80</td>
<td>5.604798E-006 (109 ms)</td>
</tr>
<tr>
<td>CR = 0.90</td>
<td>3.914301E-005 (109 ms)</td>
</tr>
</tbody>
</table>
Strategy 10:
NP = 20

| NP = 20 | CR = 0.80 | 1.259527E-009 (110 ms) | 8.828525E-009 (93 ms) |
| NP = 20 | CR = 0.90 | 1.220839E-006 (94 ms) | 3.677681E-007 (94 ms) |

Results:

<table>
<thead>
<tr>
<th>Strategy No.</th>
<th>Strategy</th>
<th>NP</th>
<th>CR</th>
<th>F</th>
<th>Optimal Value</th>
<th>Constraint Violation</th>
<th>NFE</th>
<th>Time Taken(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DE/rand/1/bin</td>
<td>20</td>
<td>0.90</td>
<td>0.50</td>
<td>1.954033E-006</td>
<td>0.0000E+000</td>
<td>703</td>
<td>63</td>
</tr>
<tr>
<td>2</td>
<td>DE/best/1/bin</td>
<td>20</td>
<td>0.80</td>
<td>0.50</td>
<td>8.589940E-010</td>
<td>0.0000E+000</td>
<td>681</td>
<td>78</td>
</tr>
<tr>
<td>3</td>
<td>DE/best/2/bin</td>
<td>20</td>
<td>0.90</td>
<td>0.50</td>
<td>4.402272E-007</td>
<td>0.0000E+000</td>
<td>681</td>
<td>93</td>
</tr>
<tr>
<td>4</td>
<td>DE/rand/2/bin</td>
<td>20</td>
<td>0.80</td>
<td>0.60</td>
<td>5.341329E-007</td>
<td>0.0000E+000</td>
<td>769</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>DE/rand-to-best/1/bin</td>
<td>20</td>
<td>0.90</td>
<td>0.60</td>
<td>1.778123E-007</td>
<td>0.0000E+000</td>
<td>681</td>
<td>94</td>
</tr>
<tr>
<td>6</td>
<td>DE/rand/1/exp</td>
<td>20</td>
<td>0.90</td>
<td>0.50</td>
<td>8.234626E-008</td>
<td>0.0000E+000</td>
<td>879</td>
<td>125</td>
</tr>
<tr>
<td>7</td>
<td>DE/best/1/exp</td>
<td>20</td>
<td>0.90</td>
<td>0.50</td>
<td>7.055386E-009</td>
<td>0.0000E+000</td>
<td>681</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>DE/best/2/exp</td>
<td>20</td>
<td>0.90</td>
<td>0.50</td>
<td>4.150216E-008</td>
<td>0.0000E+000</td>
<td>725</td>
<td>109</td>
</tr>
<tr>
<td>9</td>
<td>DE/rand/2/exp</td>
<td>20</td>
<td>0.80</td>
<td>0.50</td>
<td>1.954033E-006</td>
<td>0.0000E+000</td>
<td>747</td>
<td>109</td>
</tr>
<tr>
<td>10</td>
<td>DE/rand-to-best/1/exp</td>
<td>20</td>
<td>0.80</td>
<td>0.50</td>
<td>1.259527E-009</td>
<td>0.0000E+000</td>
<td>769</td>
<td>110</td>
</tr>
</tbody>
</table>

Best Strategy is Strategy DE/rand/1/bin

Minimum constraint violation (CV) : 0.0000E+000
Minimum objective value with min CV: 1.954033E-006
Minimum time taken : 63

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>3.000126</td>
</tr>
<tr>
<td>X₂</td>
<td>2.000219</td>
</tr>
</tbody>
</table>

End of report

“Convergence.txt” output file prints the record of the best objective function value for each generation, strategy wise.

The maximum number of generations for the program has been limited to 5000. If the user requires to run the program for more than 5000 generations, a minor modification in the program is necessary. The first line in the main( ) program assigns the variable nx=5000. The value of the variable nx needs to be modified accordingly. Similarly, the value of EPSILON (i.e., the difference in objective function values between two consecutive generations) can also be changed by the user in the program. It can be found in the 17th line of the program as “#define EPSILON 0.000001”. The default value of
EPSILON is set as 0.000001 which is good enough for all kinds of optimization problems.

Thus, one can modify the program to determine the optimal solution for any optimization problem. This source code is distributed for academic purposes only. It has no warranty implied or given, and the author assumes no liability for damage resulting from its use or misuse. Please forward comments or question to the author at vasan.arunachalam@gmail.com.
CHAPTER 4

CONCLUSION

The book explains optimization using the Differential Evolution (DE) algorithm in detail. The DE computer program is available on the accompanying CD. The source code is written in C language in Microsoft Visual C++ 6.0 environment and a basic knowledge of C programming language is sufficient to modify the program. The explanations on how to modify the DE code to use it for various optimization problems are provided. The use of the DE computer program is illustrated using an example problem.


APPENDIX A
SOURCE CODE OF DE ALGORITHM IN C

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <math.h>
double sumarray(double *x, int start, int end);
double evaluateWOOpenalty(double *x);
double evaluate(double *x);
double getDouble(const char *prompt);
double rnd_uni();
int getInt(const char *prompt);
void copyPoint(double *dest, double *source, int D);
double optimise(int gen_max, int D, int NP, double F, double CR, int strategy, double *Point);
int feval;
long timeTaken;
double **x1, **x2, resultpen;
#define EPSILON 0.000001
#define startBold fprintf(fp,"\n");fprintf(fp,"<font face="Arial" size=+1>\n")
#define endBold fprintf(fp,"\n");fprintf(fp,"</font>\n")
#define doubleLine fprintf(fp,"<BR><BR>\n")
#define singleLine fprintf(fp,"<BR>\n")
#define Heading1 fprintf(fp,"<font face="Arial" size=2>\n")
#define endHeading1 fprintf(fp,"</font><BR>\n")
#define Heading2 fprintf(fp,"<font face="Arial" size=2>\n")
#define endHeading2 fprintf(fp,"</font><BR>\n")

/* HTML Report Generation functions */
void startHTMLBODY(FILE *fp)
{
    fprintf(fp,"<HTML><HEAD>\n")
    fprintf(fp,"<style type="text/css">\n")
    .borderTL {
        border-top: 1px solid #cccccc;
        border-right: 1px none #999999;
        border-bottom: 1px none #999999;
        border-left: 1px solid #cccccc;
    }
    .borderBR {
        border-top: 1px none #e9e9e9;
        border-right: 1px solid #cccccc;
    }
}
void closeHTML(FILE *fp)
{
    fprintf(fp,"\n</FONT>\n</BODY>\n</HTML>\n")
    return;
}

int main()
{
    int GENMAX, PARAMS, NP, NPmin, NPmax, sNP, i, j, nx=5000, ny, mx=10, oNP[10],
    oNFE[10], strategy, best=0;
    double CR, F, result, CRmin, CRmax, sCR, Fmin, Fmax, sF, *oCR, *oF, *oResult,
    *wResult, **oPoint, *Point;
    long *oTime;
    char *strategies[14] = {
        "DE/rand/1/bin",
        "DE/best/1/bin",
        "DE/best/2/bin",
        "DE/rand/2/bin",
        "DE/rand-to-best/1/bin",
        "DE/rand/1/exp",
        "DE/best/1/exp",
        "DE/best/2/exp",
        "DE/rand/2/exp",
        "DE/rand-to-best/1/exp"
    };
    FILE *fp = fopen("ParameterLimits.txt", "r");
    fscanf(fp,"%d",&PARAMS);
    fscanf(fp,"%d",&GENMAX);
    fscanf(fp,"%d %d %d", &NPmin, &NPmax, &sNP);
    fscanf(fp,"%lf %lf %lf", &CRmin, &CRmax, &sCR);
    fscanf(fp,"%lf %lf %lf", &Fmin, &Fmax, &sF);
    fclose(fp);
    ny=PARAMS;
    Point = (double *)malloc(sizeof(double)*ny);
    oCR = (double *)malloc(sizeof(double)*mx);
    oF = (double *)malloc(sizeof(double)*mx);
    oResult = (double *)malloc(sizeof(double)*mx);
    wResult = (double *)malloc(sizeof(double)*mx);
    oTime = (long *)malloc(sizeof(long)*mx);
    x1 = (double **) malloc(nx * sizeof(double)*ny);
    x1[0] = (double *) malloc((nx * ny) * sizeof(double));
    for(i = 1; i < nx; i++) x1[i] = x1[i-1] + ny;
    x2 = (double **) malloc(nx * sizeof(double)*ny);
    x2[0] = (double *) malloc((nx * ny) * sizeof(double));
    for(i = 1; i < nx; i++) x2[i] = x2[i-1] + ny;
oPoint = (double **) malloc(mx * sizeof(double*));
oPoint[0] = (double *) malloc(mx * ny) * sizeof(double));
for(i = 1; i < mx; i++) oPoint[i] = oPoint[i-1] + ny;

for(i=0;i<10;i++)
    for(j=0;j<PARAMS;j++)
        oPoint[i][j] = 0;

fp= fopen("Convergence.txt","w");
cfclose(fp);

fp= fopen("ReportDE.html","w");
startHTMLBODY(fp);
for(strategy=1;strategy<=10;strategy++)
{
    printf("Strategy %d\n",strategy);
    Heading1;
    fprintf(fp,"Strategy %d :",strategy);
    endHeading1;
    oResult[strategy-1] = -1;
    for(NP=NPmin;NP<=NPmax;NP+=sNP)
    {
        printf("NP = %d\n",NP);
        Heading2;
        fprintf(fp,"NP = %d",NP);
        endHeading2;
        fprintf(fp,"<TABLE cellpadding="0" cellspacing="0"
class="borderTL">\n");
        fprintf(fp,"<TR><TD class="data1">NP = %d</TD>
",NP);
        for(F=Fmin;F<=Fmax;F+=sF)
        {
            fprintf(fp,"<TD class="data" align="center">F = %0.2lf</TD>
",F);
        }
        fprintf(fp,"<TR></TR>\n");
        for(CR=CRmin;CR<=CRmax;CR+=sCR)
        {
            fprintf(fp,"<TR><TD class="data1" align="center">CR =
%0.2lf</TD><TD align="center">CR = %0.2lf</TD><TD align="center">CR =
%0.2lf</TD></TR>\n",CR);
            for(F=Fmin;F<=Fmax;F+=sF)
            {
                feval = 0;
                result = optimise(GENMAX, PARAMS, NP, F, CR, Point);
                if(oResult[strategy-1]==-1 || oResult[strategy-1]>result)
                {
                    oNFE[ strategy-1 ] = feval;
                    oResult[ strategy-1 ] = result;
                    wResult[ strategy-1 ] = resultpen;
                    oNP[ strategy-1 ] = NP;
                    oF[ strategy-1 ] = F;
                    oCR[ strategy-1 ] = CR;
                }
            }
        }
    }
}

endHTMLBODY(fp);
```c
oTime[strategy-1] = timeTaken;

//Minium constraint violation value
resultpen = wResult[0];
for(i=1;i<10;i++) if(wResult[i]<resultpen) resultpen = wResult[i];

//Best objective function value with minimum constraint violation
i = 0;
while(wResult[i]!=resultpen) i++;
result = oResult[i];
best = i+1;
for(i=best;i<10;i++) if(wResult[i]==resultpen & result<oResult[i]) oResult[i]=result;

//Best strategy with time
best = 0;
timeTaken=-1;
for(i=0;i<10;i++)
    if(wResult[i]==resultpen & oResult[i]==result)
        if(timeTaken==-1 || timeTaken > oTime[i])
            {
                timeTaken = oTime[i];
                best=i;
            }

best++;

//Heading
Heading1;

//End Heading
endHeading1;
```

The code segment above is a part of a program that calculates the optimal strategy based on various parameters such as time taken and objective function values. It involves loops, conditional statements, and array operations to find the best strategy. The code uses printf for output formatting and includes elements of HTML-like syntax for table creation.
double optimise(int gen_max, int D, int NP, double F, double CR, int strategy, double *Point)
{
    int min_cost, count, a, b, c, d, e, i, j, k, imin;
    FILE *fp;
    struct timeb start, end;
    double epsilon;
    count = 0;
    xlow = (double *)malloc(sizeof(double)*D);

    for(strategy=0;strategy<10;strategy++)
    {
        fprintf(fp,"<TR><TD class="data1">%d</TD><TD class="data1">%s</TD><TD class="data1">%d</TD><TD class="data1">%0.2lf</TD><TD class="data1">%0.6lE</TD><TD class="data1">%d</TD><TD class="data1">%ld</TD></TR>");
        fprintf(fp,"<TR class="data1" align="center"><B>Time Taken(ms)</B></TR>");
        for(strategy=0;strategy<10;strategy++)
        {
            fprintf(fp,"<TR><TD class="data1">%d</TD><TD class="data1">%s</TD><TD class="data1">%d</TD><TD class="data1">%0.2lf</TD><TD class="data1">%0.6lE</TD><TD class="data1">%d</TD><TD class="data1">%ld</TD></TR>");
        }
        fprintf(fp,"</TABLE>");

        fprintf(fp,"<font size="1">End of report</font>" );
        fclose(fp);
        return 0;
    }

    double optimise(int gen_max, int D, int NP, double F, double CR, int strategy, double *Point)
xhigh = (double *)malloc(sizeof(double)*D);
best = (double *)malloc(sizeof(double)*D);
bestit = (double *)malloc(sizeof(double)*D);
trial = (double *)malloc(sizeof(double)*D);
cost = (double *)malloc(sizeof(double)*NP);

fp = fopen("limits.txt","r");
for(i=0;i<D;i++)
    fscanf(fp,"%lf %lf",&xlow[i],&xhigh[i]);
fclose(fp);
ftime(&start);
for(i=0;i<NP;i++)
{
    for(j=0;j<D;j++)
        x1[i][j] = xlow[j]+rnd_uni()*(xhigh[j]-xlow[j]);
    cost[i] = evaluate(x1[i]);
}
cmin = cost[0];
imin = 0;
for (i=1; i<NP; i++)
{
    if (cost[i]<cmin)
    {
        cmin = cost[i];
        imin = i;
    }
}
copyPoint(best,x1[imin],D); /*save best member ever          */
copyPoint(bestit,x1[imin],D);   /*save best member of generation */

fp = fopen("Convergence.txt","a");
fprintf(fp,"\n\nStrategy %d (NP=%d; CR=%0.2lf; F=%0.2lf)\n",strategy, NP, CR, F);
fclose(fp);
while(!(fabs(epsilon)<EPSILON && count>=gen_max))
{
    for(i=0;i<NP;i++)
    {
        do a=(int)(rnd_uni()*NP); while(a==i);
        do b=(int)(rnd_uni()*NP); while(b==i || b==a);
        do c=(int)(rnd_uni()*NP); while(c==i || c==b || c==a);
        do d=(int)(rnd_uni()*NP); while(d==i || d==c || d==b ||d==a);
        do e=(int)(rnd_uni()*NP); while(e==i || e==d || e==c ||e==b || e==a);

       ="/******************************** DE/rand/1/bin  "*/
        if(strategy == 1)
        {
            j = (int)(rnd_uni()*D);
for(k=1;k<=D;k++)
{
   if(rnd_uni()<CR || k==D)
      trial[j] = x1[c][j]+F*(x1[a][j]-x1[b][j]);
   else
      trial[j] = x1[i][j];
   if(trial[j]>xhigh[j] || trial[j]<xlow[j])
      trial[j] = x1[i][j];
   j= (j+1)%D;
}
/*--------------- DE/best/1/bin ---------------------*/
else if(strategy == 2)
{
   j= (int)(rnd_uni()*D);
   for(k=1;k<=D;k++)
   {
      if(rnd_uni()<CR || k==D)
         trial[j] = bestit[j]+F*(x1[a][j]-x1[b][j]);
      else
         trial[j] = x1[i][j];
      if(trial[j]>xhigh[j] || trial[j]<xlow[j])
         trial[j] = x1[i][j];
      j= (j+1)%D;
   }
}/*---------------- DE/best/2/bin ---------------------*/
else if(strategy == 3)
{
   j= (int)(rnd_uni()*D);
   for(k=1;k<=D;k++)
   {
      if(rnd_uni()<CR || k==D)
         trial[j] = bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-
            x1[d][j]);
      else
         trial[j] = x1[i][j];
      if(trial[j]>xhigh[j] || trial[j]<xlow[j])
         trial[j] = x1[i][j];
      j= (j+1)%D;
   }
}/*---------------- DE/rand/2/bin ---------------------*/
else if(strategy == 4)
{
   j= (int)(rnd_uni()*D);
   for(k=1;k<=D;k++)
   {

if(rnd_uni()<CR || k==D)
    trial[j] = x1[e][j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-
    x1[d][j]);
else
    trial[j] = x1[i][j];
if(trial[j]>xhigh[j] || trial[j]<xlow[j])
    trial[j] = x1[i][j];
    j= (j+1)%D;
}
/*--------------- DE/rand-to-best/1/bin ---------------------*/
else if(strategy == 5)
{
    copyPoint(trial,x1[i],D);
    j= (int)(rnd_uni()*D);
    for(k=1;k<=D;k++)
    {
        if(rnd_uni()<CR || k==D)
            trial[j] = trial[j]+F*(bestit[j]-
            trial[j])+F*(x1[a][j]-x1[b][j]);
        else
            trial[j] = x1[i][j];
        if(trial[j]>xhigh[j] || trial[j]<xlow[j])
            trial[j] = x1[i][j];
        j= (j+1)%D;
    }
} /*--------------- DE/rand/1/exp ---------------------*/
else if(strategy == 6)
{
    copyPoint(trial,x1[i],D);
    j= (int)(rnd_uni()*D);
    k=0;
    do
    {
        trial[j] = x1[c][j]+F*(x1[a][j]-x1[b][j]);
        if(trial[j]>xhigh[j] || trial[j]<xlow[j])
            trial[j] = x1[i][j];
        j= (j+1)%D;
        k++;
    }while(rnd_uni()<CR && k<D);
}
/*--------------- DE/best/1/exp ---------------------*/
else if(strategy == 7)
{

copyPoint(trial,x1[i],D);
  j= (int)(rnd_uni()*D);
  k=0;
  do
  {
    trial[j] = bestit[j]+F*(x1[a][j]-x1[b][j]);
    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
      trial[j] = x1[i][j];
    j= (j+1)%D;
    k++;
  }while(rnd_uni()<CR && k<D);
}

/*--------------- DE/best/2/exp ---------------------*/
else if(strategy == 8)
{
  copyPoint(trial,x1[i],D);
  j= (int)(rnd_uni()*D);
  k=0;
  do
  {
    trial[j] = bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
      trial[j] = x1[i][j];
    j= (j+1)%D;
    k++;
  }while(rnd_uni()<CR && k<D);
}

/*--------------- DE/rand/2/exp ---------------------*/
else if(strategy == 9)
{
  copyPoint(trial,x1[i],D);
  j= (int)(rnd_uni()*D);
  k=0;
  do
  {
    trial[j] = x1[e][j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
      trial[j] = x1[i][j];
    j= (j+1)%D;
    k++;
  }while(rnd_uni()<CR && k<D);
}

/*--------------- DE/rand-to-best/1/exp ---------------------*/
else if(strategy == 10)
{

```c
copyPoint(trial,x1[i],D);
j= (int)(rnd_uni()*D);
k=0;
do
{
    trial[j] = trial[j]+F*(bestit[j]-trial[j])+F*(x1[a][j]-
x1[b][j]);
    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
        trial[j] = x1[i][j];
    j= (j+1)%D;
k++;
} while(rnd_uni()<CR && k<D);

score = evaluate(trial);
if(score<=cost[i])
{
    copyPoint(x2[i],trial,D);
cost[i] = score;
}
else
    copyPoint(x2[i],x1[i],D);

for(i=0;i<NP;i++)
copyPoint(x1[i],x2[i],D);

imin = 0;
for (i=1; i<NP; i++)
{
    if (cost[i]<cmin)
    {
        cmin = cost[i];
imin = i;
    }
} count++;
epsilon = evaluate(bestit);
fp = fopen("Convergence.txt","a");
fprintf(fp,"%d\%0.6le\n", count, epsilon);
fclose(fp);
copyPoint(bestit,x1[imin],D);        /* save best member of generation */
epsilon -= evaluate(bestit);  /* Determines the epsilon value */
}
min_cost = 0;
for(i=1;i<NP;i++) if(cost[i] < cost[min_cost]) min_cost = i;
copyPoint(Point,x1[min_cost],D);
result = evaluateWOpenalty(Point);
resultpen = (fabs(fabs(result)-fabs(evaluate(Point))));
ftime(&end);
```
timeTaken= (end.time-start.time)*1000;
timeTaken+= (end.millitm-start.millitm);
return result;
}

double sumarray(double *x,int start,int end)
{
    int i;
    double result=0;
    for(i=start;i<=end;i++)
        result+= x[i];
    return result;
}

int getInt(const char *prompt)
{
    int result;
    printf(prompt);
    scanf("%d",&result);
    return result;
}

double getDouble(const char *prompt)
{
    double result;
    printf(prompt);
    scanf("%lf",&result);
    return result;
}

double rnd_uni()
{
    int r;
    r= rand();
    return (double)(r)/RAND_MAX;
}

void copyPoint(double *dest,double *source,int D)
{
    int i;
    for(i=0;i<D;i++)
        dest[i] = source[i];
    return;
}

double evaluateWOpenalty(double *x)
{
    double value;
double evaluate(double *x)
{
    int ncons, i;
    double *val, *penal, temp;
    long double value=0;
    feval++;

    /* DO NOT CHANGE ANYTHING ABOVE */
    /* Enter the Number of Constraints to the variable ncons*/
    ncons = 2;

    /* CODE YOUR OBJECTIVE FUNCTIONS HERE*/
    val = (double *)malloc(sizeof(double)*ncons);
    penal = (double *)malloc(sizeof(double)*ncons);
    penal[0]=1;

    /* CODE YOUR OBJECTIVE FUNCTIONS HERE*/
    /* All functions must be of minimization type*/
    /*Example for Objective Function*/
    val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) + (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7);

    return value;
}
/*============Start Coding Your Objective Function============*/
val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) + (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7);
/*==========END YOUR CODING UPTO THIS POINT============*/
/*----------------------CODE YOUR CONSTRAINTS HERE------------*/
/*All functions must be of minimization type, negate maximization functions */
/*Constraints must be of the following type: g(x) <= 0. Enter the constraint equation to temp variable and don't change the expression for val[1]. Similarly, do the same for other constraints. See the example below.*/

/*Example for Objective Function*/

temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
val[1] = max(temp,0);

If the constraint of the example is an equality constraint, the constraint should be entered as follows:

val[1] = abs((x[0]-5)*(x[0]-5) + x[1]*x[1] - 26);

/*============Start Coding Your Constraints============*/
/* Constraint 1 */
temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
val[1] = max(temp,0);

/*Constraint 2*/
temp = 4*x[0] + x[1] - 20;
val[2] = max(temp,0);

/*Enter the Penalty Coefficient for Each constraint*/

penal[1] = 1000;
penal[2] = 1000;

/*==========END YOUR CODING UPTO THIS POINT============*/
for (i=0;i<=ncons;i++)
    value = value + penal[i]*val[i];
return value;
}
APPENDIX B
PREVIOUS REPORTS IN THE SERIES

ISSN: (print) 1913-3200; (online) 1913-3219


dynamics model for analyzing behaviour of the social-economic-climatic system:
Model description and model use guide. Water Resources Research Report no. 059,
Facility for Intelligent Decision Support, Department of Civil and Environmental
(online) 978-0-7714-2680-3.