Comparison of Derivative Free Algorithms in Optimization High Dimension Problems

by

Jianliang Hao

A master thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
Dec 12, 2015

Keywords: Gradient free, particle swarm algorithm, genetic algorithm, Nelder-Mead simplex, Nelder-Mead simplex Plus, quasi Gradient simplex, artificial neural network

Approved by

Bogdan M. Wilamowski, Chair, Professor of Electrical and Computer Engineering
John Hung, Professor of Electrical and Computer Engineering
Hulya Kirkici, Professor of Electrical and Computer Engineering

Abstract

Gradient free optimization algorithms are often used when gradient information of a function is not available or it is difficult and costly to obtain or estimate. Many kinds of derivative free algorithms are used in the fields of statistics, and engineering. This paper compared five derivative-free optimization algorithms using a set of benchmark functions. Then the comparison results will be reported and discussed. The employed algorithms include Genetic algorithm, Particle Swarm Optimization, Nelder-Mead simplex and Nelder-Mead simplex Plus and quasi Gradient method. This master thesis (1) reviews how each optimization method works in detail; (2) tests the quality of each algorithm in optimizing high dimensional problems; (3) discuss different optimization method's performance in optimization;(4) choose top performing optimization method solve particular artificial neural networks problems, which is intended to explore the potential of combining derivative free method with artificial neural networks.

Acknowledgments

First of all, I would like to express my thanks to Auburn University and the ECE department for giving me the good environment to study. Secondly, I would like to express my thanks to those professors who guide me in class. Most importantly, I want to express my sincere appreciation to my advisor, Professor B. M. Wilamowski. Without his patience and guidance I would not finish my graduate research. From him I have learned a lot, his perseverance in academic research affects me profoundly during my study here.

I also want to express my thanks to my parents and my sister, who always give me support whenever I face difficulties. I also would like to thank all my friends, who are willing to share happiness and unhappiness with me and help me in my oversea life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Optimization plays an important role in many fields, such as industry, finance and engineering. It helps factories and financial institutions save energy, improve product quality, and reduce operation cost. Excellent optimization methods can increase the profits of companies from a variety of aspects. As the number of application in different field increases, much research about optimization methods has been conducted. A significant portion of this research is based on the assumption that the objective function of nonlinear programming problems is approximately or sufficiently smooth. Theoretical conditions for optimization of these problems are in relation to the gradient and Hessian of the objective function. From mathematical knowledge, the gradient and Hessian of the objective functions includes useful information for optimization. The mathematical characteristic of the minimum or maximum requires that the first-order gradients are zero for a continuously differentiable function and then we can get the minimum and maximum point. With the availability of the gradient, gradient-based methods, including first order gradient method, Newton's method [1], as well as quasi-Newton methods [2], are widely used both in theoretical analysis and practical applications. The derivative can be obtained through analytical formulas by computing the differentiation.

However, there are many cases where derivative information is not available or not reliable. Frequently the objective function is simply not differentiable or computing the derivative of the function is infeasible even it is differentiable. In some cases, it is time-consuming to calculate the derivative when the objective function has many variables. In other cases, the execution of the objective function is disturbed by noise, and computing the objective function's derivative is not

reliable for converging to minimum. Under such circumstances, one still has a desire to conduct optimization. As a result, a set of nonlinear optimization methods, called derivative-free methods, are needed.

In fact, derivative free optimization is a vital and challenging field in engineering. There are an increasing number of optimization problems defined by objective functions for which derivative are not accessible or only accessible at a high computational cost. For instance, if the objective function is computed using a black-box simulation module, it is impossible to get derivative with automatic differentiation. Another reason why derivative free algorithms are needed is the increasing complexity in mathematical modeling and higher difficulty of scientific computing. Since the derivative of the objective function, which contains useful information, is missing, derivative-free methods attempt to solve the problems of optimization using only the objective function.

Before we use the derivative-free methods, there are some concepts related to optimization that need to be clarified. Optimization problems can be classified in different ways. One classification is about constrained and unconstrained optimization. In constrained optimization problems, there are some limitations on the values of input variables or the value of objective function. In contrast, unconstrained optimization problems have no limitation on the input variables, or the constraints of variables have no effect on the solution. Another way to classify the optimization problems is stochastic or deterministic optimization. Stochastic optimization uses random variables for optimization problems, which may not be defined and may contain random constraints. In some cases, stochastic optimization adopts random iterates solving stochastic problems [3]. In general, stochastic optimization methods includes evolutionary algorithm, swarm algorithms, and simulated annealing by S. Kirkpatrick [4]. Deterministic

optimization assumes the module of optimization problems is specified in certain domain space. Optimization methods can be classified with global and local optimization. Global optimization can search a large solution domain, and can get out of some local minimums, and converge to global minimum compared with local optimization method. In contrast, the local optimization can converge faster than global minimum.

Although there are many kinds of classification, good optimization methods should have some properties in common: robustness, efficiency, and exactness. The robustness means that the optimization method is proper to optimize a large set of problems instead of very particular problems. Efficiency means the optimization method can converge to the required minimum in certain computing time and memory storage. Exactness means that the method can obtain the solution of the problems correctly and not be sensitive to noise.

In this master thesis, the focus is to solve unconstrained minimization problems represent as a set of benchmark functions with five derivative free algorithms. Since function minimization is widely used in sciences with applications in many real life models.

Two heuristic methods are adopted, genetic algorithm, and particle swarm optimization method. These two methods are widely applied in many fields. GA resembles the natural behavior of genetic heritage, optimizing problems with recombination, mutation and reinsertion operations. In contrast, PSO, presented by Kenndy and Eberhart [5], works by the interaction of individuals among the whole population. Each individual updates their values by its previous performance and by the previous best performance among the whole population each time step. So far, PSO and GA have been widely used to optimize many problems. Nelder-Mead simplex method and its two improvement by Pham and Bogdan. M. Wilamowski [6] are compared with two heuristic

methods. Nelder-Mead simplex, proposed by Nelder and Mead [7], is favored by scientists and engineers owing to its simplicity and convergence rate even without using gradient information. Nelder-Mead Plus algorithm and Quasi Gradient Simplex algorithm are two improvements of Nelder-Mead Simplex by improving the success rate in optimizing high dimensional problems.

The rest of the thesis is outlines as follows.

In the chapter 2, review the fundamentals of the five optimization methods, genetic algorithm (GA), Particle Swarm Optimization method, Nelder-Mead simplex method and Nelder-Mead simplex Plus and quasi Gradient simplex method.

In the chapter 3, conduct the experiment to test the robustness, efficiency and exactness, of each algorithm by a set of benchmark functions. Further explore the characteristics of top performed algorithms.

In the chapter 4, use top performed algorithms train real life problems, artificial neural network weight.

In the chapter 5, major results of the thesis are summarized and further research is discussed.

Chapter 2

Review of Tested Algorithm

This chapter reviews the background information of five derivative-free algorithms one by one and illustrates how they work.

2.1 Genetic Algorithm

Genetic Algorithm [8] is to follow the processes observed in natural evolution according to Charles Darwin's 'survival of the fittest' theory. Because of the fact that many individuals will compete with each other for rare resources in nature, which lead to the fittest individuals survived. GA has been widely used to generate the solution of optimization and search problems in many fields since its development, as it can be implemented easily without calculating its derivative information. GA performs well in both global and local search at the same time and is capable to find the optimum solution without getting stuck in local minima because it works on the encoding of the parameter set rather than the parameter itself. This section will introduce the implementation details of GA and describe why GA can be acted as a global search method.

Generally, GA consists of list steps:

2.1.1 Population Representation and Initialization

GA begins with a population of potential solutions, called initial population, then applies the principle of survival of the fittest to approach to an optimal solution. The initial population is made up of encoding of the parameter set. The commonly used representation of chromosomes for GA is single-level binary string, which is used to encode every variable, then concatenating

all encoded variables to form a chromosome [9]. In the conversion of binary-coded chromosome to their corresponding real values, Gray coding is used because some reports shows this method can overcome the representational bias in other binary representation as the Hamming distance between adjacent values is constant, which can efficiently locate the global minima [10]. After determining the representation, random number generator is used to generate the required number of individuals which are uniformly distributed in the setting range. For instance, a binary population of n individuals has chromosomes with l bit long, then $n \times l$ random numbers will be generated uniformly from the set $\{0\ 1\}$. Each individual is encoded as a string, called chromosome. The commonly used representation for GA is binary alphabet $\{0\ 1\}$. For instance, a problem with 4 variables, $\{x1\ x2\ x3\ x4\}$, can be represented to the chromosome structure in the following way:

$$\begin{Bmatrix} x_1 & x_2 & x_3 & x_4 \\ 01100 & 00100 & 0111011111 & 1001111111 \end{Bmatrix}$$

Where $\{x1\ x2\}$ are encoded with 5 bits and $\{x3\ x4\}$ with 10 bits, the different length of encoding reflects the different accuracy levels which is problem dependent.

2.1.2 Evaluation Functions

Evaluation Functions include objective function and fitness function. The objective function measures the performance of each individual in the problem domain. For example, the fittest individuals should have the lowest numerical value of the associated objective function in a minimization problem [9]. However, the objective function cannot evaluate the relative performance among all individuals, that is why fitness function needed. Fitness function acts as a

tool that converts the objective function value into a measure of relative fitness [6], and these relative fitness values will be used to select individuals for future reproduction, thus:

$$F(x) = g(f(x)) \tag{2.1}$$

In this equation, f is the objective function, g converts the value of the objective function $f(x)$ to a non-negative number and F is the relative fitness value. This conversion is essential as the lower objective function value corresponds to better individuals in minimization problems.

2.1.3 Selection

Selection is the process of stochastically choosing a fraction of the existing population to breed a new generation. The selection process is based on the fitness value discussed in the previous section, which requires those who have higher fitness values to be selected with more probability and survive to form the mating pool for next generation. For weaker ones, they still have chance but not that large. There are several selection methods used to rank the fitness of each solution and select the best solutions. The popular methods is Roulette Wheel Selection Method [11]. Table 1 lists a sample population of 8 individuals with objective function (2.2), in this table, the objective function value and fitness function value are calculated by (2.1) and (2.2).

$$F_2 = \sum_{i=1}^{n} (x_i - a_i)^2 \quad \text{,where } a_i = i \tag{2.2}$$

Table 1: Population selection based on converted value

No.	Chromosome	Variable value	Objective function value	Fitness function value	Fraction of total
1	0101100001	-1.2805	5.2009	1.6	0.2667
2	1111011011	2.8641	3.4750	2.0	0.3333

3	0101101101	-1.4370	5.9387	1.2	0.2000
4	0011100111	-6.3636	54.2231	0	0
5	0101110100	-1.7302	7.4540	0.8	0.1333
6	0010000100	-5.1515	37.8411	0.4	0.0667

The binary population in table 1 represents a set of variables in the range of $[-10\ 10]$. The Gray code binary representation will be converted to real value using arithmetic scaling. The objective values are taken as the function of real-valued phenotypes, and then convert the objective values into fitness function values. One can get the fraction of total by summing these fitness values over (sum) all the individuals in the population. From table 1, one may notice that individual No.2 is the fittest and occupies the largest interval, whereas No.4 is the weakest and have relatively smaller interval within the roulette wheel. The percentage fitness value will be used to select an individual for next generation. A random number will be generated in the interval $[0, sum]$ and the individuals whose segment spans the random number is selected [8]. The number of times the roulette wheel is spun equals to the desired number of individuals.

2.1.4 Recombination

Crossover is used to generate the new chromosomes. Like the crossover in nature, crossover breeds new individuals that bring many characteristics of their parents. In this operation, the parents are selected from the mating pool previous chosen by fitness value. And this operation will be repeated until the required number of individuals is generated. There are many crossover forms. In this section, the single-point crossover and multi-point crossover are described and discussed.

2.1.4.1 Single-point crossover

Consider the two parent binary strings:

$$\begin{aligned} Parent1 &: 01100001 \\ Parent2 &: 01011111 \end{aligned} \tag{2.3}$$

For single point crossover, one integer will be chosen uniformly at random between 1 and the length of the string minus one $[1, l-1]$ for crossover position, then, from this point, the genetic information is exchanged between these two parents. Two new offspring then are produced. In the following example, the crossover point is $i = 6$, thus

$$\begin{aligned} & 12345678 \\ Offspring1 &: 01100011 \\ Offspring2 &: 01011101 \end{aligned} \tag{2.4}$$

However, this crossover operation is only performed based on a probability px which is the probability to choose the pairs for breeding, not all strings.

2.1.4.2 Multi-point crossover

For multi-point crossover operation, m integers will be chosen uniformly at random between 1 and the length of the chromosome $[1, l-1]$ without duplicates and sorted into ascending order [8]. Then, the information between these two successive crossover points is exchanged to generate new offspring. The part between the first allele position and the first crossover point is not exchanged between two parents. Use (2.3) as parent chromosomes. The operation is demonstrated in (2.5).

Offspring generated by multi-point crossover operation:

$$\begin{aligned} Offspring1 &: 01000011 \\ Offspring2 &: 01111101 \end{aligned} \tag{2.5}$$

This process produces the new generation population different with the previous population. Just like the single-point crossover operation, multi-point crossover also is based on probability px. It is reported that this disruptive nature of multi-points crossover tends to produce some new offspring that do not converge to higher fit individuals early in the search domain [12]. However, newly generated offspring can explore more search space and may prevent from premature convergence and make the search more robust [13]. The idea behind multi-point crossover is that the parts of gene representation that contribute most to the performance of a particular individual may not necessarily be confined in adjacent substrings [11].

These two crossover operation methods can be implemented on any chromosome representation.

2.1.5 Mutation

Mutation process is intended to introduce diversity from one generation to the next generation. This diversity can increase the population's ability to search larger problem space and constrain the probability of converging to local minimum. Mutation operation is randomly attained with low probability, typically in the range of 0.001 and 0.01 [8].

A common method used in mutation is generating a random variable for each bit in a sequence and this variable decides which bit should be modified. The mutation operation is similar to single-point crossover, the difference is it only change one bit. The table 2 shows the effect of mutation on a 8-bit chromosome which represents a real value decoded over the interval [-10 10] using both standard and Gray coding. The mutation position is 6 in this binary string.

Table 2 binary mutation

Chromsomes	Gray to real value	Binary to real value
original string : **1 1 0 1 0 1 1 1**	2.0784	6.8627
mutated string : **1 1 0 1 0 0 1 1**	2.3137	6.5490

The red bit shows the mutation position. Given that mutation is applied uniformly at random to the entire population of chromosome, it is entirely possible that one certain string may be mutated more than one time. Indeed, Tate and Smith [14] argue that high mutation rate can generate better solutions than the normal one in optimization problem.

2.1.6 Reinsertion and elitist GA

After the process listed above, selection, recombination and mutation, a new population has been produced from the old population and the corresponding fitness values of this new population may be determined. If the number of the new population is less than the original population, then the proportion between the new population and old population is called generation gap [15]. In this case, if some of the individuals from the old population is allowed to carry over to the new generation without changes [16]. This strategy is called, *elitist strategy* [17], which ensures that the number of new population is the same as the old population and the new population quality will not be worse than the new generation.

In the elitist strategy, some individuals from old population will be reinserted to the new population, replacing parents with offspring and returning the resulting population. When deciding which individuals of the old population should be replaced by the new generation individuals, the common strategy is to replace the worse or less fit members. The selection method for replacing parents with offspring is uniform selection, in which offspring replace parents uniformly at random, or fitness-based selection, in which offspring replace less fit

parents, discussed in the evaluation part. Thus, the elitist strategy may make some individuals survive through successive generations if these individuals are sufficiently fit.

The above six parts described how the general GA works, including elitist GA. These procedures are repeated until the best chromosome close to the desired global minimum.

2.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is introduced and developed by James Kennedy [18] in 1995. This method simulates the behavior of bird or fish school finding food or prey, and is computationally effective with respect to the convergence speed [17]. Besides, the algorithm is very compact and can be implemented with few lines of Matlab code. This algorithm has been proved effective in solving many problems [19][20][21][22].

In PSO a number of particles are randomly generated and located in the search domain of the to-be-solved function or problems and each particle, representing a potential solution to the problem, assesses the objective function value at its current position. These particles have almost no power to solve any problem if they did not interact with each other [19]. Each particle communicates with other particles and is influenced by the best point found by other members, if one particle finds a greater region in the search domain, which will be denoted as Pg, all other members of the swarm are attracted to this domain. Then, each particle adjusts its movement through search domain by incorporating the information of the history of its own best and current locations with the information of other particles. The next iteration occurs after all particles have been moved. This process is repeated iteratively until the stopping criteria reached. This kind of communication mechanism contributes to the quick convergence to global minimum.

Compared with Genetic Algorithm, PSO is very compact, as it does not have complex operations, such as chromosome crossover, mutation and reinsertion. That can explain why PSO has gained popularity in different fields these years. The following sections describe the details of the PSO process.

2.2.1 Initialization

Particle swarm algorithm initializes a population array of particles at random uniformly within bounds. Each individual in the swarm is composed of n dimensional row vector, where n is the number of the variables. For example, particle i has position $x(i)$, which is a row vector with the number of variable elements. The spread of the initial swarm is controlled using the initial bound. The range of component i is the $(ub(i) - lb(i))$, where ub and lb refer to upper and lower bound respectively. Similarly, initial particle velocities $v(i)$ are created in a uniform, random distribution within the range $[-v, v]$, where v is the vector of initial ranges.

For each particle, assess its objective function value. The memory records the current position $x(i)$ of every particle i. In following iterations, $x(i)$ $(pbest)$ will be the best location of each particle found based on the objective function and g is the best objective function value over all particle with $g = \min(func(x(i)))$. The coordinate of fittest function value is denoted as $gbest$.

2.2.2 The main step in the particle swarm algorithm is the generation of new velocities and new position for the swarm of particles, this is achieved by the equations (2.6) and (2.7)

$$v(i+1) = W \times v(i) + c_1 \times u_1 \times \left(p(i) - x(i)\right) + c_2 \times u_2 \times \left(gbest(i) - x(i)\right) \tag{2.6}$$

$$x(i+1) = x(i) + v(i+1) \tag{2.7}$$

Where $W = inertia\ weight,$ proposed by Shi and Eberhart [23],which proved to play a vital role in balancing the exploration and exploitation process of swarm. $c_1 = self\ adjustment$ used to change velocity based on the history of particle itself. $c_2 = socia\ adjustment$ used to adjust location towards best particle coordinate. u_1, u_2 generated at random in the range of [0 1]. $v(i)$ and $x(i)$ are the previous velocity and position respectively. Much research has been done on the parameter setting of the PSO algorithm [22][24]. The inertial weight is commonly either taken as a linearly decreasing function in iteration from 0.9 to 0.4 or as a constant. The linear inertial function will make the effect of past velocity become smaller as the increase of iteration. Two acceleration constants $c_1 c_2$ are most commonly both set to equal 2 according to [22].

2.2.3 Boundary control

During the process described in 2.2.2, boundary control is always indispensable. If any component of x is out of the setting boundary, it is replaced with a randomly generated number or set to the boundary value. The former has been proved to be effective that the latter one in this thesis.

2.3 Nelder-Mead simplex

Nelder-Mead simplex method, put forward by J.A. Nelder and R.Mead in 1965 [7], is a direct search algorithm and can be used for multidimensional optimization, as proven in the original paper [7]. Unlike other nature inspired algorithm, it does not work like a flock of birds foraging for food or genetic heritage. Instead, it generates a geometric simplex and uses this simplex's

movement to guide its convergence [6]. The geometric simplex is composed by $(n+1)$ vertices for functions with n variables. The function value of each vertex in the simplex will be evaluated iteratively and ranked in ascending order, and the worst vertex with the highest function value is excluded and replaced by a newly generated vertex if the function value of new vertex is smaller than the worst vertex. A new simplex is formed and the search will continue iteratively. The function value of the simplex's vertices tends to be smaller and smaller as a set of new vertices is created. As a result, the size of the simplex is reduced and the location of the minimum point is found.

The Nelder-Mead simplex algorithm is simple and compact [7]. Its convergence speed is affected by three parameters α,β,γ, where α is the reflection coefficient, β is the contraction coefficient, γ is the expansion coefficient. It has been proved that α=1, β=1/2, γ=2 perform best in optimization [7]. Therefore, the shape of the simplex is changed based on these three coefficient values. The following parts will illustrate Nelder-Mead simplex through a two-dimensional case (n=2).

2.3.1 Initialization

For the minimization of a function with 2 variables, 3 vertex points are generated to form an initial simplex. The objective function determines $B=(x_1,y_1), G=(x_2,y_2), W=(x_3,y_3)$ vertices, representing vertex points with best, good, worse function values, which then form simplex ΔBGW. The center M of line segment BG of simplex BGW is obtained, refer to Figure 2.2 (a)

$$M=\frac{B+G}{N}=\frac{B+G}{2} \tag{2.8}$$

2.3.2 Reflection

A new vertex R is generated by reflecting the worst point. The idea is that the function value reduced when move along the side of the triangle from W to B and from W to G. So it is feasible that the function will have a smaller value at the testing point R. To determine the location of R, draw a line from midpoint M to W and define its length d. and extend the same distance d in the same direction $\overrightarrow{WM}$. See Figure 2.2 (b)

$$R = (1+\alpha)M - \alpha W = 2M - W \tag{2.9}$$

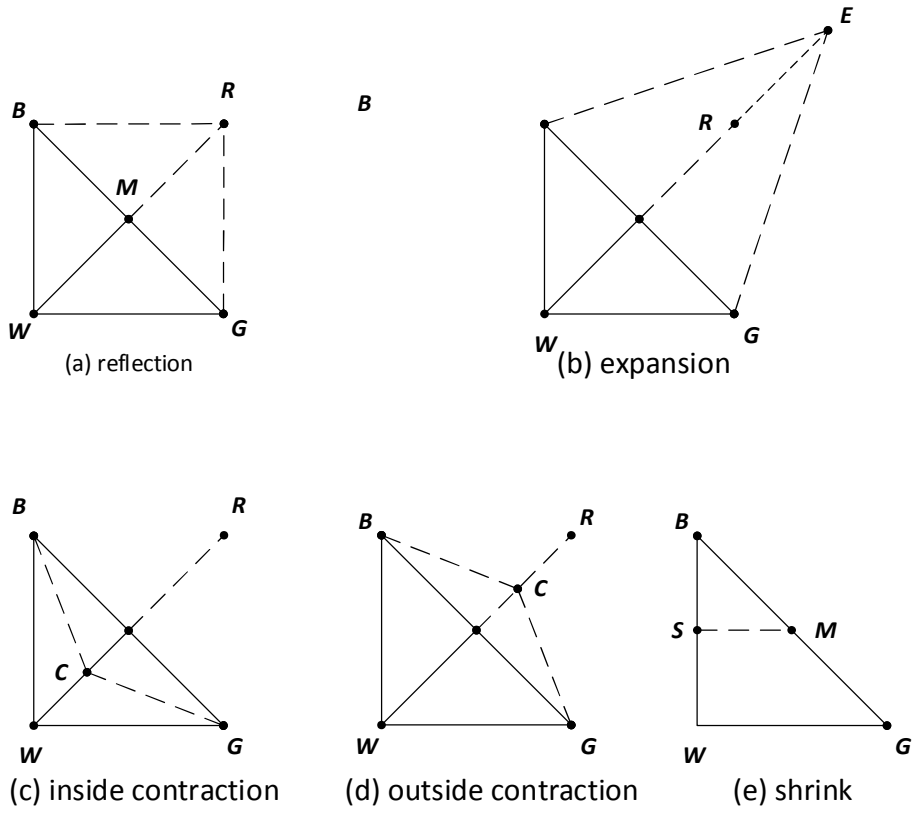


Figure 2.2 Nelder-Mead Simplex operation. The original simplex is shown with solid line

2.3.3 Expansion

If $f_R < f_w$, where f_R and f_w represent reflection and worse value respectively, then we moved in the proper direction toward the minimum. Extend the line segment through M and R to the point E with distance d using equation (2.10), based on the guess that the minimum is just a little farther than the point R. If the function value at E is smaller than function value at R, then a better vertex is reached and new simplex ΔBGE is formed.

$$E = \gamma R + (1-\gamma)M = 2R - M \tag{2.10}$$

2.3.4 Contraction

If $f_R > f_w$, another point has to be tested. Two midpoints C_1 and C_2 of the line segment $\overrightarrow{WM}$ and $\overrightarrow{MR}$ are considered and the points with the minimum function value between C_1 and C_2 is termed as C. If $f_c \leq f_w$, W is replaced with C (2.11) and a new simplex ΔBGC is formed.

$$C = \beta W + (1-\beta) \times M \tag{2.11}$$

Where β represent contraction coefficient, suggested β =0.5 by Nelder and Mead [10].

2.3.5 Shrink

If $f_c > f_w$, where f_c represents contraction value, shrinkage will be attempted. The points G and W will shrink toward B using equation (2.12). The point G is replace with M, and W is replaced with S, which is the midpoint of the line segment $\overrightarrow{BW}$.

$$P_i = \delta P_i + (1-\delta)B \text{ ,where } P_i = G \,\&\, W \tag{2.12}$$

Where δ is the shrinkage coefficient, suggested δ=0.5 by Nelder and Mead [7]. The process continues and repeats to generate a sequence of triangles that converge down on the solution point.

The simplex method is a compact algorithm. It is simple with computational process and is neither based on gradients nor on quadratic forms [7]. This algorithm uses less information at each stage and need no information of past positions. However, just like other optimization methods, the Nelder-Mead simplex may fail to converge to global minimum in some problems. This false convergence has been found in using the simplex method on a four-dimensional problems, may tend to be worse in higher dimensional problems. Because of this deficiency, Pham and Bogdan. M. Wilamowski proposed two improved simplex methods to improve the convergence success rate and speed [6].

2.4 Nelder-Mead's Simplex Plus Method

As we described above, the original simplex method is a fast and simple algorithm, it is based on the formation of geometric simplex and its movement to find minimum. But its false convergence in high dimensional problems limits its application. Indeed, many researches have been done in improving its convergence speed and success rate in high dimensional optimization problems. Fuchang Zhao and Lixing Han proposed an implementation of simplex method which the dimension of the problem determines the parameters of expansion, contraction and shrinking [25]. But this implementation will make the simplex method complicated. In order to maintain the simplex simplicity and make it robust and reliable in high dimensional optimization at the same time, Pham and Bogdan. M. Wilamowski proposed that one can approximate better

vertices along the gradient direction to form new simplex instead of calculating the reflected vertex as the original simplex did [6].

Nam Pham used two quasi gradient methods, which used numerical instead of analytical analysis, to approximate gradients. It is reported that the improved algorithm can converge faster than original one [6]. The following part still uses two dimension case to illustrate this reasoning. Assuming the three vertices $B=(x_1,y_1), G=(x_2,y_2), W=(x_3,y_3)$ have totally different gradient respectively. In the case (a) Fig. 2.1a the function value at W and G are similar while in the case (b) Fig. 2.1b the function values at B and G are similar. In these two cases, the gradient of the plane directs to different directions from the simplex method. Depending on the original simplex method, the simplex BGW starts to get the reflected point in $\overrightarrow{MR}$ direction. If it cannot get a better vertex in this direction, W and G start to shrink toward B , and new simplex $BG'W'$ is formed. The simplex $BG'W'$ continues search in the same direction, which has been proved to be the wrong direction to the minima. This illustration shows why the original simplex method cannot search for the correct direction to minima based only on the geometrical movement and explains why this algorithm is easy to false convergence in multi-dimensional problems and to converge relatively slow. After detecting the bottleneck of the original simplex, Pham adopts the gradient method witch directs where new reflected points should be generated.

Nelder-Mead Simplex Plus method approximates the gradient of a (n+1) dimensional plane, which is composed by combining an extra point with n selected vertices in the original simplex. The following section illustrates the details of this method through 2-dimension case.

2.4.1 Initialization.

For the minimization of functions with N variables, N+1 vertices are generated to define initial simplex. 3 vertex points are generated for 2 variable case. Calculate the extra vertex Xe with its coordinates equals to the diagonal of matrix X, see equation (2.13), which is composed by n selected vertex points from original (n+1) random points except the worst point, see figure 2.3.

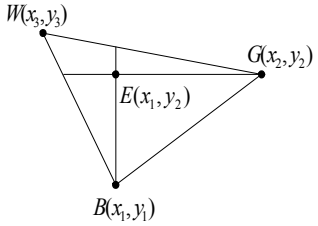


Figure 1 Nelder-Mead simplex Plus

$$X_E = diag\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} \tag{2.13}$$

2.4.2 Approximate gradient

Calculate each gradient of the plane in each dimension based on the following equations [6].

for 2-dimension case, the plane gradient of ΔBGE is:

$$\begin{cases} g_1 = \dfrac{\Delta f}{\Delta x} = \dfrac{f_B - f_E}{x_2 - x_1} \\ g_2 = \dfrac{\Delta f}{\Delta y} = \dfrac{f_G - f_E}{y_1 - y_2} \end{cases} \tag{2.14}$$

2.4.3 Reflection

Unlike original Nelder-Mead simplex using the middle point of BG, the Nelder-Mead simplex Plus defines the reflection point with point B (best) and gradient vector G

$$x_{R'} = x_B - \sigma * G \text{, where } G = [g_1; g_2] \tag{2.15}$$

Where σ is the learning constant.

If $f(x_{R'}) \le f(x_B)$, $\overrightarrow{BR'}$ is the correct direction toward minimum, then extend the line segment to E in the direction of $\overrightarrow{BR'}$ with the following relation

$$x_{E'} = (1-\gamma)x_B + \gamma x_{R'} \tag{2.16}$$

Where γ is expansion coefficient, suggested $\gamma = 0.5$, same with original algorithm suggested.

2.5 Quasi Gradient method using hyper plane equation

Quasi Gradient method using hyper plane equation needs to calculate the gradient of the (n+1) dimensional plane, but it used the originally generated (n+1) vertices in the simplex rather than generating a new extension point. The following section illustrates it in detail:

2.5.1 Initialization.

For the minimization of functions with n variables, generate $n+1$ vertex to define initial simplex. Assuming this $n+1$ dimensional plane formed by the vertex has the following relation

$$V = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_{n-1} x_{n-1} + a_n x_n \tag{2.17}$$

2.5.2 Calculate gradient

Substitute $n+1$ points into equation (2.17), $n+1$ equations are generated to solve $n+1$ unknowns.

$$\begin{cases} V_1 = a_0 + a_1 x_{11} + a_2 x_{21} + \ldots + a_{n\text{-}1} x_{1,n\text{-}1} + a_n x_{1,n} \\ V_2 = a_0 + a_1 x_{21} + a_2 x_{22} + \ldots + a_{n\text{-}1} x_{2,n\text{-}1} + a_n x_{2,n} \\ \cdots\cdots \\ V_n = a_0 + a_1 x_{n1} + a_2 x_{n2} + \ldots + a_{n\text{-}1} x_{n,n\text{-}1} + a_n x_n \\ V_{n+1} = a_0 + a_1 x_{n+1,1} + a_2 x_{n+1,2} + \ldots + a_{n\text{-}1} x_{n+1,n\text{-}1} + a_n x_{n+1,n} \end{cases} \quad (2.18)$$

Next, we calculate the approximate gradient vector by solving equation (2.18) based on the matrix operation $V = X \times G$.

$$G = X^{-1}V = [a_0\, a_1\, a_2 \ldots a_{n-1}\, a_n] = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_{11} & x_{21} & \ldots & x_{n+1,1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n} & x_{2n} & \ldots & x_{n+1,n} \end{bmatrix}^{-1} \times \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_{n+1} \end{bmatrix} \quad (2.19)$$

2.5.3 Reflection and Expansion

Similar to Nelder-Mead Simple Plus with extra vertex, the Quasi Gradient method defines reflection point R' with vertex B instead of the midpoint M of BG in the original simplex using equation $R' = B - \sigma * G$

If $f(x_{R'}) \le f(x_B)$, $\overrightarrow{BR'}$ is the correct direction toward minimum, then extend the line segment to E' in the direction of $\overrightarrow{BR'}$ with the relation $E' = (1-\gamma)B + \gamma R'$

The two improvement of Nelder-Mead simplex algorithm are described above. One may notice that these two methods approximate or calculate gradients of simplex, and then use this gradient

to guide the generation of new reflected point. It is well-known that the gradient includes important information of the function. These two improvements of original simplex makes full use of gradient information. It has been reported that these two improved methods are to be more efficient and reliable than the original simplex[6], the comparison of these methods is presented in the next section and the results are discussed.

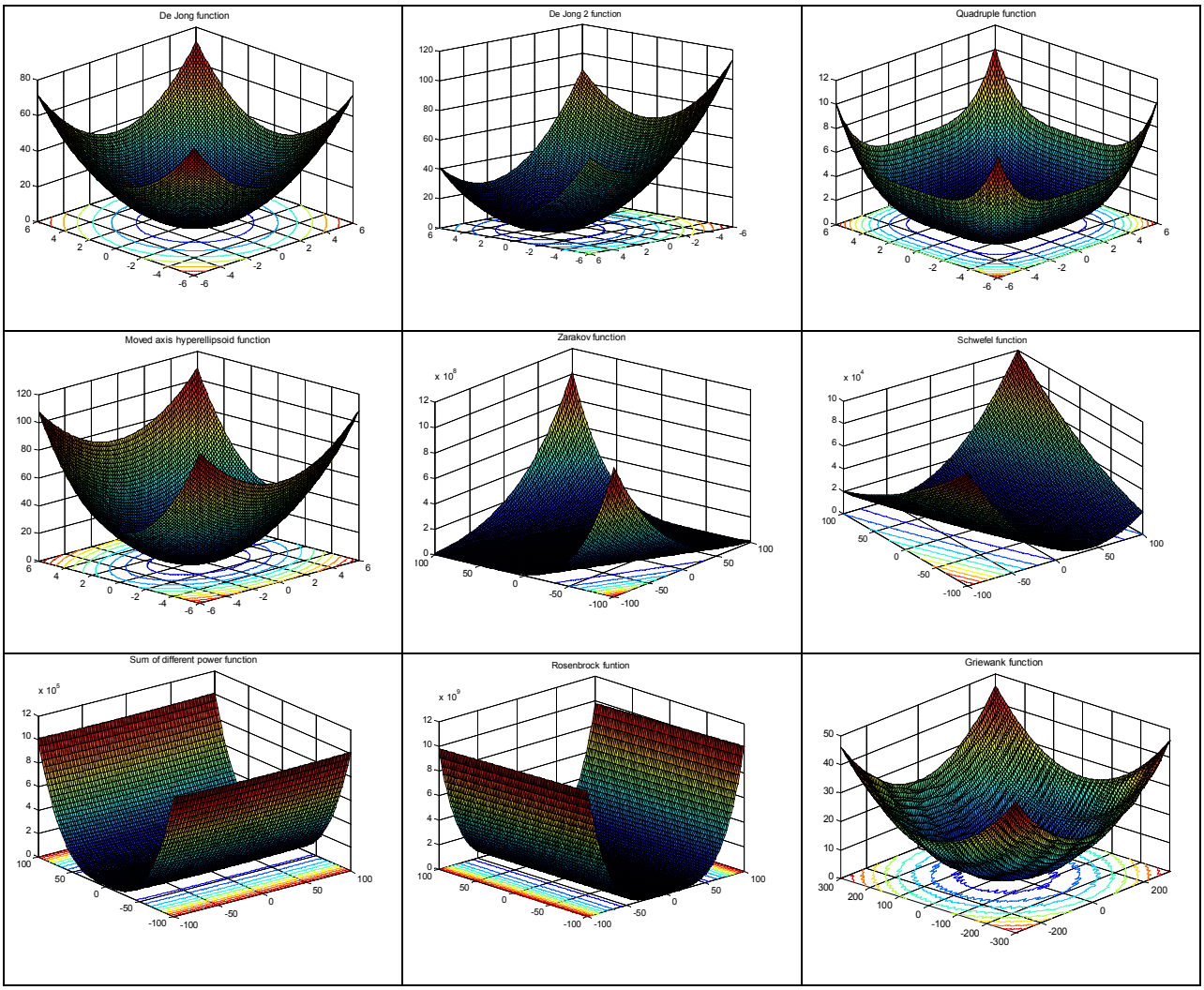
Chapter 3

Experimental results

3.1 Benchmark functions used for comparison

A set of functions with various difficulty levels are used to test the reliability and robustness of five derivative-free algorithms. These functions listed in figure 2 are famous unconstrained optimization functions. Some algorithms may have a tendency to converge towards local minimum or any arbitrary points instead of the global minimum of the function. The possibility of this occurring relies on the shape of the function landscape: some problems may provide an easy descent towards the global minimum, while others may make it easier to get stuck in local optima. Figure 2 shows a snapshot of some of the functions in two dimensions. These graphs can give some clues of the topographical structures of each problem.

Figure 2: topographical structure of some functions (for n=2)



A large number of problems are comparatively sufficient to test whether these algorithms are reliable and robust or not than a small scale of problems. Thus, all these functions will be tested. In order to measure the reliability and robustness fairly, all algorithms are tested within a wide range rather than close to solution domain, which may not show the genuine exploration and exploitation capability of each algorithm. Thus, the initial points for simplex series method or initial population boundary for PSO and GA are generated at random within same boundary. Experiments results have already shown that the location of initial point or position for particles

affects the success rate and computing time, measured by Matlab, dramatically. Besides, we also use the listed functions below explore the effects of different population sizes on PSO performance. We use the following general format define the benchmark functions: The test functions employed have minimum $f = 0$, the minimum coordinate is followed behind the definition of the functions.

Table 3 Test functions used in experiment

Name	Formula	minimum
De Jong function	$F_1 = \sum_{i=1}^{n} x_i^2$	$f = 0$ at $(0,0,...,0)$
De Jong function with moved axis	$F_2 = \sum_{i=1}^{n} (x_i - a_i)^2$	$f = 0$ at $(a_1, a_2, ..., a_n)$
Quadruple function	$F_3 = \sum_{i=1}^{n} \left(\frac{x_i}{4}\right)^4$	$f = 0$ at $(0,0,...,0)$
Powell function	$F_4 = \sum_{i=1}^{n} [(x_i + 10x_{i+1})^2 + 5(x_{i+2} - x_{i+3})^2 + (x_{i+1} - 2x_{i+2})^4 + 10(x_i - x_{i+3})^4]$	$f = 0$ at $(0,0,...,0)$
Moved axis Parallel hyper-ellipsoid function	$F_5 = \sum_{i=1}^{n} i x_i^2$	$f = 0$ at $(0,0,...,0)$
Zarakov function	$F_6 = \sum_{i=1}^{n} [x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^4]$	$f = 0$ at $(0,0,...,0)$
Schwefel function	$F_7 = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	$f = 0$ at $(0,0,...,0)$
Sum of different power function	$F_8 = \sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$	$f = 0$ at $(0,0,...,0)$
Step function	$F_9 = \sum_{i=1}^{n} \lvert x_i + 0.5 \rvert^2$	$f = 0$ at $(-0.5, -0.5, ..., -0.5)$
Griewank function	$F_{10} = \frac{1}{4000} \sum_{i=1}^{n} (x_i - 100)^2 - \prod_{i=1}^{n} \cos[\frac{(x_i - 100)}{\sqrt{i}}] + 1$	$f = 0$ at $(0,0,...,0)$
Rosenbrock function	$F_{11} = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$f = 0$ at $(1,1,...1,1)$
Biggs Exp6 function	$F_{12} = \sum_{i=1}^{n} [x_{i+2} e^{t_i x_i} - x_{i+3} e^{-t_i x_{i+1}} + x_{i+5} e^{t_i x_{i+4}} - y_i]^2$, where $t_i = 0.5i$ and $y_i = e^{-t_i} - 5e^{-10t_i} + 3e^{-4t_i}$	$f = 0$ at many points

Wood function	$F_{13} = \sum_{i=1}^{n} [\ 100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2 + 90(x_{i+3} - x_{i+2}^2)^2 + (1 - x_{i+2})^2$ $+10.1((1 - x_{i+1})^2 + (1 - x_{i+3})^2) + 19.8(1 - x_{i+1})(1 - x_{i+3})]$	$f = 0$ at $(1,1,\ldots 1,1)$

3.2 Experiments and Results

Experiments were run on the listed functions above. In order to measure each of these algorithm's problem solving ability, the success rate and computing time of the benchmark functions are evaluated. In order to test the effect of function dimension to each algorithm, all functions were run with dimension 10, 15, and 20. Then based on the obtained results, use top performed algorithms solve 25 30 dimensional problems. For particle swarm optimization, the self-adjust and social-adjust parameters are set to $c_1 = c_2 = 2$, and the inertial weight w varies with problems. The initial population for GA and PSO is 100 at the first start. This initial number can balance the success rate and convergence speed by the author of this thesis. For Nelder-Mead Simplex, suggested parameters were used by Nelder and Mead $\alpha=1,\beta=0.5,\gamma=2$, which have been proved to be the best value for this algorithm [7]. Same parameters were used in two Improved Simplex method and the learning constant $\sigma=1$ in both two improvement [6]. In another words, the setting of parameters for each algorithm will make sure each algorithm perform their best potential.

Experiments were also run to test the effect of different initial population or swarm sizes. These tests are based on the test experiments in dimension 10, 15, 20. For those problems, the convergence success rate is not 100% or computing cost is high, try different initial population or swarm size 200 300 500 and test the success rate and computing time of different population size. After comparing their performance, choose the proper initial population size do further experiments.

In order to fairly compare the performance of these algorithms, assuming some setting: all the algorithms are started within a range of [-100 100; the dimension of all benchmark functions are tested in 10,15,20, 25, 30. Maximum iteration is equal to 100000 for all algorithms; and stopping criteria is equal to 0.001. Because genetic algorithm need more time to do recombination crossover and mutation and reinsertion operations, which need relatively high time cost, so all results are the average value calculated over 25 random running times. All experiments are tested on a laptop with Inter i7 processor, 2.40 GHz with MatLab software.

Table 4: evaluation of success rate and computing time of 10-dimensional functions

Test **function**	**GA(Nind=100)**		**PSO(Nind=100)**		**SIM**		**NM+**		**QG**	
	Success rate	Comp. time (s)	Succes s rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
F_1	100%	4.7827	100%	0.0712	100%	0.0742	100%	0.0487	100%	0.0262
F_2	100%	5.6787	100%	0.0741	100%	0.1234	100%	0.0677	100%	0.0319
F_3	100%	2.5658	100%	0.0794	88%	1.0789	100%	0.0579	100%	0.0270
F_4	100%	12.3699	100%	0.6480	100%	0.0305	100%	0.0414	100%	0.0479
F_5	100%	6.4087	100%	0.1835	100%	0.0847	100%	0.0484	100%	0.0366
F_6	Failed		100%	0.6212	Failed		100%	0.2281	100%	0.2065
F_7	Failed		100%	0.5222	84%	1.1378	100%	0.3682	100%	0.6806
F_8	100%	5.3202	100%	0.1447	Failed		100%	0.0826	100%	0.0987
F_9	100%	2.3618	100%	0.0296	100%	0.1026	100%	0.0590	100%	0.0284
F_{11}	Failed		100%	0.9276	28%	4.8608	68%	5.3331	88%	8.3774
F_{12}	44%	94.3697	85%	1.7766	Failed		20%	17.8289	48%	13.7819
F_{13}	Failed		100%	0.995	32%	4.6299	56%	6.9449	56%	11.0435

Table 5: evaluation of success rate and computing time of 15-dimensional functions

Test **function**	**GA**		**PSO**		**SIM**		**NM+**		**QG**	
	Success rate	Comp. time (s)	Succes s rate	Comp. time (s)	Success rate	Comp. time(s)	Succes s rate	Comp. time (s)	Success rate	Comp. time (s)
F_1	100%	8.4377	100%	0.11426	Failed		100%	0.3450	100%	0.3450
F_2	Failed		100%	0.1949	Failed		100%	0.1837	100%	0.0481
F_3	100%	4.4090	100%	0.2021	Failed		100%	0.1743	100%	0.0439

F_4	100%	24.8846	100%	2.7320	100%	0.0443	100%	0.0553	100%	0.0648
F_5	100%	10.9912	100%	0.2498	Failed		100%	0.1238	100%	0.0752
F_6	Failed		100%	3.6743	Failed		100%	0.7858	100%	0.6764
F_7	Failed		100%	3.1185	Failed		100%	1.5104	100%	3.7148
F_8	Failed		100%	0.3475	Failed		100%	0.2051	100%	0.1852
F_9	100%	3.1270	100%	0.0394	Failed		100%	0.1233	100%	0.0420
F_{11}	Failed		70%	6.3059	Failed		56%	9.3315	84%	5.7074
F_{12}	20%	777.223	70%	6.4808	Failed		27%	9.0644	27%	9.0644
F_{13}	Failed		80%	3.7016	Failed		48%	10.1281	60%	12.0324

Table 6: evaluation of success rate and computing time of 20-dimensional functions

Test **function**	**GA**		**PSO**		**SIM**		**NM+**		**QG**	
	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp time(s)	Succe ss rate	Comp. time (s)	Success rate	Comp. time (s)
F_1	100%	12.9404	100%	0.1143	Failed		100%	0.2502	100%	0.0975
F_2	Failed		100%	0.4189	Failed		100%	0.5117	100%	0.1181
F_3	100%	6.4663	100%	2.4447	Failed		100%	0.2631	100%	0.0765
F_4	100%	38.6432	100%	7.8028	100%	0.0652	100%	0.0743	100%	0.1346
F_5	100%	17.6976	100%	0.5339	Failed		100%	0.2510	100%	0.2124
F_6	Failed		100%	9.9392	Failed		100%	2.0740	100%	2.4617
F_7	Failed		20%	12.2404	Failed		100%	4.5500	100%	20.8769
F_8	Failed		100%	0.8183	Failed		100%	0.4396	100%	0.5778
F_9	100%	3.6904	100%	0.0506	Failed		100%	0.2397	100%	0.0948
F_{11}	Failed		Failed		Failed		68%	10.5583	84%	10.3003
F_{12}	Failed		40%	15.2367	Failed		27%	9.0644	27%	9.0644
F_{13}	Failed		90%	4.5525	Failed		40%	13.9083	56%	22.7699

Comparing five derivative free algorithms' capability in finding global minimum with dimension 10, 15, 20 respectively. The result is analyzed within same dimension 10 for five algorithms, then analyzed by discussing the effect of dimension to same algorithm.

1. Genetic algorithm:

When compare with other algorithms for function dimension 10, there are seven cases that GA can obtain 100% success rates (from table 4), which proves the genetic algorithm's global search ability. In particular, it can get 72% success rate for Biggs Exp6, which is better than Simplex, and two improved Simplex method. If one changes the mutation and crossover probability, the result also changes. If one adopts elite GA, the result would be better as a whole, this comparison about *GA and elite GA* is in next section. But one also notice that GA needs more time in finding global minimum in cases with 100% success rate compared with other four algorithms. This can be explained by its natural behavior operations, the complex recombination and crossover, as well as mutation operations.

When compared the effect of function dimension to genetic algorithm, one can see that the computing time for same function increases as the increase of dimension from 10 to 20. Take De Jong function as an example, the computing time increases from 4.7827s for dimension 10 to 12.9404s for dimension 20. The computing time increases dramatically with the dimension growth. However, genetic algorithm can maintain the same success rate for those functions it can converge successfully.

2. Particle swarm algorithm:

When compared results with other algorithms about 10 dimension optimization, particle swarm algorithm performs really well with these benchmark functions. There are eight cases that PSO can obtain 100% success rate. And five cases can converge with more than 50 percent success rate. When comparing computing time, PSO can converge faster than genetic algorithm while keep higher success rate as the same time, but a little bit slower for same functions than two improved simplex algorithms. For cases such as Biggs exp6 and Wood function, PSO wins over

other algorithms. It can reach 90% and 60% success rate respectively for dimension 10, which is the best result among all algorithms. There are two cases, Schewefel and Rosenbrock functions, PSO cannot converge with 100% success rate but the result looks promising. One try PSO with more particles to better search the domain, further comparsion is shown in next section. The success rate can reach almost 100% if the particles is equal to 300 (among all the PSO simulation the particle number is equal to 100). Although the time cost is a little bit higher than Improved simplex methods with some cases, as a trade-off, the success rate is higher.

When we compared the effect of function dimension, similar to GA, the computing cost increases with the growth of function dimension, for instance computing time for Powell function (F_4) is 0.6480s for 10 dimension while the computing time increased to 7.8028s for 20 dimension. Same tendency happens at other functions. the success rate decreases. On the other hand, the increase of dimension also makes function optimization more complicated. This can be seen from the decrease of success rate as the growth of dimension. Biggs Exp6 function can reach 85% success rate at dimension 10, while it only reach 40% success rate at dimension 20 with same particle swarm parameter setting.

3. Nelder-Mead Simplex Method:

From the simulation result in Table 4, one may notice that there are six cases that it can get 100% success rate for dimension 10. And two cases, it can converge with low success rate. It cannot converge with five cases. These facts proves the limitation of Nelder-Mead's Simplex method. Since it uses (n+1) vertex get the direction, it may be easily fall into local minimum or may not converge at all in solving complex problems, especially in optimizing high dimension case. For problem with higher dimension 15 and 20, the Nelder-Mead Simplex fails to converge to global

minimum easily, as we can see form table 5 and 6 none of functions were converged with 100% success rate. That is what urges Nam Pham presents a solution to improve the deficiency of Nelder-Mead's simplex algorithm by incorporating with a extra vertex and hyper plane equation and calculating gradient of these plane.

4. Nelder-Mead Plus method with an extra vertex:

The experiment results shows, from table 4, there are 10 cases that can reach 100% success rate, which contrast sharply with original simplex's six cases 100% success convergence. Besides, other cases like F_{11} F_{12} F_{13} also get higher success rate than original simplex algorithm. On the other hand, the computing time for all cases for improvement with extra vertex is less than that of original simplex algorithm. Therefore, the SIM1 greatly improved the success rate and computing time and can be more reliable and robust. From table 4, 5, 6, the improved simplex method maintains its reliability even in higher dimension problem.

5. Gradient Method with a hyper plane:

From table 4, one may notice that there are 10 cases that can reach 100% success rate. Other three cases also obtain higher success rate than both original simplex and NM+ as a whole. From another perspective, the computing time of QG for all 100% cases are fastest among all algorithms. So the computation to approximate the gradient matrix did not increase the computations cost, instead, with relatively proper function guide, the improved simplex with hyper plane can converge with less iterations. However, for those cases which did not obtain 100% success rate, the computing time is larger than NM+ and PSO. This can be explained by complexity of the problems since some functions have hundreds of local minimum, such as F_{10} Griewank function, and the gradient of current plane may not direct to global minimum, the

computation of gradient may not reliable in this situation. But there is no doubt that the improved simplex method with gradient greatly increases the reliability and robustness compared with original simplex method.

When compared the effect of function dimension, similar to GA and PSO, the computing cost increases with the growth of function dimension. On the other hand, the increase of dimension also makes function optimization more complicated. This can be seen from the decrease of success rate as the growth of dimension for same function. However, this is not exception for improved simplex algorithms.

In sum, by analyzing the results from table 4, 5, 6, we can see that genetic algorithm can converge on some of problems with high success rate even in high dimension but the computing time increases dramatically as the growth of function dimension, so it is suitable for those real-time required problems. Nelder-Mead Simplex method fails to converge easily when the dimension of function increases to 15. On the contrary, our version of particle swarm performs its potential in high dimension problem solving, which adopts the vector operation speeding the convergence. The two improved simplex methods also show their reliability and robustness in solving high dimension problems. For those problems which fails to converge by original Nelder-Mead simplex algorithm, these two improvements can deal with them with better success rate and less computing cost. The first improvement does not need high computational cost since it uses an extra vertex but its accuracy strongly relies on the linearity of to-be-optimized functions in the vicinity of the simplex. By contrast, the second method is better by using the hyper plane equation. In short, the guide of gradient really improved the performance of two improved Nelder-Mead simplex methods. Thus, we will use outperformed three algorithms,

particle swarm optimization algorithm and two improvements of Nelder-Mead simplex, optimize high dimension benchmark functions in the future.

Chapter 4

Test Effect of Initial Population Size for PSO

4.1 **Study of the effect of Initial population size**

In this section, experiments were run on four typical functions selected from benchmark function table 3, De Jong function, Quadruple function, Zarakov function, and Rastrigin function. The intention of this part is to test the effect of different initial population size on computing time and success rate. By comparing the effect of different initial population size, one can choose the proper initial size for PSO to optimize higher dimension problems. In this test, initial population varies from 50 100, 200, 300, 500 800 1000 (for time-consuming case 1000 is eliminated). The relationship between population size and performance of each algorithm is plot out.

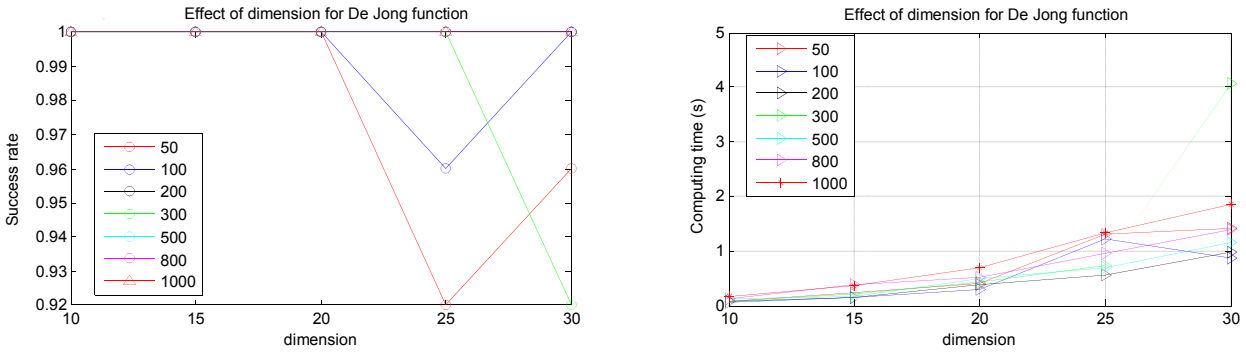


Figure 3 Effect of dimension for De Jong Function

Figure 3 shows the effect of dimension on De Jong function. Since De Jong function is very similar to sphere, the global minimum can be found simply by following the gradient of the sphere. By analyzing De Jong function, one can extend to sphere-like functions like De Jong 2 and moved axis parallel hyper-ellipsoid and so on. For this problem, when the dimension of the function is less than 20, any tested population size can achieve 100% success rate, and the

computing cost is the smallest with population size 100. However, when the dimension of the function equals to 25 or larger, the success rate decreases with the population size 50 or 100. Meanwhile, the computing time also increase dramatically when the success rate fails to reach 100%. This can be explained by the fact, if it fails to converge to the global minimum, it stops until reaching the setting iteration number. From the above figure, one may notice that performance improved when the population size increased from 100 to 200, the success rate can go back to 100% and the computing cost is the smallest among all the tested population size in high dimension optimization. As the growth of population size, the computing cost also increases although same success rate keeps. As a tradeoff between the success rate and computing time, the proper tested initial population size for De Jong function should be 200~300.

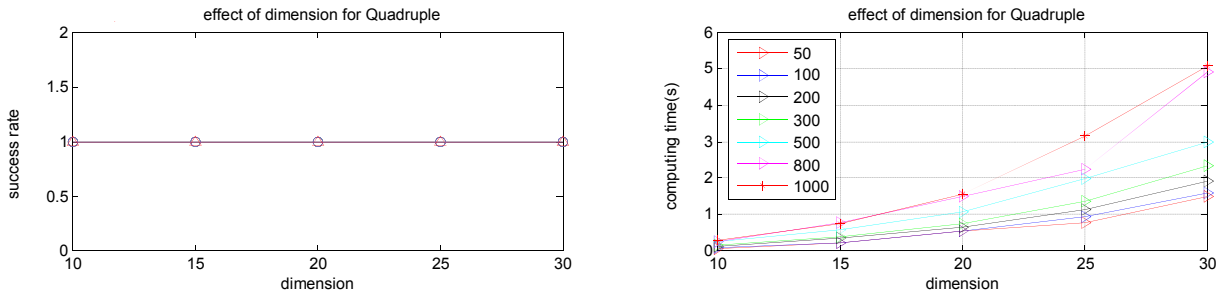


Figure 4 Effect of dimension for Quadruple function

Figure 4 shows the effect of dimension on quadruple function, all the tested population sizes are reaching same success rate 100%. And the lower of the number of population size, the less of computing cost, which can be seen from the figure 4. When dimension of the problem larger than 25, the computing cost increase dramatically with the growth of initial population size while the success rate keeps the same. For this function, 50 initial population size is suitable to balance the success rate and computing time. In order to keep consistent with other function 300 hundred is chosen.

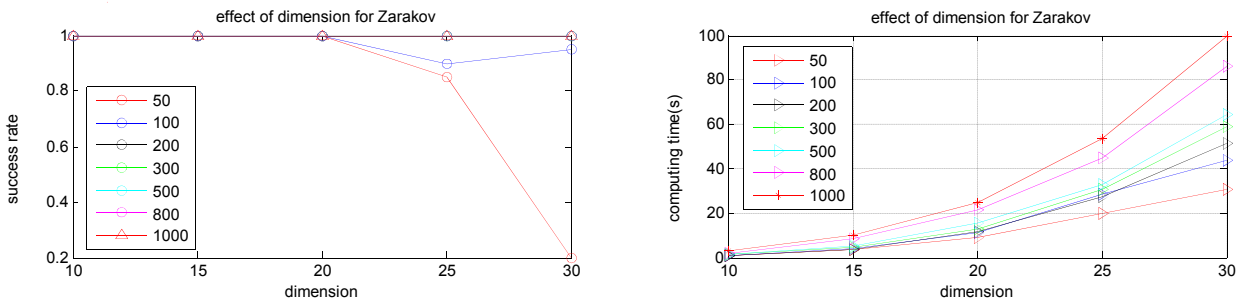


Figure 5 Effect of dimension for Zarakov Function

For Zarakov function, when dimension of the function is less than 20, initial population size with 50 performs really well. However, when dimension of function is increased to 25, the success rate tends to decrease severely. The degradation of the performance observed from the generated data is obvious when dimension equals to 30. When initial population increased to 200, PSO performs well both on success rate and computing time. The degradation of performance as the initial swarm size increases from 300 to1000 is much severe, since the computing time increase dramatically. For this problem 200 ~300 is proper.

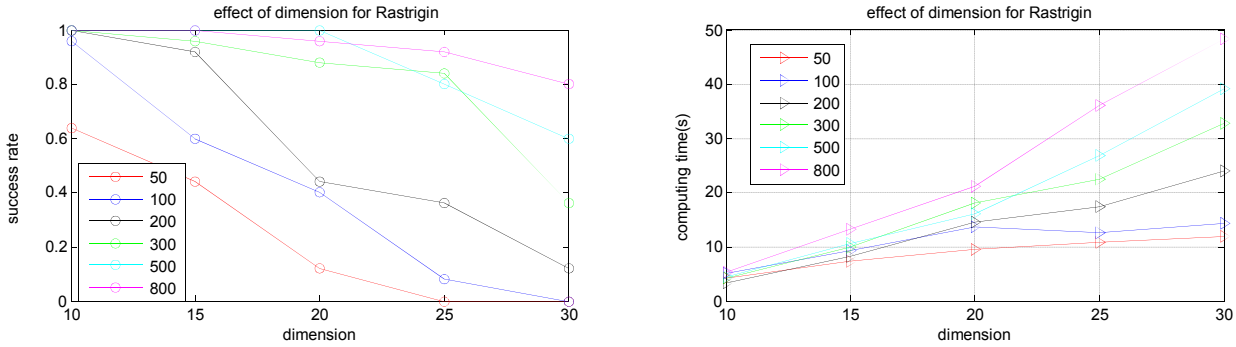


Figure 6 Effect of dimension for Rastrigin function

Figure 6 shows the effect of dimension on Rastrigin function. Compared with De Jong and Sphere liked funtions, the performance of particle swarm algorithm increases with the growth of

initial population size. From the left figure of 6, one may notice when initial population equals to 50, it obtain lowest success rate for all dimension problems. With a larger population size, there is a better chance that one of particles will fall into the minima close to global minima and attract other particle to global minima area. Since there are hundreds of local minimums for rastrigin function, it is really a challenge for algorithm, especially for those derivative-based algorithms. Thus, the success rate increases as the initial population size is growing, especially in high dimension optimization. However, from right figure of 6, one can see as the increase of success rate is obtained at the price of the computing time, which increased dramatically with the growth of initial population. As a whole, 300 initial population size is feasible for this kind of functions.

The reason why we test the relationship between function dimension and initial population size is because the author finds sometimes it may not work well using original parameter setting. As a result, this is problem-dependent just as what we analyzed above, so it is necessary to do this kinds of research before using particle swarm optimization dissolve the real life problems.

4.2 Optimization with top performing algorithms

From section 4.2, we compared five algorithms performance from the perspective of computing time and success rate. The genetic algorithm can converge on some of problems with high success rate even in high dimension but the computing time increases dramatically as the growth of function dimension, so it is not a good choice for solving high dimension problems, especially for those real-time required problems. Nelder-Mead Simplex method fails to converge easily when the dimension of function increases. On the contrary, our version of particle swarm performs its potential in high dimension problem solving, which adopts the vector operation. Based on section 4.3 and 4.2, we can see our version of particle swarm optimization can be a reliable method for solving high dimension problems if we choose proper initial population size, which is proved to be a important parameter in control the balance between the success rate and computing cost. The population size 300 has been proved to be suitable for general use. The two improved simplex methods also show their reliability and robustness in solving high dimension problems. For those problems which fails to converge by original Nelder-Mead simplex algorithm, these two improvements can deal with them with better success rate and less computing cost. The guide of gradient really improved the performance of two improved Nelder-Mead simplex methods. Thus, we will use particle swarm optimization algorithm and two improvement of Nelder-Mead simplex optimize high dimension benchmark functions in this section.

For particle swarm optimization, the self-adjust and social-adjust parameters are set to $c_1 = c_2 = 2$ the inertial weight w set to 0.81. The initial population PSO is 300 for general use. This initial number can balance the success rate and convergence speed proved in section 4.3. For two

improvement of Nelder-Mead Simplex, suggested parameters were used γ=2, the learning constant σ=1,which have been proved to be the best value for this algorithm [10],[11]. In order to fairly compare the performance of these algorithms, same assumption used: all the algorithms are started within a range of [-100 100], instead of using standard initial points for a certain functions like some papers did; the dimension of all benchmark functions tested in 25, 30. Maximum iteration is equal to 100000 for all algorithms; and stopping criteria is equal to 0.001. All results are the average value calculated over 25 random running times. All experiments are tested on a laptop with Inter i7 processor, 2.4 GHz.

Table 7: Performance of 25-dimensional functions

Test **function**		**PSO**		**NM+**		**QG**	
		Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
F_1	*De Jong*	100%	0.5606	100%	0.4762	100%	0.1493
F_2	*De Jong with moved axis*	100%	0.8796	100%	0.7537	100%	0.1742
F_3	*Quadruple*	100%	1.6464	100%	0.5318	100%	0.1194
F_4	*Powell*	96%	30.3961	100%	0.0898	100%	0.1711
F_5	*Moved axis Parallel hyper-ellipsoid*	100%	1.1362	100%	0.4825	100%	0.4096
F_6	*Zarakov*	100%	30.3786	100%	4.2772	100%	4.8671
F_7	*Schwefel*	72%	43.7442	100%	13.0545	Failed	
F_8	*Sum of different power*	100%	3.4545	100%	1.2938	100%	0.9763
F_9	*Step*	100%	0.0642	100%	0.4435	100%	0.1426
F_{10}	*Griewank function*	24%	28.0918	48%	13.5620	60%	20.5668
F_{11}	*Rosenbrock*	4%	26.8775	16%	12.5733	44%	30.2219
F_{12}	*Biggs Exp6*	36%	63.8550	Failed		Failed	
F_{13}	*Wood*	92%	11.3487	48%	13.5620	60%	20.5668

Table 8: Performance of 30-dimensional functions

Test **function**		**PSO**		**NM+**		**QG**	
		Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
F_1	*De Jong*	100%	1.1885	100%	0.7435	100%	0.2017
F_2	*De Jong with moved axis*	100%	1.6284	100%	1.2280	100%	0.2458
F_3	*Quadruple*	100%	2.5896	100%	0.7957	100%	0.1785
F_4	*Powell*	92%	57.7884	100%	0.1195	100%	0.2505
F_5	*Moved axis Parallel hyper-ellipsoid*	100%	1.7313	100%	0.6468	100%	0.6567
F_6	*Zarakov*	100%	61.1399	100%	8.2332	100%	9.0665
F_7	*Schwefel*	12%	63.0315	92%	31.1472	Failed	
F_8	*Sum of different power*	100%	4.6451	84%	4.9543	100%	1.7589
F_9	*Step*	100%	0.0726	100%	0.6798	100%	0.2061
F_{10}	*Griewank function*	28%	27.9921	48%	17.7923	36%	35.9403
F_{11}	*Rosenbrock*	Failed		24%	12.6696	82%	17.6331
F_{12}	*Biggs Exp6*	Failed		Failed		Failed	
F_{13}	*Wood*	80%	22.4153	48%	17.7923	36%	35.9403

From table 7 and 8, we can see three optimization methods can search the global minimum successfully for functions like De Jong, De Jong function with moved axis, Quadruple function, sum of different power function, as well as step function. Review figure 2, from the 2 dimension shape of these functions, we can see that these function have steep gradient toward the global minimum, that is why the Second improvement with quasi gradient performs better that particle swarm and improvement with extra vertex, for these functions, it can converge faster than two other algorithms. The smooth figure and steep gradient make it easy to converge. This high dimension function optimization further tests each algorithm's search ability.

4.3 **Real World Application**

It has been proved above that Particle Swarm Optimization can works well in optimizing high dimensional problems by a set of benchmark functions. One will try to use the vectorized PSO optimize artificial neuron networks. Currently many researchers use derivative-based algorithms, Error Back Propagation (EBP) [26], Lavenberg Marquardt (LM) [27] and Neuron by Neuron algorithm (NBN) [28][29], train neuron network parameters. It is nature for people to try to apply new methods proved to be efficient on other field. Before we train neuron network, we need to briefly illustrate some parameters of neuron network, for more detailed information of neuron network derivative-based algorithms refer to [26][30].

The neural network (NN) is formed by connected neurons and trained by input patterns and desired output. The purpose of training NN is to minimize the sum square error between actual output and desired outputs by adjusting the weights of network. Usually the neural network includes input layer, a nonlinear hidden layer and a linear output summator, as shown in Fig.1.

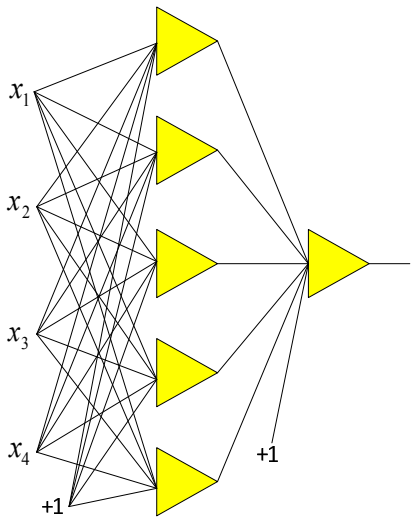


Figure 7 Neural network architecture

Hidden parameters $w=[w_1,w_2,...,w_m]$; Output weights $\theta=[\theta_0,\theta_1,...,\theta_k]$; all parameters $\varphi=[w,\theta]$; desired outputs $y=[y_1,y_2,...y_p]^T$; actual outputs $\bar{y}=[\bar{y}_1,\bar{y}_2,...,\bar{y}_p]^T$.

For all training patterns $p=(1,2,...,P)$ minimize Sum Squared Error $SSE=(y-\bar{y})^T\times(y-\bar{y})$

The hidden layer neurons have nonlinear activation functions. The most popular used activation function is sigmoid functions. The output of the sigmoid neurons can be calculated by

$$h(x)=\frac{1}{1+e^{-\rho(\sum_i^n w_i x_i - w_{bias})}} \tag{1.1}$$

The main challenges of neural network training include 1) what kind of neural network architectures should be used. 2). How large a neural network should be [31]. There are three neural network architecture widely used, Multilayer perceptron (MLP), bridged multilayer perceptron (BMLP) and fully cascade connected architecture (FCC). The MLP architecture is favored by most researchers because it applies to almost any problems. But it is less powerful than FCC and BMLP (bridged multilayer perceptron) architecture. In this master thesis, we will use all of these three architectures neural network topology to reconstruct function surface. Trail-and error approach is used in order to determine the optimal network size. One important feature of neural network is the generalization ability, which means the neural network should correctly respond to new patterns which are not used in training [30]. The number of neurons affects the generalization ability of the neuron network, so in order to obtain optimum performance, neural networks should have as few neurons as possible [31]. In another word, the simpler, the better for neuron network.

In the experiment, we apply 2, 3, 4 neurons to MLP, BMLP, and FCC architecture one by one. 400 input patterns are generated by function for training, 1600 patterns for reconstruction the

actual surface. As we described before, the number of neurons affects the generalization ability of the neuron network. Comparison with 2, 3, 4 neurons for MLP and FCC architecture, one may notice from the following result: 2 neurons is not enough for this the reconstruction of this control surface, 3 or 4 neurons is suitable for the reconstruction of the surface.

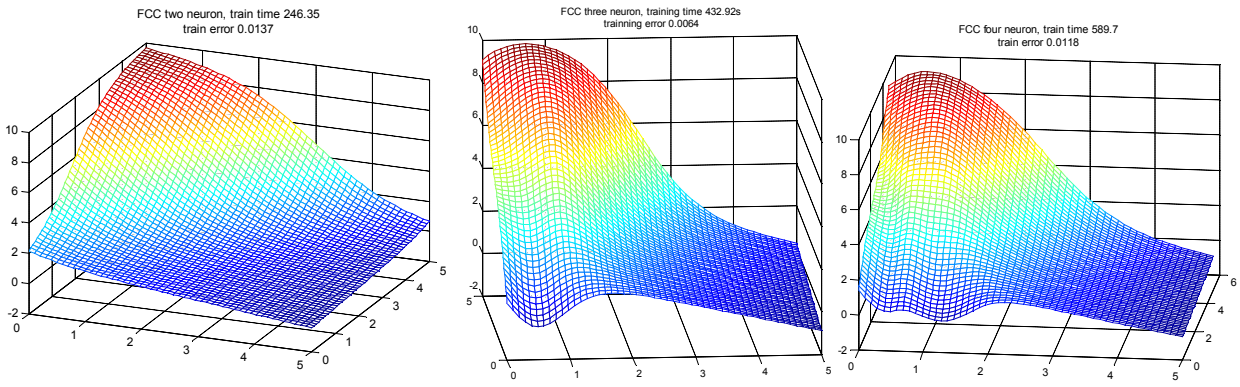


Figure 8 FCC training with two, three, four neurons

In the following section, 400 patterns are used for training. The initial population for PSO algorithm increases from 300 to 500, at the same time, the iteration number decrease from 1e6 to 5000 in order to balance the training time. The desired surface is in the following figure.

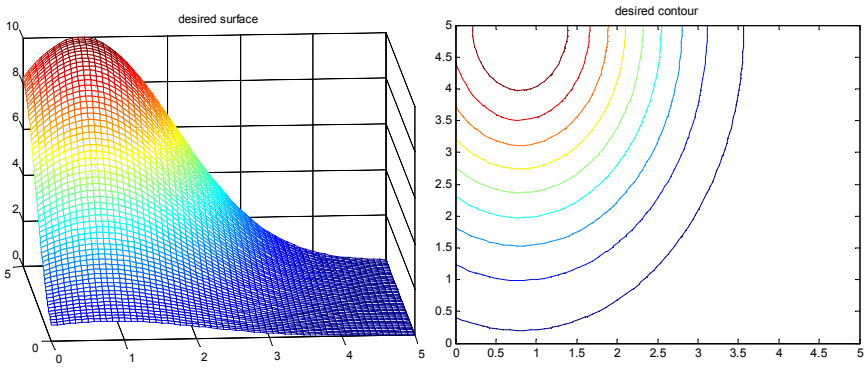


Figure 9 Desired surface and contour

1. MLP

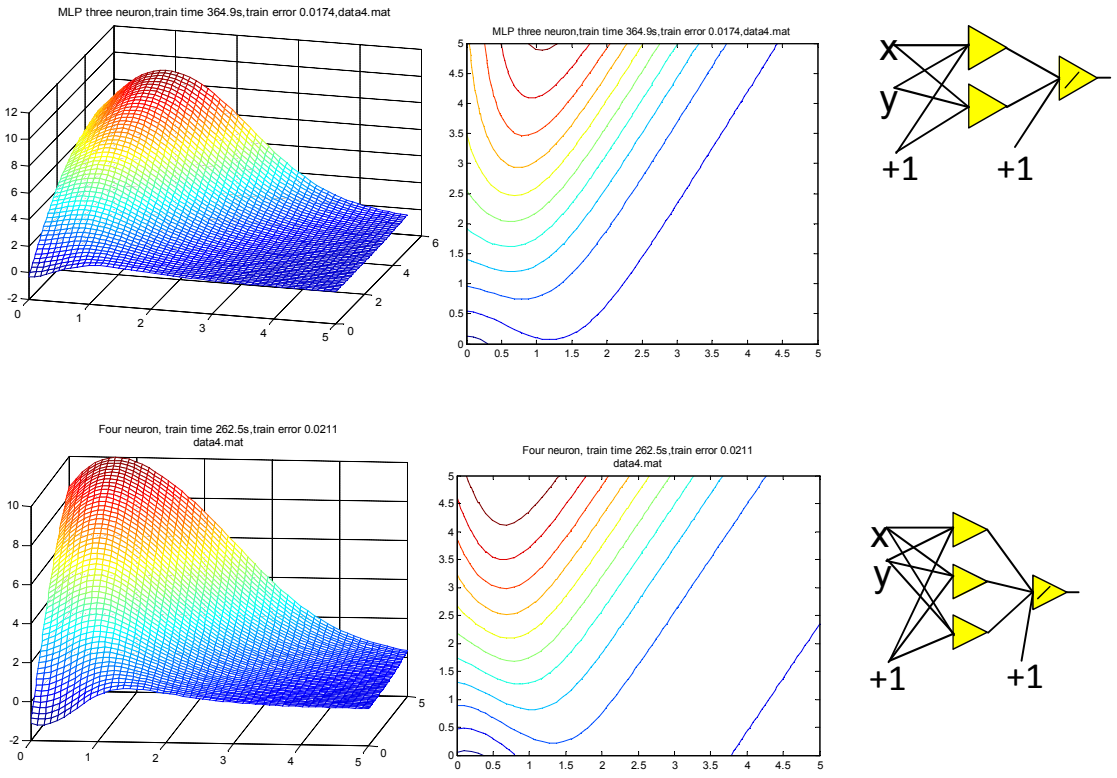


Figure 10 MLP training architecture and result with three and four neurons

Compared with data1.mat training result, it seems that data4.mat training result is better than that in terms of the overtraining problem. The control surface is better than data1.mat. Although the train error for data4.mat is 0.0211, smaller than train error 0.0081 for data1.mat, it produces better surface for new patterns. Also it seems that 3 neurons architecture may perform well than four neuron architecture.

2. BMLP

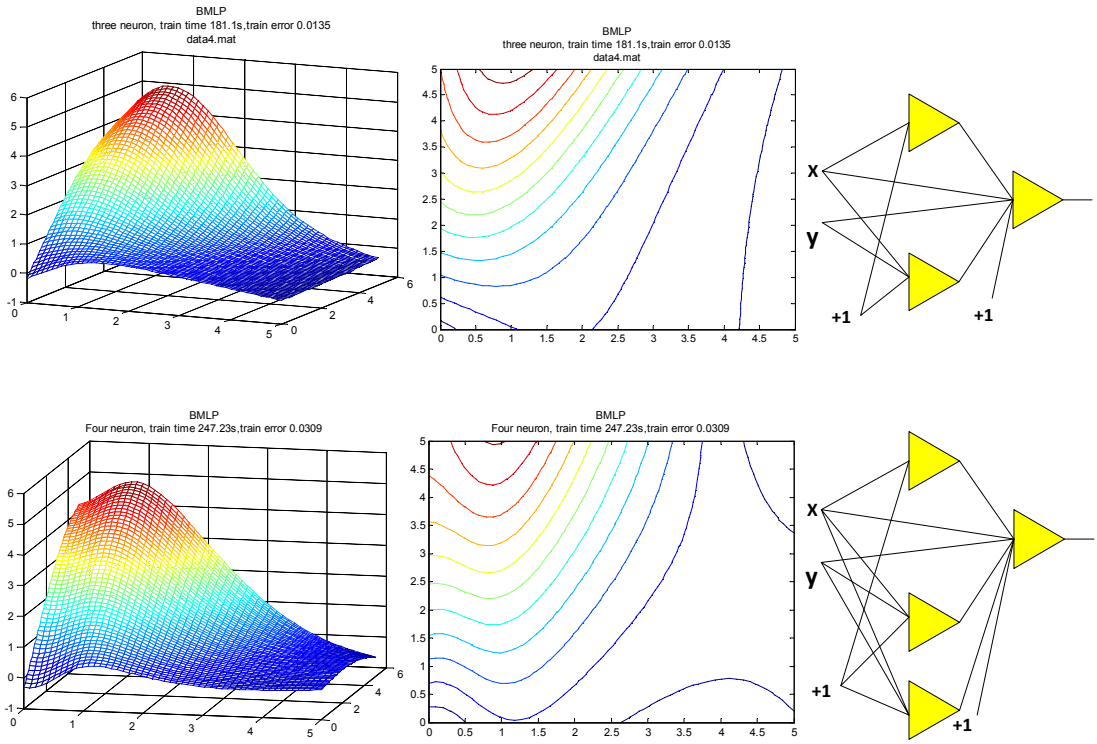
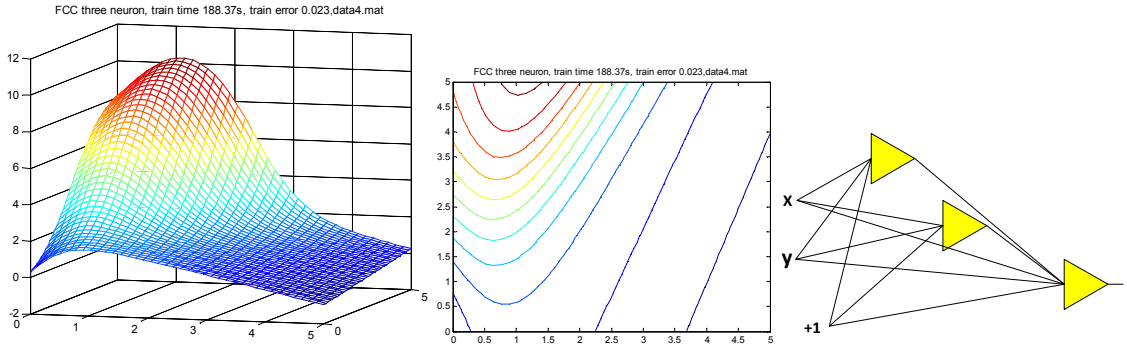


Figure 11 BMLP training architecture and result with three and four neurons

BMLP is a kind of bridged connection, meaning connection across layers. This architecture increases the difficulty in training. From the contour figures above, it seems that for BMLP the three neurons network performs better than four neurons network with PSO training method.

3. FCC



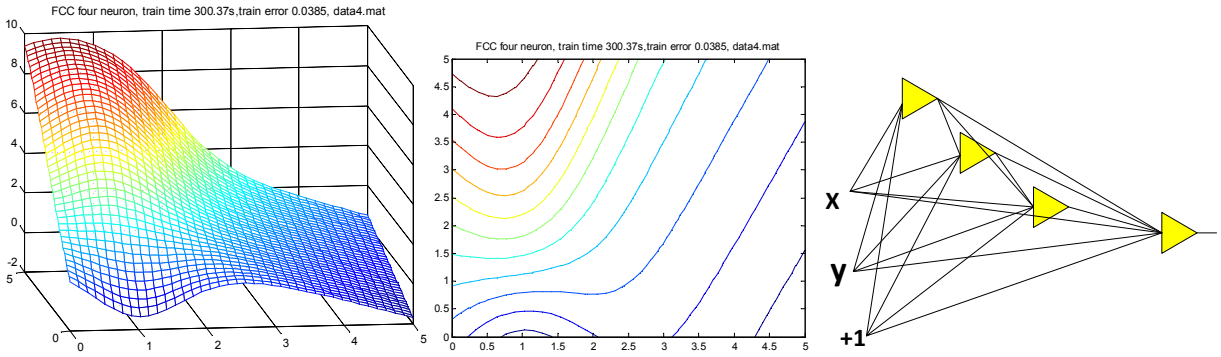


Figure 12 FCC training architecture and result with three and four neurons

FCC is the most powerful architecture among these three architecture, but it is also knows as difficult to train. Within 5000 iteration, the train error for 3 neuron is smaller than 4 neuron. By comparing with two contour figure of 3 and 4 neurons, one may notice that three-neuron network performs better, which is not consistent with derivative training algorithms training result used in [31]. There is one possibility that as the increase of the architecture, PSO is not powerful enough to train FCC architecture. That may imply the limitation of the derivative free algorithm.

Table 9 Comparison of three algorithm training result

	MLP		BMLP		FCC	
	Train time	**Train error**	**Train time**	**Train error**	**Train time**	**Train error**
3 neurons	179.9s	0.0174	181.1s	0.0135	188.37s	0.023
4 neurons	262.5s	0.0211	247.23s	0.0309	300.37s	0.0385

However, the PSO shows the potential in optimizing the weights of three neuron network architectures. No efficient training method can be used for all neural network architecture.

Chapter 5

Conclusion and Future Work

Gradient free optimization algorithms are often employed when gradient information of the function or the problem is not available or it is difficult to estimate. In this thesis, we take a review of five typical derivative free optimization algorithms and compare their performance based on a set of benchmark functions. Since the minimization of function has been widely used in different filed, such as control the robot arm movement, searching the best parameter setting of engineering process or training neural network weights, it is necessary to explore each algorithm's problem solving ability. Based on the analysis of table 4,5,6, we can conclude Genetic algorithm (GA) can deal with the global search problem better in a complex, vast surface by a series of natural behaviors, but the computing time limits its application in industrial filed, particularly for those require real-time optimization field. The Nelder-Mead's simplex method is simple and compact, easy to be employed. This method is favored due to its simplicity and convergence speed. from table 4, we can see Nelder-Mead's simplex works well with small number of variables (10 variables). The computing time and success rate result is very competitive with the rest methods. However, it is easy to fail in large scale problems of variables (15 variables in table 5), that is the limitation of Nelder-Mead simplex method. Two improved Nelder-Mead's simplex method, put forward by Pham and Wilamowski, greatly improve the original Nelder-Mead's simplex's convergence rate and success rate in large scale problems. Depend on the guide of quasi gradient, Nelder-Mead simplex Plus and Quasi Gradient method can converge faster than original vertex based searching method. Particle swarm optimization algorithm, as another stochastic method mimicking nature behavior, performs really well among in optimizing these benchmark functions. The computing time and success rate in optimization

high dimensional problems win over other method. Unlike genetic algorithm, it has few parameters to be adjusted. That is why it is used extensively across a wide range of field. This vector-based particle swarm performs its potential in high dimension problem solving. Based on section 4.3 and 4.2, we can see our version of particle swarm optimization can be a reliable and efficient method for solving high dimension problems if we choose proper initial population size, which is proved to be an important parameter in control the balance between the success rate and computing cost.

This thesis investigates five typical derivative free algorithms' potential in solving high dimension unconstrained nonlinear optimization problems. Improvement of SIM can converge fast when the objective function is relative smooth. Particle swarm optimization performs well for those problems which have many local minimums. A number of improvements in combining the advantage of simplex and particle swarm optimization as well as combining gradient method with particle swarm optimization are currently being explored by author. The combination of particle swarm optimization with other gradient methods would be worth of further studying.

Reference

[1] C. Kelley, *Solving Nonlinear Equations with Newton's Method*. Society for Industrial and Applied Mathematics, 2003.

[2] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Math. Comput.*, vol. 24, no. 111, pp. 647–656, 1970.

[3] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons, 2005.

[4] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[5] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS '95*, 1995, pp. 39–43.

[6] N. Pham, A. Malinowski, and T. Bartczak, "Comparative study of derivative free optimization algorithms," *Ind. Inform. IEEE Trans. On*, vol. 7, no. 4, pp. 592–600, 2011.

[7] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–313, Jan. 1965.

[8] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1 edition. Reading, Mass: Addison-Wesley Professional, 1989.

[10] J. Rowe, D. Whitley, L. Barbulescu, and J.-P. Watson, "Properties of Gray and Binary Representations," *Evol. Comput.*, vol. 12, no. 1, pp. 47–76, 2004.

[11] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Phys. Stat. Mech. Its Appl.*, vol. 391, no. 6, pp. 2193–2196, Mar. 2012.

[12] Á. Bűrmen, J. Puhan, and T. Tuma, "Grid Restrained Nelder-Mead Algorithm," *Comput. Optim. Appl.*, vol. 34, no. 3, pp. 359–375, Mar. 2006.

[13] W. M. Spears and K. A. De Jong, "An Analysis of Multi-Point Crossover," 1990.

[14] D. M. Tate and A. E. Smith, "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms," in *Proceedings of the 5th International Conference on Genetic Algorithms*, San Francisco, CA, USA, 1993, pp. 31–37.

[15] FOGA, *Foundations of Genetic Algorithms 1993 (FOGA 2)*. Morgan Kaufmann, 2014.

[16] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 116–121.

[17] R. Huang and T. C. Fogarty, "Adaptive classification and control-rule optimisation via a learning algorithm for controlling a dynamic system," in *, Proceedings of the 30th IEEE Conference on Decision and Control, 1991*, 1991, pp. 867–868 vol.1.

[18] J. Kennedy, "Particle Swarm Optimization," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer US, 2011, pp. 760–766.

[19] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, 2001, vol. 1, pp. 94–100.

[20] R. C. Eberhart and X. Hu, "Human tremor analysis using particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999, vol. 3.

[21] J. Kennedy, J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. Morgan Kaufmann, 2001.

[22] R. C. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, 2001, vol. 1, pp. 81–86.

[23] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *Evolutionary programming VII*, 1998, pp. 591–600.

[24] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999, vol. 3.

[25] F. Gao and L. Han, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters," *Comput. Optim. Appl.*, vol. 51, no. 1, pp. 259–277, 2012.

[26] "Learning representations by back-propagating errors." [Online]. Available: about:reader?url=http%3A%2F%2Fwww.nature.com%2Fnature%2Fjournal%2Fv323%2Fn6088%2Fabs%2F323533a0.html. [Accessed: 31-Jul-2015].

[27] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[28] B. M. Wilamowski, N. J. Cotton, J. Hewlett, and O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms," in *11th International Conference on Intelligent Engineering Systems, 2007. INES 2007*, 2007, pp. 127–132.

[29] B. M. Wilamowski, N. J. Cotton, O. Kaynak, and G. Dundar, "Method of computing gradient vector and Jacobean matrix in arbitrarily connected neural networks," in *IEEE International Symposium on Industrial Electronics, 2007. ISIE 2007*, 2007, pp. 3298–3303.

[30] B. M. Wilamowski, "Can computers be more intelligent than humans?," in *2011 4th International Conference on Human System Interactions (HSI)*, 2011, pp. 22–29.

[31] B. M. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 56–63, 2009.

Appendix 1: Vectorized Particle swarm optimization

```
function out=psovect(fnc,popsize,dim,low,up,epoch,times)
% This is a vectorized Particle Swarm Optimizaiton
% Syntax psovect('functionname',popsize,dim,low,up,epoch,times)
% popsize - It is the number of initial population size
% dim - It is the number of variables or the dimensions of the search space
% low - A number or column matrix depicting the lower boundary for each variable
% up - a numver or column matrix compose the upper boundary
% epoch - The max number of iterations stopping criterion
% times-the number of  testing times
% Example
% Type this in your command window :-
% low = -100;
% up =100;
% epoch=1e5;times=25;
% pso_nb(@dejong,100,10,low,up,epoch,times);

%code in detail

 succ=0;tsecond=0;
for j=1:times
   % Initialization and Poulation Generation
   tic;
   pop=GeneratePopulation1(popsize,dim,low,up);
   fitnesspop=feval(fnc,pop);
   PBest=pop;
   PBest_value=fitnesspop;
   [GBest_value,avain]=min(fitnesspop);
   GBest=repmat(pop(avain,:),popsize,1);
   v=zeros(popsize,dim);
   c1=2;                %selfadjust parameter
   c2=2;                %socioadjust parameter
   w=0.83;              %inertial weight

   for i=1:epoch

      % pso dynamic equation
      mp=1;
      vel = w.*v + mp.*((c1*rand).*(PBest-pop)+(c2*rand).*(GBest-pop));
      offspring = vel + pop;

      % BOUNDARY CONTROL
      offspring=BoundaryControl1b(offspring,low,up);

      % Cost Value of offspring
      fitness_offspring=feval(fnc,offspring);

      % updating the value of PBest and GBest
      ind =  fitness_offspring<PBest_value;
      PBest(ind,:)=offspring(ind,:);
      PBest_value(ind,:)=fitness_offspring(ind,:);
      [pop_best,ind1]=min(fitness_offspring);
      if pop_best<GBest_value
         GB=offspring(ind1,:);
         GBest=repmat(offspring(ind1,:),popsize,1);
%           disp(['minimum located at [',num2str(GB),']']);
         GBest_value=pop_best;
      end

      %  new co-ordinates and the velocity for next generation
      pop = offspring;
      v = vel;
      assignin('base','globalminimizer',GBest(1,:));
      assignin('base','globalminimum',GBest_value);
%       fprintf('iteration %5.0f  global best =%9.16f\n',i,GBest_value);
      if GBest_value<0.001
         succ=succ+1;
         break;
      end
   end
   disp(['minimum located at [',num2str(GB),']']);
%    disp(['minimum value is [',num2str(GBest_value),']']);
   tsecond=tsecond+toc;
end
succ_rate=succ/times;
ave_time=tsecond/times;
% disp(['success rate is  [',num2str(succ_rate),']']);
% disp(['average time is  [',num2str(ave_time),']']);
out=[dim,succ_rate,ave_time,popsize];
return

% Generate initial population funtion
function pop=GeneratePopulation1(popsize,dim,low,up)
pop=ones(popsize,dim);
[nbound]=size(low,2);
```

```
for i=1:popsize
   for j=1:dim
      if nbound==1
         pop(i,j)=rand*(up-low)+low;
      else
         pop(i,j)=rand*(up(j)-low(j))+low(j);
      end
   end
end
%  Boundary control function
function pop=BoundaryControl1b(pop,low,up)
% reach the boundary, generate new random
[popsize,dim]=size(pop);
for i=1:popsize
   for j=1:dim
      if pop(i,j)<low,
            pop(i,j)=rand*(up-low)+low;
      end
      if pop(i,j)>up,
           pop(i,j)=rand*(up-low)+low;
      end
   end
end
return
```

Appendix 2: Quasi Gradient Method with quasi plane

```
function out=nelder_mead_ndmd2(obj,x0,d_SIM,df_min,ite_max,times)
%assertation:
% this code is originally programmed by Pham, modified and used as comparsion in experiment
% INPUT ARGUMENTS:
%   nelder_mead_ndmd2(@testf1,[100,100],1,1e-4,2e2,100)
%   obj        - Handle of objective function.
%   x0         - Initial depot point.
%   d_SIM   - Size of initial simplex.
%   df_min  - Minimum improvement required for termination.
%   ite_max - Desired number of iterations.
%  OUTPUT ARGUMENTS:
%   BEST    - Location of baest solution.
%   f_BEST  - Best value of the objective found.
%   SIMPLEX - Matrix conatining final simplex
format long;
tavg_ite=0;
tsecond=0;
second=0;
succ_time=0;
avg_ite=0;
avg_time=0;
avg_error=0;
average_min=0;
% Initialize parameters and create simplex
for itee=1:times,   %training timesa=1;
   tic;
   alpha=1;
   a=1;
   b=2;
   c=0.5;
   n=length(x0);
   mo=zeros(1,n);
   mu=0.1;
   X0=ones(n,1)*x0;
   SIMPLEX=[X0+diag(d_SIM*(rand(1,n)));x0]; % create simplex vertices
   f(n+1)=0;
   f_mid(n)=0;
   mid=zeros(n);
   for init=1:n+1
      f(init)=feval(obj,SIMPLEX(init,:));
   end
   init=0;
   SIMPLEX(:,end+1)=f';
   SIMPLEX=sortrows(SIMPLEX,n+1);    %sort row depending of value of f in ascending order;
   f=SIMPLEX(:,end)';
   SIMPLEX(:,end)=[];
   % Simplex Code
   for ite=1:ite_max,
      Pb=sum(SIMPLEX(1:n,:))/n;       %calculate the centroid P_ of points with i#h
      Ps=(1+a)*Pb-a*SIMPLEX(end,:);    %calculate reflection point of Ph:Ps
      f_Ps=feval(obj,Ps);
      Pss=(1-b)*Pb+b*Ps;    %calculate P** by expansion
      f_Pss=feval(obj,Pss);
      if f_Ps>f(1)
         % using hyper plane equation
         I=SIMPLEX(:,1:n);
         A=ones(1,n+1)';
         A(:,2:n+1)=I;
         B=f';
         P=pinv(A)*B;
         grad=P';
         Gs=SIMPLEX(1,:)-alpha*grad(1,2:n+1);
         % Calculate reflected point
         P3=(1+a)*SIMPLEX(1,:)-SIMPLEX(end,:);
         P1=SIMPLEX(1,:);
         P2=Gs;
         PP=(P3-P1).*(P2-P1);
         u=sum(PP)/sum((P2-P1).^2);
         Gs=P1+u*(P2-P1);
         f_Gs=feval(obj,Gs);
         if f_Gs<f_Ps
            Ps=Gs;            %new reflected point
            f_Ps=f_Gs;
            Pb=SIMPLEX(1,:);
            Pss=(1-b)*SIMPLEX(1,:)+b*Ps;   %calculate P** by expansion
            f_Pss=feval(obj,Pss);
```

```
            end
        end
        if f_Ps<f(1)    %f(P*)<f(l)
            if f_Pss<f(1)    %f(P**)<f(l)
                SIMPLEX(end,:)=Pss;      %replace Ph by P**
                f(end)=f_Pss;
            else
                SIMPLEX(end,:)=Ps;       %replace Ph by P*
                f(end)=f_Ps;
            end
        else
            check=0;
            for i=1:n,
                if f_Ps>f(i)      % f_P*>f_i and i#h
                    check=1;
                    break;
                end
            end
            if check==0
                SIMPLEX(end,:)=Ps;       %replace Ph by P*
                f(end)=f_Ps;
            else
                if f_Ps>f(end)    %f_P*>f_h
                    Pss=c*SIMPLEX(end,:)+(1-c)*Pb; %calculate P** by expansion
                    f_Pss=feval(obj,Pss);
                    if f_Pss>f(end)    %f(P**)>f(h)
                        for i=1:n+1
                            SIMPLEX(i,:)=(SIMPLEX(i,:)+SIMPLEX(1,:))/2; %replace all Pi' by (Pi+Pl)/2
                            f(i)=feval(obj,SIMPLEX(i,:));
                        end
                    else
                        SIMPLEX(end,:)=Pss;      %replace Ph by P**
                        f(end)=f_Pss;
                    end
                else
                    SIMPLEX(end,:)=Ps;       %replace Ph by P*
                    f(end)=f_Ps;
                end
            end
        end
        SIMPLEX(:,end+1)=f';
        SIMPLEX=sortrows(SIMPLEX,n+1);
        f=SIMPLEX(:,end)';
        SIMPLEX(:,end)=[];
        error(ite)=f(1);
        t(ite)=ite;
        if f(1)<df_min,
            succ_time=succ_time+1;
            avg_ite=avg_ite+ite;
            avg_time=avg_time+1;
            avg_error=avg_error+f(1);
            second=second+toc;
            break;
        end
    end;
    % display the result
    succ_rate=succ_time/times;
    BEST=SIMPLEX(1,:);
    f_BEST=f(1);
    average_min=average_min+f_BEST;
    tavg_ite=tavg_ite+ite;
    tsecond=tsecond+toc;
end
    avg_iteration=avg_ite/avg_time;
    avg_errors=avg_error/avg_time;
    avg_second=second/avg_time;
    tavg_iteration=tavg_ite/times;
    avg_minimum=average_min/times;
    avg_tsecond=tsecond/times;
    out=[succ_rate,avg_tsecond,d_SIM];
    disp(['Average Iteration = ',num2str(avg_iteration),])
    disp(['Average Error = ',num2str(avg_errors),])
    disp(['Average second = ',num2str(avg_second),])
    disp(['tAverage Iteration = ',num2str(tavg_iteration),])
    disp(['tAverage Minimum = ',num2str(avg_minimum),])
    disp(['tAverage second = ',num2str(avg_tsecond),])
return
```