EGE UNIVERSITY GRADUATE SCHOOL OF APPLIED

AND NATURAL SCIENCES

(MASTER OF SCIENCE THESIS)

DIFFERENTIAL EVOLUTION FOR OPTIMIZATION OF NONLINEAR CHEMICAL PROCESSES

Ercüment DENİZ

Chemical engineering department
Department Code: 603.02.00
Presentation Date: June 2008

Supervisor: Prof. Dr. Beno KURYEL

Bornova – IZMIR

V

ÖZET

DOĞRUSAL OLMAYAN KİMYASAL SÜREÇLERİN OPTİMİZASYONUNDA DİFERANSİYEL EVRİM YAKLAŞIMI

DENİZ, Ercüment

Yüksek Lisans Tezi, Kimya Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Beno KURYEL

Haziran 2008, 159 sayfa

Son yıllarda, evrimsel algoritmalar, birçok mühendislik disiplininde karşılaşılan multimodal doğrusal olmayan problemlerin optimal çözümlerinde ilgi çekmektedir. Evrimsel algoritmalardan birisi olan diferansiyel evrim (DE), yeni bir optimizasyon yöntemi olup, türevlenemeyen, doğrusal olmayan multimodal amaç fonksiyonlarının ele alınmasında etkin olmaktadır. Daha önceki çalışmaların gösterdiği gibi, diferansiyel evrim, verimli, etkin ve dolaysız bir evrimsel optimizasyon yöntemidir. Bu yöntemi daha etkin kılabilmek için modifiye edilmiş diferansiyel evrimi gündeme getirmek gerekmektedir. Yakınsama özellikleri de geliştirilmiş bu yaklaşımda bilgisayar desteği de kolaylaşmaktadır. Bu çalışmada, geliştirilecek bilgisayar programı, gerçek yaşamdan süreçlere uygulanacaktır.

Anahtar Sözcükler: Diferansiyel evrim, doğrusal olmayan problem çözümlemesi.

VII

ABSTRACT

DIFFERENTIAL EVOLUTION FOR OPTIMIZATION OF NONLINEAR CHEMICAL PROCESSES

DENİZ, Ercüment

MSc in Chemical Engineering

Supervisor: Prof. Dr. Beno KURYEL

June 2008, 159 pages

In recent years, evolutionary algorithms are gaining popularity for finding the optimal solution of nonlinear multimodal problems encountered in many engineering disciplines. Differential evolution (DE), one of the evolutionary algorithms, is a novel optimization method capable of handling nondifferentiable, nonlinear and multimodal objective functions. Previous studies have shown that differential evolution is an efficient, effective and robust evolutionary optimization method. This thesis aims to introduce a modification to original DE that enhances the probability without compromising on solution quality. The modified differential evolution (MDE) algorithm utilizes only one set of population as against two sets in original DE at any given point of time in a generation. Application to real processes is targeted in this work

Keywords: Differential evolution, nonlinear problem analysis.

XI

ACKNOWLEDGEMENT

I would like to thank Professor Beno Kuryel for suggesting me study such an interesting and necessary subject. I am deeply indebted to him from whose assurance, stimulating suggestions and encouragement helped me in all the time of research for and writing of this thesis. I owe great deal to him since the beginning of my first study.

My former colleagues from the Department of Chemical Engineering supported me in my research work. I want to thank them for all their moral support and interest.

I would like to give my special thanks also to my family, especially to my mother whose patient and support enabled me to complete this work.

Ercüment DENİZ June 2008, Bornova

XI

CONTENTS

XII

CONTENTS (continued)

XIII

CONTENTS (continued)

XIV

CONTENTS (continued)

XV

LIST OF FIGURES

XVI

LIST OF FIGURES (continued)

XVII

LIST OF FIGURES (continued)

XVIII

LIST OF FIGURES (continued)

XIX

LIST OF FIGURES (continued)

XX

LIST OF TABLES

1.0 INTRODUCTION

DE grew out of Kenneth Price's attempts to solve the [1]Chebyshev polynomial fitting problem that had been posed to him by Rainer Storn. A breakthrough happened when Kenneth came up with the idea of using vector differences for perturbing the vector population. Since this seminal idea a lively discussion between Kenneth and Rainer and endless ruminations and computer simulations on both parts yielded many substantial improvements which make DE the versatile and robust tool it is today.

Genetic Annealing, developed by Price, was the beginning for the DE algorithm. The first paper about Genetic Annealing was published in October 1994 issue of *Dr. Dobb's Journal*. It was a population-based combinatorial algorithm that realizes an annealing criterion via thresholds driven by the average performance of the population. Soon after this development, K. Price was contacted by R.Storn, who was interested in solving the Tchebychev polynomial fitting problem by

[1] In mathematics the **Chebyshev polynomials**, named after Pafnuty Chebyshev, are a sequence of orthogonal polynomials which are related to de Moivre's formula and which are easily defined recursively, like Fibonacci or Lucas numbers. Chebyshev polynomials are important in approximation theory because the roots of the Chebyshev polynomials of the first kind, which are also called Chebyshev nodes, are used as nodes in polynomial interpolation. The resulting interpolation polynomial minimizes the problem of Runge's phenomenon and provides an approximation that is close to the polynomial of best approximation to a continuous function under the maximum norm. This approximation leads directly to the method of Clenshaw-Curtis quadrate

Genetic Annealing. After some experiments Price modified the algorithm using floating-point instead of bit-string encoding and arithmetic vector operations instead of logical ones. These recasts have changed Genetic Annealing from a combinatorial into a continuous optimizer.

In this way, Price discovers the procedure of differential mutation. Price and Storn detected that differential mutation combined with discrete recombination and pair-wise selection is not in need of an annealing factor. Hence, annealing mechanism had been finally removed and thus obtained algorithm started the era of Differential Evolution.

For the first time Differential Evolution was described by Price and Storn in the ICSI technical report ("Differential Evolution -- a simple and efficient adaptive scheme for global optimization over continuous spaces, 1995"). One year later, the success of DE was demonstrated at the First International Contest on Evolutionary Optimization in May of 1996, which was held in conjunction with the 1996 IEEE International Conference on Evolutionary Computation. The algorithm won third place on proposed benchmarks.

Inspired by results, Price and Storn write an article for *Dr. Dobb's Journal* ("Differential Evolution: A simple evolution strategy for fast optimization"), which was published in April 1997. Also, their article for the *Journal of Global Optimization* ("Differential Evolution -- A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces") was published in December 1997. These papers introduce DE to a large international public and demonstrate the advantages of DE over

the other heuristics. Very good results had been shown on a wide variety of benchmarks.

Furthermore, Price presents DE at the Second International Contest on Evolutionary Optimization in 1997 (Differential Evolution vs. the Functions of the Second ICEO). There, DE was one of the bests among emulous algorithms. And finally, two years later, in 1999, he summarized the algorithm in the compendium *"New Ideas in Optimization"*.

R. Storn had been concentrating on various DE applications and had published his web-site containing source codes and many useful links. In 1998, J. Lampinen set up the official bibliography site, which furnishes all materials and also some links on DE dated from 1995 up to 2002.

Before the differential evolution algorithm is to find acceptance, some population based optimization algorithms have been used. And some of these algorithms have been the basis of D.E. Some of them are:

Ant colony optimization (ACO) uses many ants (or agents) to traverse the solution space and find locally productive areas. While usually inferior to genetic algorithms and other forms of local search, it is able to produce results in problems where no global or up-to-date perspective can be obtained, and thus the other methods cannot be applied.

Bacteriologic Algorithms (BA) inspired by evolutionary ecology and, more particularly, bacteriologic adaptation. Evolutionary ecology is the study of living organisms in the context of their environment, with the aim of discovering how they adapt. Its basic concept is that in a heterogeneous environment, you can't find one individual that fits the whole environment. So, you need to reason at the population level. BAs have shown better results than GAs on problems such as complex positioning problems (antennas for cell phones, urban planning, and so on) or data mining.

Cross-entropy method The Cross-entropy (CE) method generates candidates' solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.

Cultural algorithm (CA) consists of the population component almost identical to that of the genetic algorithm and, in addition, a knowledge component called the belief space.

Evolution strategies (ES, see Rechenberg, 1971) evolve individuals by means of mutation and intermediate and discrete recombination. ES algorithms are designed particularly to solve problems in the real-value domain. They use self-adaptation to adjust control parameters of the search.

Evolutionary programming (EP) involves populations of solutions with primarily mutation and selection and arbitrary representations. They use self-adaptation to adjust parameters, and can include other variation operations such as combining information from multiple parents.

Extremal optimization (EO) Unlike GAs, which works with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to be assigned a quality measure ("fitness"). The governing principle behind this algorithm is that of emergent improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is decidedly at odds with a GA that selects good solutions in an attempt to make better solutions.

Gaussian adaptation (normal or natural adaptation, abbreviated NA to avoid confusion with GA) is intended for the maximization of manufacturing yield of signal processing systems. It may also be used for ordinary parametric optimization. It relies on a certain theorem valid for all regions of acceptability and all Gaussian distributions. The efficiency of NA relies on information theory and a certain theorem of efficiency. Its efficiency is defined as information divided by the work needed to get the information. Because NA maximizes mean fitness rather than the fitness of the individual, the landscape is smoothed such that valleys between peaks may disappear. Therefore it has a certain “ambition” to avoid local peaks in the fitness landscape. NA is also good at climbing sharp crests by adaptation of the moment matrix, because NA may maximize the disorder (average information) of the Gaussian simultaneously keeping the mean fitness constant.

Genetic programming (GP) is a related technique popularized by John Koza in which computer programs, rather than function parameters, are optimized. Genetic programming often uses tree-based internal data

structures to represent the computer programs for adaptation instead of the list structures typical of genetic algorithms.

Grouping Genetic Algorithm (GGA) is an evolution of the GA where the focus is shifted from individual items, like in classical GAs, to groups or subset of items. The idea behind this GA evolution proposed by Emanuel Falkenauer is that solving some complex problems, a.k.a. clustering or partitioning problems where a set of items must be split into disjoint group of items in an optimal way, would better be achieved by making characteristics of the groups of items equivalent to genes. These kinds of problems include Bin Packing, Line Balancing, Clustering with respect to a distance measure, Equal Piles, etc., on which classic GAs proved to perform poorly. Making genes equivalent to groups implies chromosomes that are in general of variable length, and special genetic operators that manipulate whole groups of items. For

Interactive evolutionary algorithms are evolutionary algorithms that use human evaluation. They are usually applied to domains where it is hard to design a computational fitness function, for example, evolving images, music, artistic designs and forms to fit users' aesthetic preference.

Memetic algorithm (MA), also called hybrid genetic algorithm among others, is a relatively new evolutionary method where local search is applied during the evolutionary cycle. The idea of memetic algorithms comes from memes, which–unlike genes–can adapt themselves. In some problem areas they are shown to be more efficient than traditional evolutionary algorithms.

Simulated annealing (SA) is a related global optimization technique that traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation that lowers fitness is accepted probabilistically based on the difference in fitness and a decreasing temperature parameter. In SA parlance, one speaks of seeking the lowest energy instead of the maximum fitness. SA can also be used within a standard GA algorithm by starting with a relatively high rate of mutation and decreasing it over time along a given schedule.

Stochastic optimization is an umbrella set of methods that includes GAs and numerous other approaches.

Tabu search (TS) is similar to Simulated Annealing in that both traverse the solution space by testing mutations of an individual solution. While simulated annealing generates only one mutated solution, tabu search generates many mutated solutions and moves to the solution with the lowest energy of those generated. In order to prevent cycling and encourage greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains elements of the tabu list, which is updated as the solution traverses the solution space.

Many engineering optimization problems contain multiple optimal solutions, among which one or more may be the absolute minimum or maximum solutions. These absolute optimum solutions are known as global optimal solutions and other optimum solutions are known as local optimal solutions. Ideally, we are interested in the global optimal solutions because they correspond to the absolute optimum

objective function value. Most of the traditional optimization algorithms based on gradient methods have the possibility of getting trapped at local optimum depending upon the degree of non-linearity and initial guess. Unfortunately, none of the traditional algorithms are guaranteed to find the global optimal solution, but population based search algorithms are found to have a better global perspective than the traditional methods (Onwubolu & Babu, 2004). In the recent past, non-traditional search and optimization techniques based on natural phenomenon (evolutionary computation) such as genetic algorithms (GA) (Holland, 1975; Goldberg, 1989), evolution strategies (ESs) (Schwefel, 1981), simulated annealing (SA) (Kirkpatrick et al., 1983), and differential evolution (DE) (Price & Storn, 1997) to name a few, have been developed to overcome the problems. Among their advantages are:

(1) They do not require the objective function to be continuous and/or differentiable,

(2) They do not require extensive problem formulation (in case of traditional methods such as Integer programming, geometric programming, branch and bound methods, etc., special mathematical formulation is required for solving a problem),

(3) They are not sensitive to starting point,

(4) They usually do not get stuck into so called local optima,

(5) They are more likely to find out a function's true global optimum.

These advantages enhance their application to various fields. They have been successfully applied in many engineering design problems (Androulakis & Venkatasubramanian, 1991; Angira & Babu, 2003; Babu, 2004; Babu & Angira, 2002a,b; Babu, Pallavi, & Syed Mubeen, 2005; Babu&Sastry, 1999; Chiou, Chang,&Su, 2004; Chiou &Wang, 1999; Deb, 1996, 2001; Hendtlass, 2001; Lee, Han, & Chang, 1999; Lu &Wang, 2001; Storn, 1995, etc.) to name a few. Recently, Onwubolu and Babu (2004) compiled new techniques and their applications to various disciplines of engineering and management.

As stated in, the key element distinguishing DE from other population- based techniques is differential mutation. The initial set of **strategies** realizing differential mutation was proposed by Storn and Price. The first attempt to guide differential mutation was presented by Price in, where “semi directed” mutation was realized by a special pre selection operation. Later, Price analyzed the strategies and noted that the strategy may consist of differential mutation and arithmetic crossover. This, in turn, gives the different dynamic effects of search.

The ideas of “directions” were spontaneously grasped by H.-Y. Fan and J.Lampinen. In 2001, they proposed alternations of the classical strategy (the first strategy suggested by Price) with a triangle mutation scheme and, in 2003, alternations with a weighted directed strategy, where they used two difference vectors. These methods give some improvements, but it is also necessary to note that the percentage of using novel strategies is quite moderate.

From this point on, there is a unique formula that describes all the strategies and clearly reflects the fundamental principle of DE. The strategies were divided into four groups. Each of the groups was associated with a certain type/behavior of search: random, directed, local, and hybrid. Thorough investigations and test results of some strategies were published in. The operation realizing a strategy in the DE algorithm was called *differentiation*.

Now consider a **crossover** operation. For DE two types of combinatorial crossovers were implemented: binary and exponential ones. The superiority of each crossover over the other cannot be uniquely defined. As for a **selection** operation, the pair wise selection, also so-called "greedy" selection or elitist selection is steadily used in the algorithm.

The next stage was the introduction of **mixed variables**. In 1999, I. Zelinka and J. Lampinen described a simple and, at the same time, efficient way of handling simultaneously continuous, integer, and discrete variables. They applied this method to design engineering problems. The obtained results outperformed all the other mixed variables methods used in engineering design.

Now pass to **constraints**. In order to handle boundary constraints two solutions can be implemented: (1) reinitialization and (2) periodic mode (or shifting mechanism). For other constraints (mostly nonlinear

functions) penalty methods are used as well as the modification of selection rules, first reported for DE, in 2001, by Lampinen and later, in 2004, by Coello et al.

Preset parameter bounds define the domain from which the *Np* vectors in this initial population are chosen (Fig. 1.1). Each vector is indexed with a number from 0 to $Np - 1$.

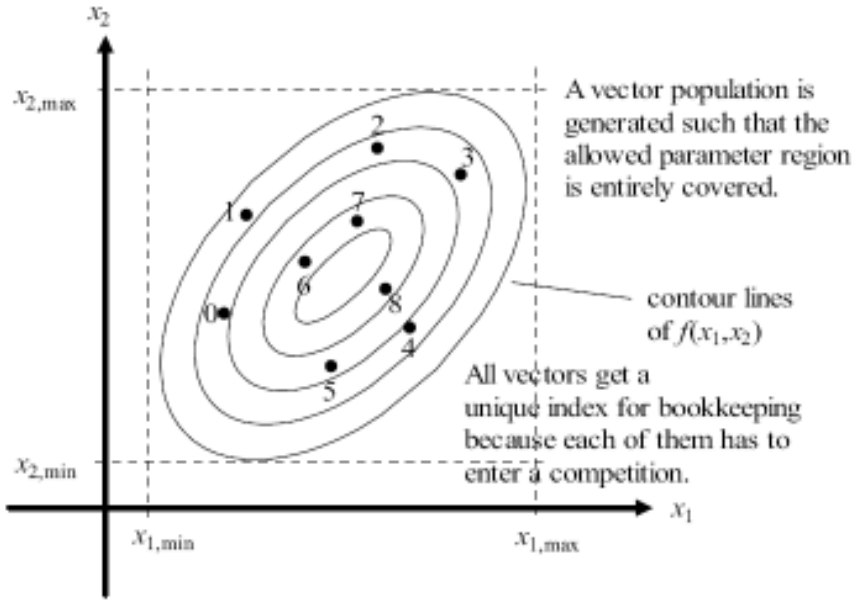


Figure 1.1 initializing the DE population

Like other population- based methods, DE generates new points that are perturbations of existing points, but these deviations are neither reflections like those in the CRS and Nelder–Mead methods, nor samples from a predefined probability density function, like those in the ES. Instead, DE perturbs vectors with the scaled difference of two randomly

selected population vectors (Fig. 1.2). To produce the trial vector, **u**0, DE adds the scaled, random vector difference to a third randomly selected population vector (Fig. 1.3). In the selection stage, the trial vector competes against the population vector of the same index, which in this case is number 0. Figure 1.4 illustrates the select-and-save step in which the vector with the lower objective function value is marked as a member of the next generation. Figures 1.5–1.6 indicate that the procedure repeats until all *Np* population vectors have competed against a randomly generated trial vector. Once the last trial vector has been tested, the survivors of the *Np* pair wise competitions become parents for the next generation in the evolutionary cycle.

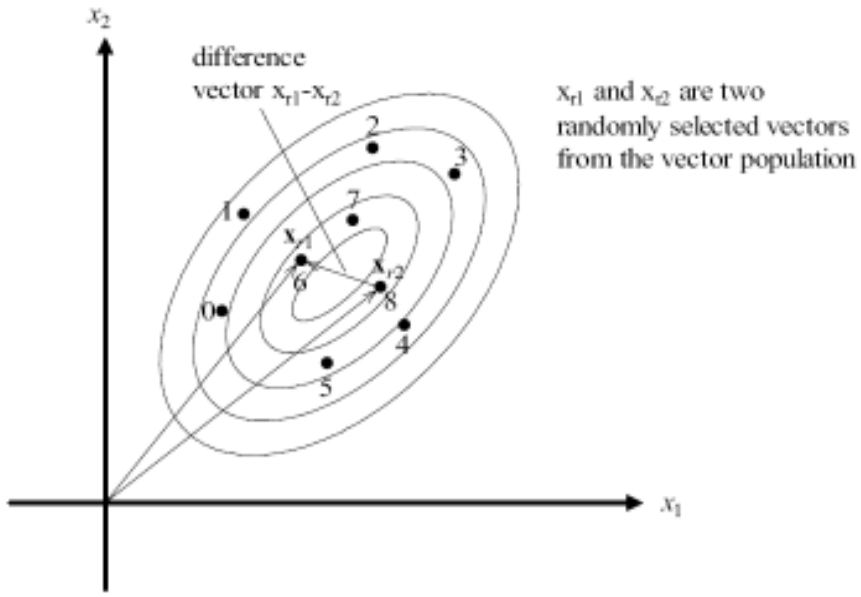


Figure 1.2 Generating the perturbation: $x_{r1} - x_{r2}$

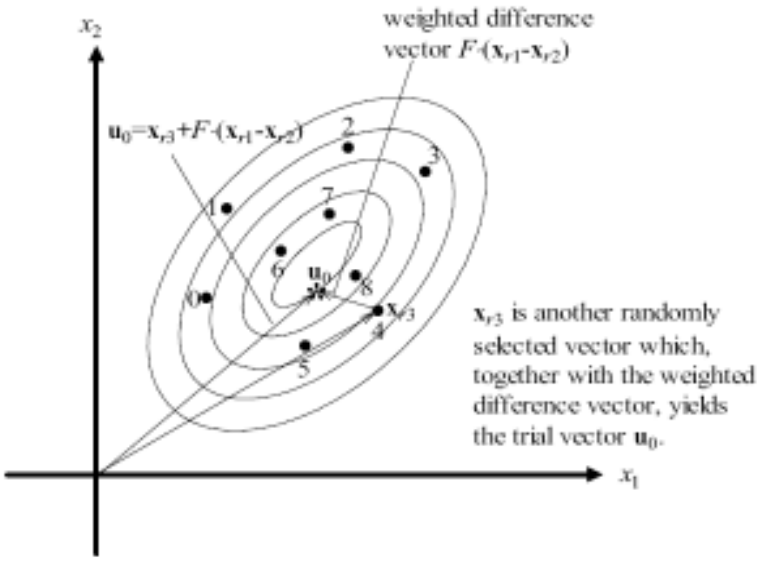


Figure 1.3 Mutations

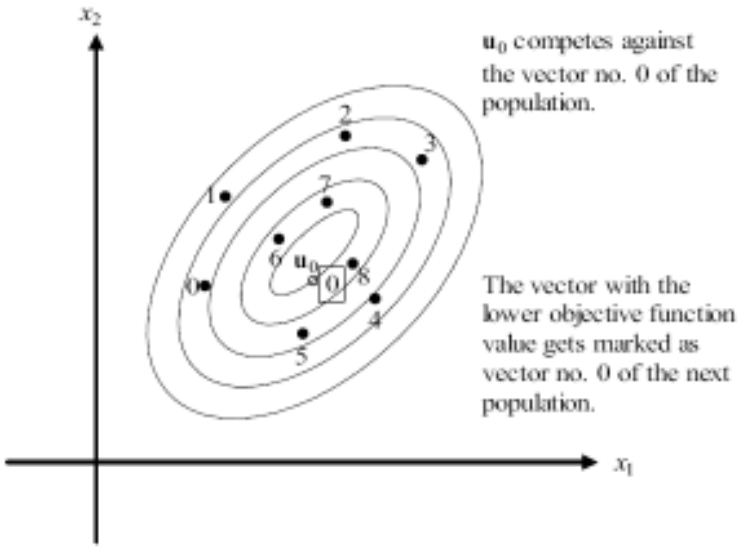


Figure 1.4 Selections. Because it has a lower function value, u0 replaces the vector with index 0 in the next generation

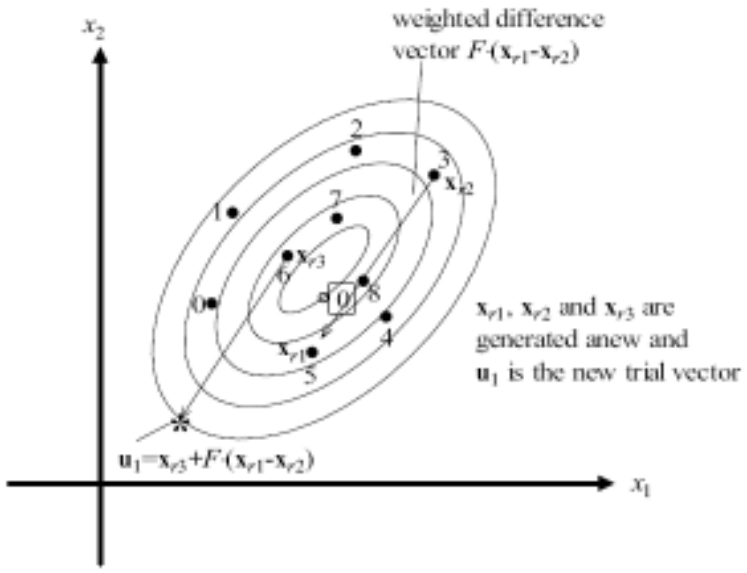


Figure 1.5 A new population vector is mutated with a randomly generated perturbation.

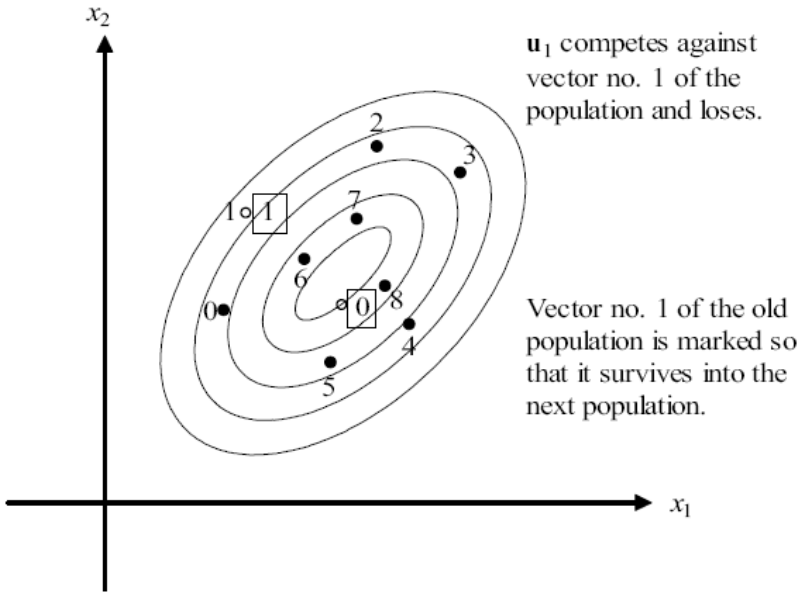


Figure 1.6 Selections. This time, the trial vector loses.

Even though the scheme described above already works remarkably well, DE's performance can be improved and its methodology adapted to a wide variety of optimization scenarios.

In the thesis of following chapters provide additional insight into how and why DE works, including a convergence proof, performance comparisons between the modification of differential evolution, practical applications, and Visual Basic (version 6) programmed for solving real-world tasks.

In the thesis, four different groups of problem were taken a matter in hand. First of all, the unconstraint benchmark test problems were investigated. The algorithms were run 10 times on each of the test problems to determine the percentages of success (ps). Consequently the differential evolution algorithm gets a 100 percentages of success.

Then the second group, which is the more real problems, constraint benchmark problems was investigated. And also these group problems gave excellent result, percents of success was 100.

The chemical engineering problems were investigated separately. Specially, highly nonlinear systems were selected for testing the program efficiency. Nearly all problems gave a good percent of success.

Finally Regression problems were taken into account. In this group of problems were also excellent results.

And also some modifications were done on the Differential Evolution algorithm. Especially, we looked around mutation part of the algorithm. Hence four different modifications were tested. These shown that, There is no a Modified Differential Evolution algorithm, which is the best result for all type of problems. Each problem was different respond for each modification. And these will explain following chapters of in the thesis.

2.0 BASIC OF DIFFERENTIAL EVOLUTION ALGORYTM

2.1 Initialization

In order for DE to work, the initial population must be distributed throughout the problem space. One-point optimizers do not require this initial diversity and even the (1, λ)-ES begins with a single point. If, however, DE is initialized with *Np* replicas of a single vector, uniform crossover and differential mutation will only clone more replicas. Consequently, DE requires a predefined *probability distribution function*, or PDF, to seed the initial population. When specifying an initial distribution, steps must be taken to ensure that its scale sufficiently broad.

2.3.1 Initial Bounds

As a matter of convenience, test function parameters are often initialized with values that are constrained to lie between a single set of upper and lower bounds. By contrast, bounds for parameters that define real-world objective functions are seldom equal, often because the parameters they delimit correspond to different physical or mathematical entities. In many cases, the existence of natural physical limits or logical constraints makes prescribing bounds for each parameter straightforward. For example, ordinary optical glass can never have an index of refraction less than or equal to 1, nor can a gear have less than one tooth. In cases like these where parameter limits are inviolable, initialization bounds should not

only delimit the initial population, but also constrain the subsequent search.

Far Initialization

When parameters exhibit no obvious limits, their upper and lower bounds, *bj*, U and *bj*, L, respectively, should be set so that the initial bounding box they define encompasses the optimum. If the optimum's general location is uncertain, then the possibility exists that it lies outside the initial bounding box. Figure 2.1 shows an example of *far initialization* in which the upper parameter limit has been reduced to the point where the initial bounding box no longer contains the optimum, **x***. In cases of far initialization, bounds on otherwise unconstrained parameters must be ignored once the population has been initialized so that DE can explore beyond the initial bounding box.

Table 2.1 records the effect that far initialization has on DE's ability to discover the optima of ten common test function. Although each function has a different set of initialization bounds, in each case, these bounds define a *D*-dimensional box that encloses the function's global optimum.

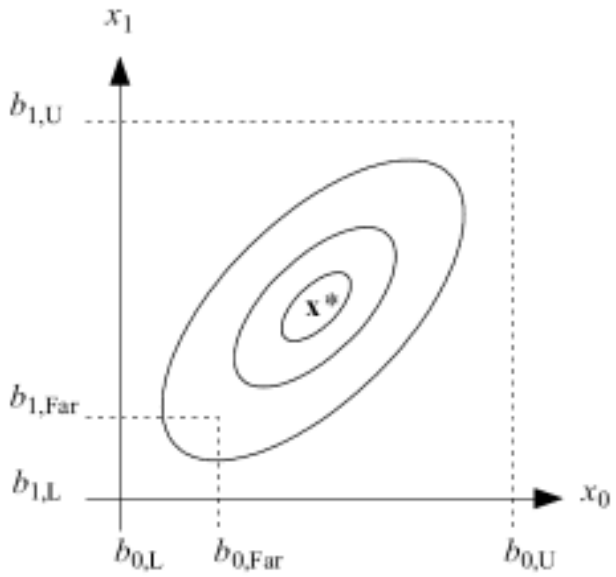


Figure 2.1 Far initialization shrinks the initial bounding box so no longer contains the optimum x^*

For the results in Table 2.1, each parameter was initialized with a uniformly distributed random value from within a range that has been reduced by a factor, *h*, when compared to the originally prescribed bounds:

$$x_{j,i,0} = b_{j,L} + h * rand_j(0,1) * (b_{j,U} - b_{j,L}) \qquad (2.1)$$

After far initializing the population with the given value of *h*, bounds were relaxed to their normal values to constrain the subsequent search. For each of the functions in Table 2.1, the initial bounding box encloses the optimum when $h = 1$. Setting $h \leq 0.1$ far initializes the population by restricting it to a corner of the original bounding box where it cannot

surround the optimum. Table 2.1 reports the average number of function evaluations taken to find a point whose objective function value differs from the optimum objective function value by less than a preset minimum. Finding such a point within the maximum allowed number of generations constitutes a success; otherwise, the trial is considered to be a failure. Only "successes" contribute to the results in Table 2.1. The fraction of successful trials, P, records the impact of failures. Results are 100-trial averages obtained using classic DE with $F = Cr = 0.9$ and $r0 \neq r1 \neq r2 \neq i$ (distinct indices).

Table 2.1 The effects of far initializing DE with a uniformly random Population.

FUNCTION	D	NP	H=1		H=0.1		H=0.01	
			Evals.	P	Evals.	P	Evals.	P
Sphere	10	30	30,994	1	31,514	1	31,722	1
Ridge	10	30	48,520	1	48,825	1	48,820	1
Rosenbrock	10	30	59,643	1	59,721	1	60,315	1
Chebyshev	9	30	69,522	1	72,211	1	71,068	1
Ackley	10	30	48,385	1	49,853	0.9	-	0
Rastrigin	5	100	59,840	1	60,199	1	-	0
Schwefel	5	100	16,245	1	22,432	0.25	-	0
Griewangk	5	100	19,420	0.98	19,555	0.99	19,492	0.99
Langerman	5	100	38,405	0.98	37,196	0.54	34,873	0.21
Michalewicz	5	100	27,749	1	29,291	0.95	32,061	0.96

As Table 2.1 shows, far initialization's effect on the sphere, ridge, Rosenbrock, Chebyshev, Michalewicz and Griewangk functions is minimal. In most cases, far initialization penalizes these six functions with a very slight increase in the average number of function evaluations and a very slight decrease in the estimated probability of success. For both the sphere and ridge functions, this result is not surprising. Both functions are uni-modal and convex, so neither poses obstacles to the population's expansion toward the minimum. Rosenbrock's function is also uni-modal, but unlike the sphere it is nonconvex. At least in the case of Rosenbrock's function, non-convexity does not impede DE's ability to locate the minimum when far initialized.

Unlike the sphere, ridge or Rosenbrock functions, the remaining functions in Table 2.1 are all multi-modal. Optimal parameter values for the Chebyshev function vary greatly in magnitude and restricting initial values to a small range means that some parameter values must inflate many orders of magnitude to be on par with their optimal values. Table 2.1 shows that except for a slight increase in the number of function evaluations, diminishing the value of h did not significantly impact DE's ability to converge on the Chebyshev optimum. Similarly, far initialization did not significantly affect DE's performance on either Michalewicz's or Griewangk's function. DE became unreliable, however, when far initializing Langerman's function and failed altogether on the Ackley, Rastrigin and Schwefel functions once h = 0.01. For these highly multi-modal functions, the entire initial population can land inside a single, non-optimal, local basin of attraction when h

becomes too small. If competing basins are sufficiently far apart, then classic DE cannot generate difference vectors large enough to escape the local basin. Thus, it is important to use a bounding box of sufficient size when initializing multi-modal functions with a uniform random distribution.

2.2 Degenerate Vector Combinations

If indices are chosen without restrictions, there is no guarantee that i, r_0, $r1$ and $r2$ will be distinct. When these indices are not mutually exclusive, DE's novel trial vector-generating strategy reduces to uniform crossover only, duplication of the base vector, an alternative form of recombination, or mutation only. These possibilities are explored below, first by looking at the three degenerate combinations of indices that comprise the mutant vector, $r0$, $r1$ and $r2$, and then by considering the three interactions of the target index, i , with the mutant indices.

Degenerate Combinations of Mutant Indices: r0, r1, r2

$r1 = r2$: No Mutation. If $r1 = r2$, then the differential formed by the corresponding vectors will be zero and the base vector, $\mathbf{x}_{r0,g,}$ will not be mutated:

$$r1 = r2(= r0): \qquad v_{i,g} = x_{r0,g} \tag{2.2}$$

When indices are chosen without restrictions, $r1$ will equal $r2$ on average once per generation, i.e., with probability $1/Np$. The probability that all

three indices will be equal is $(1/Np)^2$, but either way, the result is the same: a randomly chosen base vector that has *not* undergone mutation is recombined with the target vector by means of conventional uniform crossover:

$$u_{i,g} = u_{j,ig}\begin{cases} x_{j,r0,g} & if\ (rand_j(0,1) \le Cr \vee j \neq j_{rand}); \\ x_{j,i,g} & otherwise \end{cases} \qquad (2.3)$$

Requiring the base vector to contribute a parameter when $j = j_{\text{rand}}$ ensures that the trial vector will not simply reproduce the vector with which it is compared, i.e., the target vector, $\mathbf{x}_{i,g}$. If, however, Cr is greater than 0, the possibility exists that the trial vector will *duplicate* the base vector. When $Cr = 1$, and $r1 = r2$, duplication is a certainty:

$$r1\_r2(=r0) \wedge Cr = 1: \qquad u_{i,g} = v_{i,g} = x_{r0,g} \qquad (2.4)$$

***r*1 = *r*0 or *r*2 = *r*0: Arithmetic Recombination.** Another special case occurs when either of the difference indices, $r1$ or $r2$, equals the base index, $r0$. When indices are chosen without restrictions, each coincidence occurs on average once per generation. Equation 2.5 elaborates the two possibilities that result when DE's three-vector mutation formula (Eq. 2.1) reduces to a linear relation between the base vector and a single difference vector:

$$\begin{aligned} r1 = r0: &\qquad v_{i,g} = x_{r0,g} + F*(x_{r0,g} - x_{r2,g}), \\ r2 = r0: &\qquad v_{i,g} = x_{r0,g} + F*(x_{r1,g} - x_{r0,g}), \end{aligned} \qquad (2.5)$$

Each two-vector linear combination defines a line that connects the base vector to one of the two difference vectors (Fig. 2.2). *F* plays the role of a coefficient of combination that determines which point along the line is targeted. In the parlance of evolutionary computation, this "line search" is usually called either *continuous* or *arithmetic recombination*.

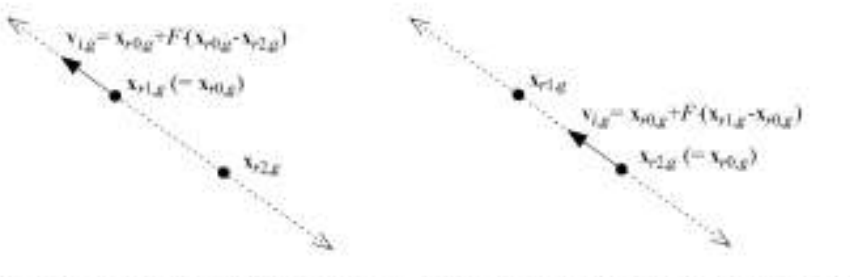


Figure 2.2 Mutation degenerates into two vector arithmetic recombination when either r1=r0 (left) or r2=r0 (right).

Degenerate Combinations Involving the Target Index, i

***r*0 = *i*: Mutation Only.** If the base index, *r*0, is not different from the target index, *i*, then crossover reduce to mutation of the target vector. In this scenario, *Cr* plays the role of a mutation probability:

$$u_{j,i,g} = \begin{cases} x_{j,i,g} + F(x_{j,r1,g} - x_{j,r2,g}) \ if \ \ (rand_j(0,1) \le Cr \vee j = j_{rand}), \\ x_{j,i,g} \quad otherwise \end{cases} \tag{2.6}$$

When base vector indices are randomly selected without restrictions, these degenerate vector combinations occur with probability 1/*Np*.

***i* = *r*1 or *i* = *r*2.** Each of the coincidental events, $i = r1$ and $i = r2$, occurs with probability $1/Np$ when indices are chosen without restrictions. Neither coincidence reduces DE's generating process to a conventional one; mutants are still three-vector combinations and crossover recombines distinct base and target vectors (assuming $r0 \neq i$). Table 2.2 summarizes the possible degenerate vector combinations that can occur.

Table 2.2 First-order degenerate combinations

EVENT	DEGENERATE PROCESS	PROB.	RESULT
r1=r2	Uniform crossover	1/Np	$v_{i,g} = x_{r0,g}$
	Duplication of b.v	Cr^{D-1}/Np	$u_{i,g} = x_{r0,g}$
r0=r1	Intermediate recomb.	1/Np	$v_{i,g} = x_{r0,g} + F(x_{r0,g} - x_{r2,g})$
r0=r2	Intermediate recomb.	1/Np	$v_{i,g} = x_{r0,g} + F(x_{r1,g} - x_{r0,g})$
i=r0	Differential mutation	1/Np	$v_{i,g} = x_{i,g} + F(x_{r1,g} - x_{r2,g})$
i=r1	None	1/Np	$v_{i,g} = x_{r0,g} + F(x_{i,g} - x_{r2,g})$
i=r2	None	1/Np	$v_{i,g} = x_{r0,g} + F(x_{r1,g} - x_{i,g})$

2.3 Differential Mutation

Most dictionaries define mutation as an alteration or change. In the context of genetics and EAs, however, mutation is also seen as change

with a random element. Thus, real-valued EAs typically simulate the effects of mutation with additive increments that are randomly generated by a predefined *probability distribution function*, or PDF. DE, however, uses a uniform PDF not to generate increments, but to randomly sample vector differences:

$$\Delta x_{r1,r2} = (x_{r1} - x_{r2}) \qquad (2.7)$$

In a population of *Np* distinct vectors, there will be *Np**(*Np* - 1) nonzero vector differences and *Np* null differences having zero magnitude giving a total of Np^2 vector differences. Figure 2.3 pictures an arbitrary population of 5 vectors and the sheaf of 20 non-null difference vectors that they generate.

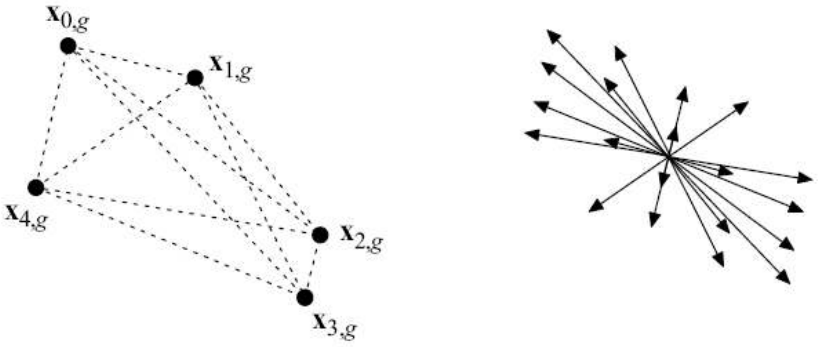


Figure 2.3 The figure on the right displays the sheaf of 20 vector differences generated by the population of 5 vectors shown on the left. Here, differentials have been scaled by half (F=0.5), and transported to a common origin. Note that the distribution is symmetric about zero.

The distribution of difference vectors will depend on the distribution of vectors and this will be different for each objective function. Each distribution, however, will be symmetric about zero because every pair of vectors gives rise to two opposite but equal difference vectors, since reversing the order of the vectors in the differential reverses the sign of the differential:

$$\Delta x_{r1,r2,g} = (x_{r1,g} - x_{r2,g}) = -(x_{r2,g} - x_{r1,g}) = -\Delta x_{r2,r1,g} \qquad (2.8)$$

2.3.1 The Mutation Scale Factor: *F*

Limits on F

Upper: The stated range for F is (0.1), although 1.0 is an empirically derived upper limit in the sense that no function that has been successfully optimized has required $F > 1$. This is not to say that solutions are not possible when $F > 1$, but only that they tend to be both more time consuming and less reliable than if $F < 1$. When $F = 1$ exactly, otherwise distinct vector combinations become indistinguishable:

$$x_{r0,g} + x_{r1,g} - x_{r2,g} = \begin{cases} x_{r0,g} + F(x_{r1,g} - x_{r2,g}) \\ x_{r1,g} + F(x_{r0,g} - x_{r2,g}) \end{cases} \qquad when\ F = 1 \qquad (2.9)$$

This discontinuity at $F = 1$ reduces the number of mutants by half and can result in erratic convergence unless $Cr < 1$, since $Cr = 1$ further

restricts the pool of possible trial vectors by not crossing mutant and target parameters.

Lower: In general, selection tends to reduce the diversity of a population, whereas mutation increases it. To avoid premature convergence, it is crucial that F be of sufficient magnitude to counteract this selection pressure. Zaharie (2002) recently demonstrated the existence of what is effectively a lower limit for F, finding that if F is too small, the population can converge even if selection pressure is absent.

2.4 Recombination

Recombination randomly exchanges or merges parameters from two or more vectors to create one or more trial vectors. *Discrete recombination*, also known as *crossover*, is an operation in which trial vector parameters are copied from randomly selected vectors. Since it only copies information, crossover can be applied to binary, real-valued or even symbolic data.

By contrast, *continuous* or *arithmetic recombination* expresses trial vectors as linear combinations of vectors, so it is inapplicable to symbolic data and inappropriate for binary variables. Both crossover and arithmetic recombination have a variety of implementations. Those with particular relevance to DE are described below.

2.5 Crossover

It was originally thought that crossover could exponentially increase the probability of above-average parameter groupings (alleles) while exponentially decreasing the likelihood of less than average groupings. More recent analysis shows that growth is not exponential because the selective advantage of a parameter grouping decreases as it becomes more prevalent. Empirical evidence also exists suggesting that (uniform) crossover does not decrease the time complexity of an EA but merely speeds convergence by a constant factor. Nevertheless, crossover plays a significant role in most EAs.

Global discrete recombination refers to the case where both vectors are chosen anew for each trial *parameter*. The ES globally recombines its strategy variables, but like DE and most GAs, it crosses objective function parameters from just two vectors (*dual crossover*). Both DE and ES also use crossover to create a single trial vector, whereas most GAs cross two vectors to produce two trial vectors, often by one-point crossover.

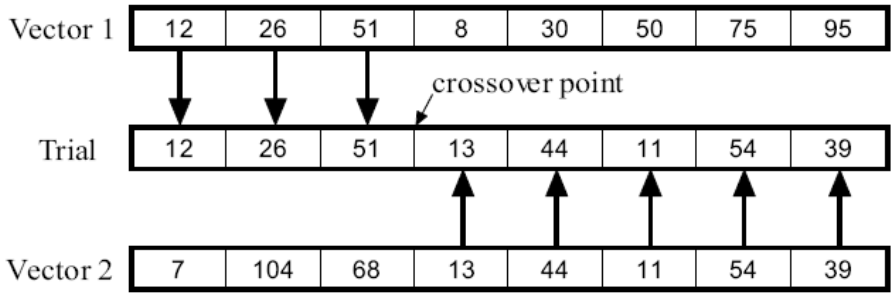


Figure 2.4 One-point crossovers. Each string represents vector parameters. In this figure, D=8 and values are integral, although real-valued or symbolic data could also have been used. Each vector contributes a contiguous series of parameters values to the trial vector. The crossover point is randomly chosen. In this case, it occurs when between the third and fourth parameters.

2.5.1 One-Point Crossover

There are several ways to assign donors to trial parameters. For example, *one-point crossover* randomly selects a single *crossover point* such that all parameters to the left of the crossover point are inherited from vector 1, while those to the right are copied from the vector 2 (Fig. 2.4).

2.5.2 N-Point Crossover

N-point crossover randomly subdivides the trial vector into $n + 1$ partitions such that parameters in adjacent partitions are inherited from different vectors. If n is odd (e.g., one-point crossover), parameters near opposite ends of a trial vector are less likely to be taken from the same

vector than when n is even (e.g., $n = 2$). This dependence on parameter separation is known as *representational* or *positional bias*, since the particular way in which parameters are ordered within a vector affects algorithm performance. Studies of n-point crossover have shown that recombination with an even number of crossover points reduces the representational bias at the expense of increasing the disruption of parameters that are closely grouped. To reduce the effect of their individual biases, DE's exponential crossover employs both one- and two-point crossover.

2.5.3 Exponential Crossover

DE's exponential crossover achieves a similar result to that of one- and two-point crossover, albeit by a different mechanism. One parameter is initially chosen at random and copied from the mutant to the corresponding trial parameter so that the trial vector will be different from the vector with which it will be compared (i.e., the target vector, $\mathbf{x}_{i,g}$). The source of subsequent trial parameters is determined by comparing Cr to a uniformly distributed random number between 1 and 0 that is generated anew for each parameter, i.e., rand$_j$(0,1). As long as rand$_j$(0,1) $\leq Cr$, parameters continue to be taken from the mutant vector, but the *first time* that rand$_j$(0,1) $> Cr$, the current *and all remaining* parameters are taken from the target vector. The example in Fig. 2.5 illustrates a case in which the exponential crossover model produced two crossover points.

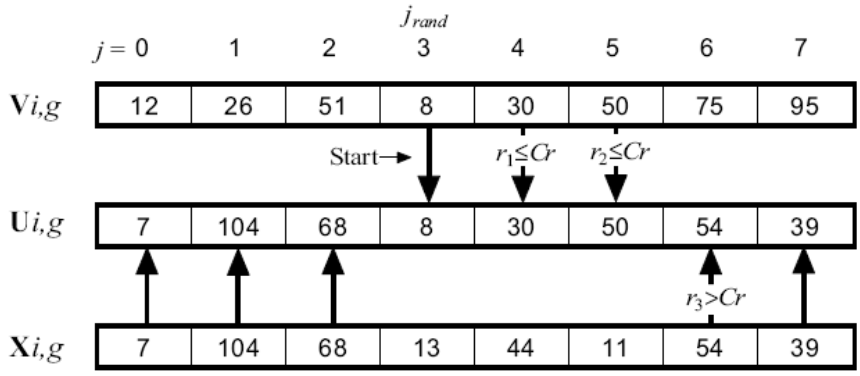


Figure 2.5 Exponential Crossovers. Starting at the randomly chosen parameter index, j_{Rand} (=3), trial parameters are inherited from the mutant, $v_{i,g}$, as long as $rand_j$ (0,1) <Cr (e.g., j=4, 5). The first time that $rand_j$ > Cr, all remaining trial parameters (e.g., j=6, 7, 0, 1, 2) are inherited from the target vector, $x_{i,\,g,}$ Indices are computed modulo D=8.

2.5.4 Uniform (Binomial) Crossover

Uniform crossover is a process in which independent random trials determine the source for each trial parameter. Crossover is uniform in the sense that each parameter, regardless of its location in the trial vector, has the same probability, p_{Cr}, of inheriting its value from a given vector. For this reason, uniform crossover does not exhibit a representational bias.

When the vectors being crossed are randomly chosen from the same population, p_{Cr} and 1 - *pCr* create the same pool of trial vectors. For example, both p_{Cr} = 0.3 and p_{Cr} = 0.7 produce a vector that on average inherits 30% of its parameters from one vector and 70% from another. In particular, when two vectors, A and B, are crossed with p_{Cr} = 0.3, trial

vectors will inherit, on average, 30% of their parameters from A and 70% from B. It is equally probable, however, that B will be drawn first and A second, in which case trial vectors inherit, on average, 30% of their parameters from B and 70% from A. These trial vectors could also have been generated by taking A first, B second and $p_{Cr} = 0.7$. Reversing the roles of the donor vectors has the same effect as using 1- p_{Cr} instead of p_{Cr}. Since the order in which vectors are chosen is random, p_{Cr} potentially generates the same population as does 1- p_{Cr}.

DE on the other hand crosses vectors from different populations and their order of crossover is not random. In DE, each value of $Cr \sim p_{Cr}$ generates a different trial population.

As with exponential crossover, DE's version of uniform crossover begins by taking a randomly chosen parameter from the mutant so that the trial vector will not simply replicate the target vector. Comparing *Cr* to rand*j* (0,1) determines the source for each remaining trial parameter. If rand*j* (0,1) $\leq$ *Cr*, then the parameter comes from the mutant; otherwise, the target is the source. Figure 2.6 illustrates the process.

2.6 The Role of *Cr* in Optimization

Despite mediating a crossover process, *Cr* can also be thought of as a *mutation rate*, i.e., the (approximate) probability that a parameter will be inherited from a mutant. In DE, the average number of parameters mutated for a given *Cr* depends on the crossover model (e.g., exponential

or binomial) but in each, a low *Cr* corresponds to a low mutation rate. Many Ga's recommend a mutation rate of 1/*D*, meaning that, on average; only one trial parameter is mutated. Indeed, Zaharie's results for Rastrigin, Griewangk and the sphere, as well as those for the simple hyper-ellipsoid, consistently found low *Cr* to be the most effective values. Similarly, optimizing the extensive test beds in Storn and Price (1997) showed that all functions could be solved with either

$0 \leq Cr \leq 0.2$ or $0.9 \leq Cr \leq 1$. The reason for the bifurcation of the crossover control space was not at first appreciated until it was realized that functions solvable with low *Cr* were inevitably decomposable, while those requiring $Cr \sim 1$ were not.

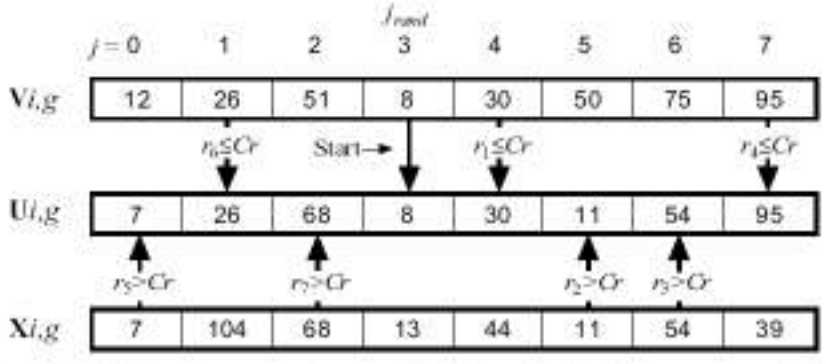


Figure 2.6 Uniform Crossovers. One an initial, randomly chosen parameter is inherited from the mutant (e.g., j_{rand} =3), D-1 independent trials are conducted to determine the source of remaining parameters. If randj (0, 1) < Cr, the mutant donates a parameter value; otherwise, parameters are copied from the target.

2.7 Selection

There are primarily two stages in the evolutionary process where selection can be applied to a population. Typically, vectors with the best function values are assigned the highest *selection probability*, making them the most likely to be chosen for mating. This strategy mimics the one employed by breeders and botanists who try to improve traits by selectively breeding individuals with superior characteristics.

In practice, methods for assigning selection probabilities involve additional assumptions about how to map objective function values to a set of probabilities. Instead of selecting mates based on objective function value, both ES and classic DE select mates with equal probability. In the ES, each vector has the same chance to be chosen for mutation and/or recombination. Similarly, classic DE randomly selects base vectors without regard for their objective function values.

In contrast to parent selection, s*urvivor selection*, also called *replacement*, chooses the next generation of vectors from the current generation of vectors and trial vectors. Most EAs apply selection pressure either when choosing vectors to recombine or when choosing survivors. GA's typically bias selections in favor of better vectors, whereas DE, ES and other EA's, however, combine randomly chosen vectors and apply selection pressure only when picking survivors. Using both parent (base vector) and survivor selection can cause premature convergence to a local optimum.

3.0 HANDLING OF INTEGER AND BINARY VARIABLES

3.1. Integer variables

Original DE algorithm is only capable of handling continuous variables. Extending it to optimize integer variables, however, is very easy and requires only couple of simple modifications (Corne et al., 1999). First, integer values should be used to evaluate the objective function, even though DE itself still works internally with continuous floating-point values. Therefore

$$y_i = \begin{cases} x_i & for\ continuous\ \text{var}iables, \\ INT(x_i) & for\ \text{int}eger\ \text{var}iables. \end{cases}$$

INT() is a function for converting a real value to an integer value by truncation. Additionally, truncation is preformed here for evaluating trial vectors and for handling boundary constraints. Truncated values are not assigned elsewhere. Hence, DE works with a population of continuous variables regardless of the corresponding object variable type which is also essential for maintaining diversity of the population and the robustness of the algorithm.

3.2 Binary or discrete variables

In the past, integer variables are handled in the same way as described above however, a new procedure is proposed and evaluated to handle binary (discrete) variables.

This procedure is called Approach 1 and is compared with another procedure (Li, 1992), called Approach 2. These are described below:

Approach 1:

In this procedure, for the function evaluation, binary variables are handled as follows:

$$y_i = \begin{cases} 0 & if\ x_i \le 0.5, \\ 1 & otherwise \end{cases}$$

where x_i is a continuous variable $0 \le xi \le 1$. However boundary constraint is handled in the same way as for continuous variables. The only difference is that lower and upper bound are set to zero and one respectively.

Approach 2:

It is a nonlinear transformation for modeling binary variables as continuous variables proposed by Li (1992). A binary variable $y \in \{0, 1\}$ can be modeled as a continuous variable $x \in [0, 1]$, simply by the addition of the following constraint in the problem:

$$x(1-x) = 0 \qquad 0 \le x \le 1$$

which forces x to take either 0 or 1. Hence with this transformation any MINLP model can be converted into an equivalent NLP model. The

function x (1 − x) is a non-convex nonlinear function. Li (1992) referred to this as the “binary condition” to model binary variables and found this procedure to be more convenient than current approaches as branch and bound method and implicit enumeration method. The resulting NLP was solved using a modified penalty function method, however only local optimal solutions were found due to the addition of non-convexities. Also, the use of standard NLP solver like SQP (sequential quadratic programming) and GRG (generalized reduced gradient) is ruled out as they could be sensitive to initial guess and hence get stuck at local optima. Ryoo and Sahinidis (1995) proposed a specialized branch and reduce algorithm for the global optimization of NLPs and MINLPs wherein they refer to the usage of such procedure for handling binary and discrete variables.

4.0 EFFECT OF KEY PARAMETERS (CR, F, AND NP)

The performance of DE and MDE algorithms depends on key parameters, namely, CR, *F*, and NP. By choosing the key parameters (NP, CR, and *F*) wisely/appropriately, the problem of premature convergence can be avoided to a large extent. To study the effect of key parameters, two highly multimodal multidimensional test functions are used. Also, a basic key parameter set (CR = 0.85, *F* = 0.9, and NP = 30) is first chosen. Then, all the numerical simulations are carried out around this basic set taking maximum number of generation to be 250. A systematic numerical simulation is performed keeping two key parameters constant while varying the third at a time. The two test functions are defined below:

Ackley's function: This is a continuous, highly nonlinear function that causes the search with moderate complications.

$$f_1 = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^{\,2}}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \qquad (4.1)$$

$$-20 \le x_i \le 30 \qquad n = 30$$

The global minimum is: $f_1 = 0$ with $x_i = 0$, $i = 1, 2, \ldots, n$.

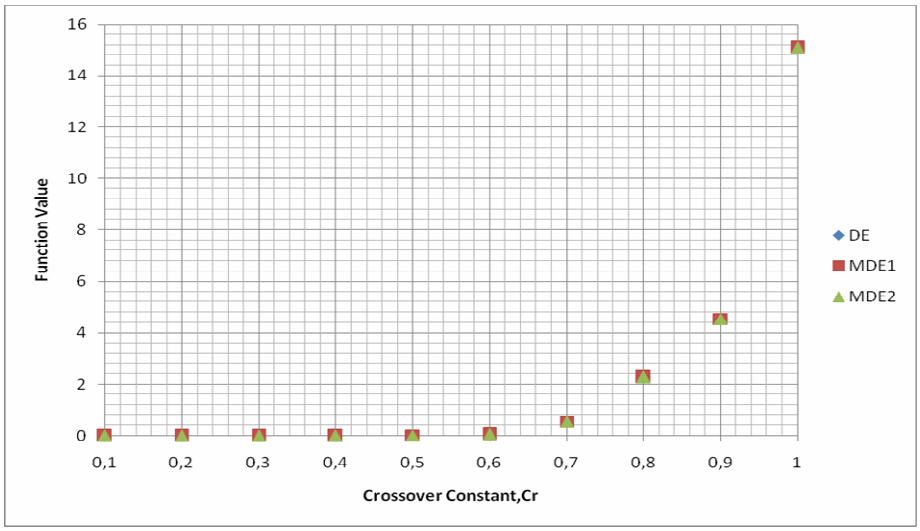


Figure 4.1 Effects of crossover constant (Cr)

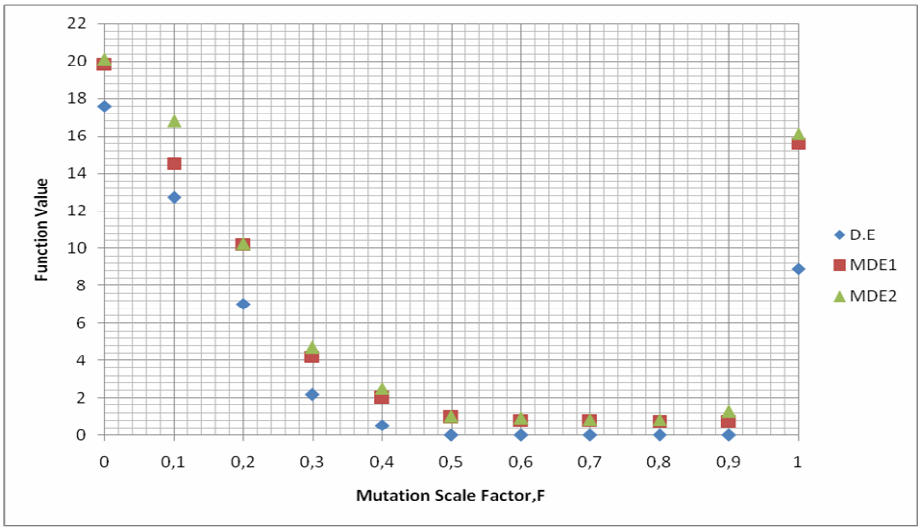


Figure 4.2 Effects of scaling factor (F)

Rastrigin's function: This function is also considered relatively difficult to minimize because the number of locally optimum points is high.

$$f_2 = 2n + \sum_{i=1}^{n} \left(x_i^{\,2} - 2\cos(2\pi x_i) \right) \qquad (4.2)$$

$$-5.12 \le x_i \le 5.12 \qquad n = 20$$

The global minimum is: $f_2 = 0$ with $x_i = 0$, $i = 1, 2, \ldots, n$.

Figure 4.3 Effects of crossover constant (Cr)

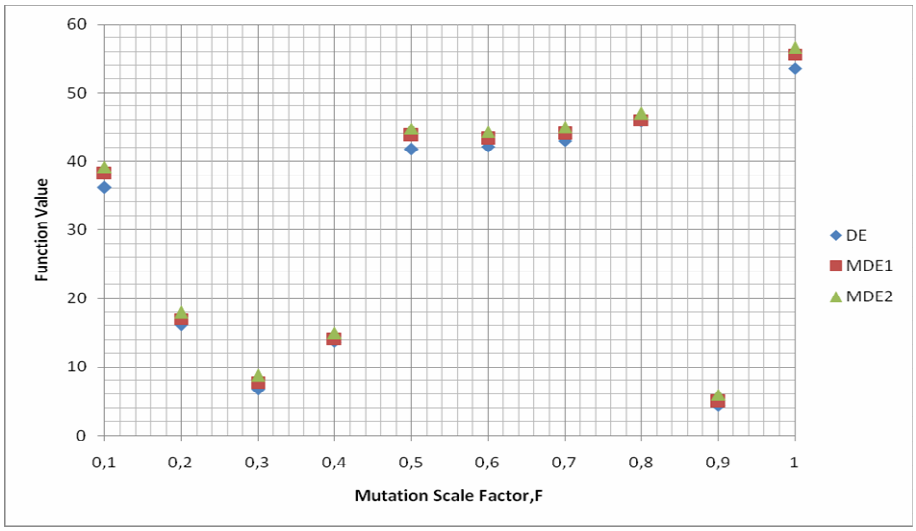


Figure 4.4 Effects of scaling factor (F)

Figs. 4.1–4.4 show the effect of CR, *F*, and NP, respectively, on the performance of MDE and DE using the two functions as mentioned above. Each point on the figures represents the average of 10 independent experiments.

It is evident from these figures that both DE and MDE are affected in a similar way, i.e., the variation of function value with CR/*F*/NP is same qualitatively. It is important to note that variation of function value with CR, *F*, and NP is different for the same problem but it is same for DE and MDE. Also, the variation of function value with CR and *F* is problem dependent, i.e., different problems may have different trends of function value vs. CR/*F*/NP. Therefore, for comparison purposes, it is essential to keep the same setting of key parameters for both DE and MDE although this setting can be different for different problems.

5.0 MODIFIED DIFFERENTIAL EVOLUTION

5.1 Strategies of Search

Antecedent Strategies

There are five DE strategies (or schemes) that were proposed by K. Price and R. Storn [Sto96a]:

• Scheme DE/rand/1 $\omega = x1 + F \cdot (x2 - x3)$

• Scheme DE/rand/2 $\omega = x5 + F \cdot (x1 + x2 - x3 - x4)$

Later, two more strategies were introduced by Fan and Lampinen:

• Trigonometric scheme [FL01]

$\omega = (x1 + x2 + x3)/3 + (p2 - p1) \cdot (x1 - x2) + (p3 - p2) \cdot (x2 - x3) + (p1 - p3) \cdot (x3 - x1)$

$pi = |f(xi)/ (f(x1) + f(x2) + f(x3)) |,\ i = 1, 2, 3$;

• Directed scheme [FL03]

$\omega = x3 + (1 - f(x3))/f(x1) \cdot (x3 - x1) + (1 - f(x3))/f(x2) \cdot (x3 - x2)$,

where $f(x3) \leq f(x1)$, $f(x3) \leq f(x2)$.

Four Groups of Strategies

What is the common point for all the methods of continuous optimization? Perhaps that we take some initial point, base point, and from this point we search some *direction* in which we suppose attaining the optimum as soon as possible. For that I searched a certain operation in the form of $\omega = \beta + F \cdot \delta$, where β, *base* vector, and δ, *difference* vector, are calculated with consideration of the actual population state (see Figure. 5.1).

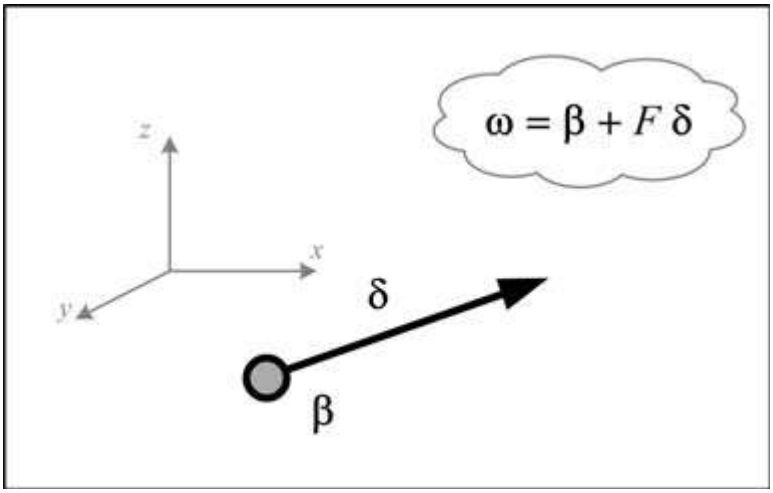


Figure. 5.1 Fundamental principle inherent to all methods of continuous optimization applied to differential evolution: β – base point, δ – optimal direction.

But how we can create this β and δ? The most practical solution is to use the barycenters of the individuals randomly chosen from the population. I accepted two variants to create β: random and local. In the latter case I

gave preference to the best individual of the population. As for δ, either one constructs two barycenters in a random manner, or constructs them taking into consideration the values of an objective function, so the difference vector is oriented or directed. I would like to note that the directed case interprets well the simulation of the gradient. All these combinations of β and δ choice lead us to the four groups of strategies: random, directed, local, and hybrid (see Figure. 5.2).

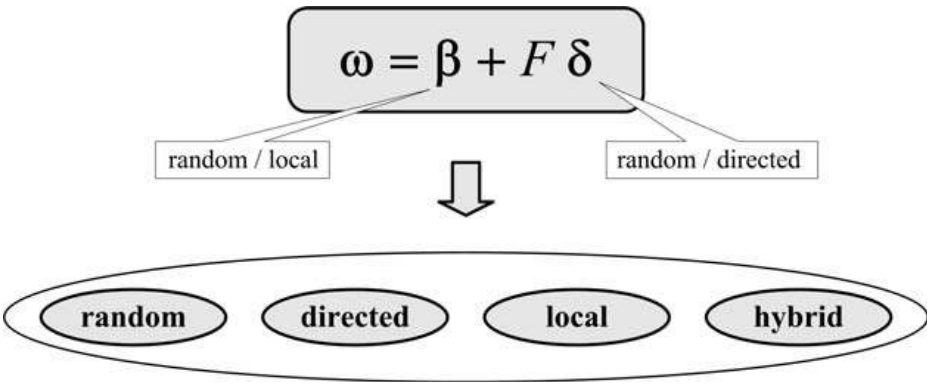


Figure. 5.2 Four groups of strategies.

So, by using the knowledge about values of the objective function I have divided all the possible strategies into four groups:

RAND group

Contains the strategies in which the trial individual is generated without any information about values of the objective function.

RAND/DIR group

Rand/Dir group consists of the strategies that use values of the objective function to determine a "good" direction. This group represents an imitation of the gradient function.

RAND/BEST group

Rand/best group uses the best individual to form the trial one. It looks like a chaotically local search around the current best solution.

RAND/BEST/DIR group

Rand/best/dir combines the last two groups into one including all their advantages.

5.1.1 RAND Group

Randomly extracted individuals X_i are arbitrarily separated into two classes $C^{'}$ and $C^{''}$ containing $n^{'}$ and $n^{''}$ elements accordingly. Then the barycenters of these classes $V_{C'}$ and $V_{C''}$ are found by the formula

$$V_c = \frac{1}{n}\sum_{i=1}^{n} X_i \qquad n = n^{'}, n^{''}$$

There are two possibilities for choosing the base vector $\beta = Vg$:
1. Using some individual from these classes Vg ∈ C'U C'';

2. Using another individual from the population Vg $\notin$ C' U C'';
Thus, the differentiation formula for this group of strategies is (Figure. 5.3)

$$w = V_g + F(V_{c''} - V_{c'}).$$

5.1.2 RAND/DIR Group

Let randomly extracted individuals X_i be divided into two classes C_+ and C_- with $n+$ and $n-$ elements so, that for each element from the class C_+ its objective function value would be less than the objective function value of any element from class C_-. That is,

$$f(X_i) \le f(X_j) \qquad (\forall X_i \in C_+) \wedge (\forall X_j \in C_-)$$

$$i = 1, ..., n_+ \qquad j = 1, ..., n_-$$

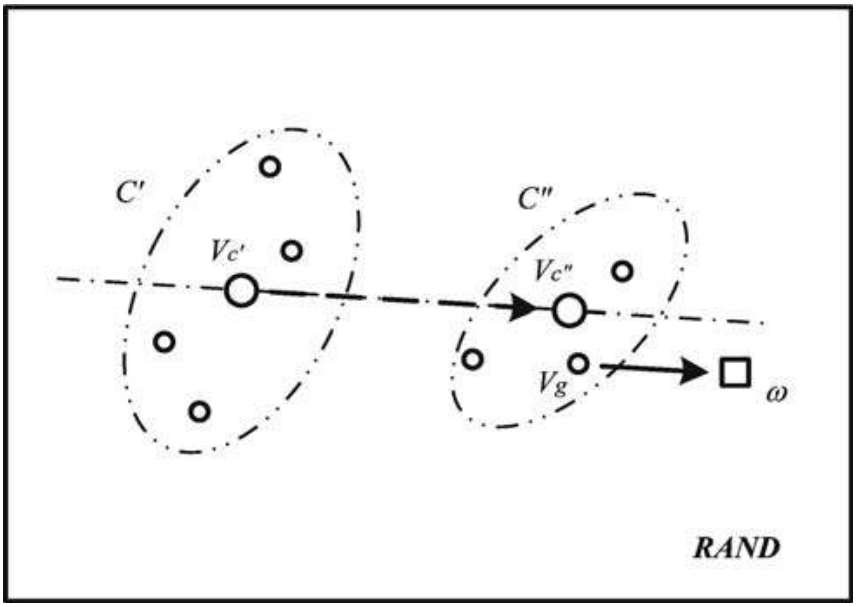


Figure 5.3 ***RAND*** group of strategies.

We find then the maximal and minimal elements of each of the classes

$$f(X_{C_+}{}^{Min}) \le f(X_i) \le f(X_{C_+}{}^{Max}) \qquad \forall X_i \in C_+$$

$$f(X_{C_-}{}^{Min}) \le f(X_i) \le f(X_{C_-}{}^{Max}) \qquad \forall X_i \in C_-$$

Then we calculate the positive and negative shifts inside the classes.

$$V_S^{\pm} = \lambda\left(X_{C_\pm}{}^{Min} - X_{C_\pm}{}^{Max}\right) \qquad \lambda = 1/2 \quad \inf luence\, cons\tan t$$

So the average shift is equal to

$$V_s = (V_s^+ + V_s^-)/2$$

Hence, the differentiation formula is

$$w = V_{C_+} + F(V_{C+} - V_{C-} + V_S)$$

where V_{C+} and V_{C-} are barycenters of C_+ and C_- accordingly (see Figure. 5.4).

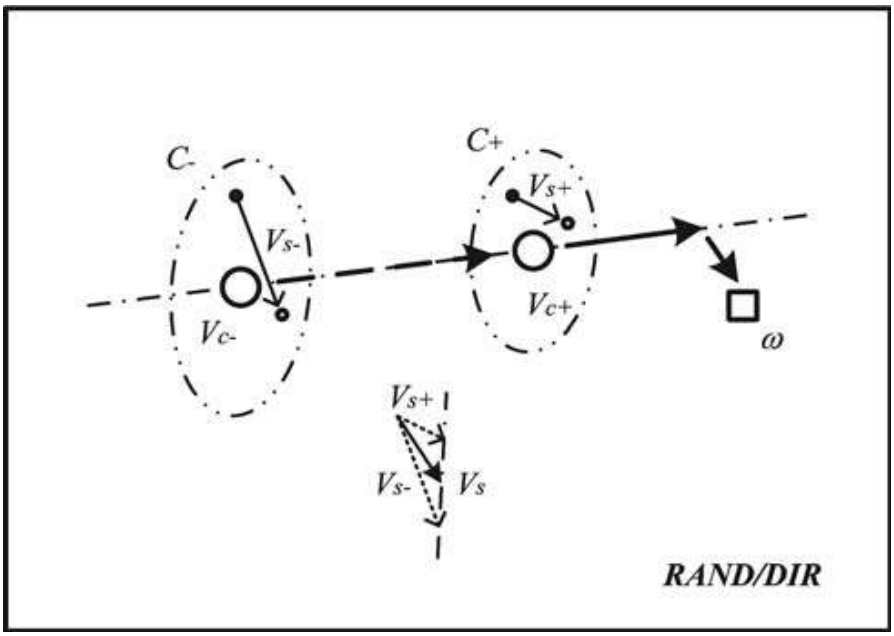


Figure. 5.4 ***RAND/DIR*** group of strategies.

6.0 EXAMPLES OF STRATEGIES

In this section detailed examples of strategies that form each of the groups are considered and setting in the particular Differentiation formulae for these strategies and deduce the interrelation between the general differentiation constant F and its particular representation $\wp$ according to the strategy.

At first, fix the notation:

- *ind* — target (current) individual.
- *{xi}* — set of extracted individuals.
- V{xi} is an individual that has the minimal value of the objective function among extracted individuals and the target one.
- $V^{'}$ and $V^{''}$ are other individuals; if there are more than three extracted individuals we denote them V_1, V_2, V_3, ...
- V_b — the best individual.
- δ — difference vector.
- $\wp$ — particular constant of differentiation.
- β — base vector, the point of the difference vector application.

So, the formula of differentiation always has the following form;

$$w = \beta + \wp\,\delta$$

6.1 RAND Strategies

6.1.1 Rand1 Strategy

In this strategy only one random individual x_1 is extracted from the population. At the same time this individual presents the base vector β of the strategy. The difference vector δ is formed by the current and extracted individuals ($x_1 - ind$). The step length is equal to $\wp$ ($x_1 - ind$). The formula of differentiation for this strategy is

$$w = x_1 + \wp(x_1 - ind)$$

Comparing with the main group's formula we can see that $Vg = V_C'' = x_1$ and $V_C' = ind$. Therefore, the constant of differentiation $F = \wp$. Such an approach gives $(NP - 1)$ possibilities for the trial individual, where NP is the size of population. This strategy is shown in Figure. 6.1.

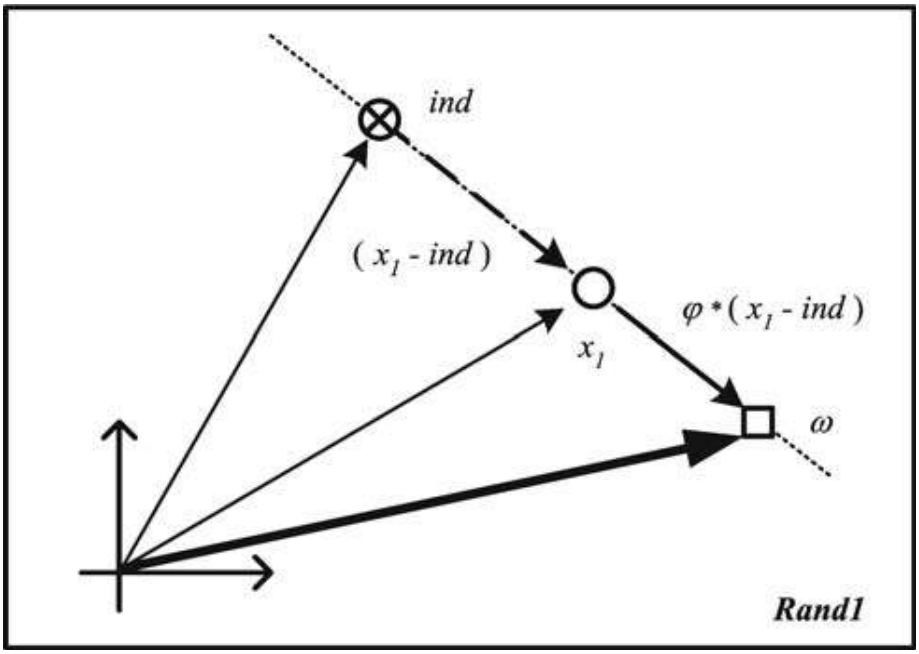


Figure. 6.1. *Rand1* strategy.

6.1.2 Rand2 Strategy

The target individual *ind* interchanges with a second randomly extracted one x_2. Thus, the number of possible trial individuals increases to (NP −1)(NP − 2). As we can see, by extracting an additional individual the exploration capabilities augment considerably. The difference is $\delta = x1 - x2$, and the base is $\beta = x1$, where $x1$ and $x2$ represent the barycenters $V_{C'}$ and $V_{C''}$ accordingly. $Vg = x_1$ and $F = \wp$. The differentiation formula of this strategy is

$$w = x_1 + \wp(x_1 - x_2)$$

The strategy is shown in Figure. 6.2.

6.1.3 Rand3 Strategy

Here, three individuals are randomly extracted to form this strategy. So, all possible combinations are augmented to $(NP-1)(NP-2)(NP-3)$. The first two extracted vectors generate the difference $\delta = x1 - x2$, but in this case the base vector is the third extracted vector $x3$. The formula of differentiation is

$$w = x_3 + \wp(x_1 - x_2)$$

The strategy is shown in Figure. 6.3.

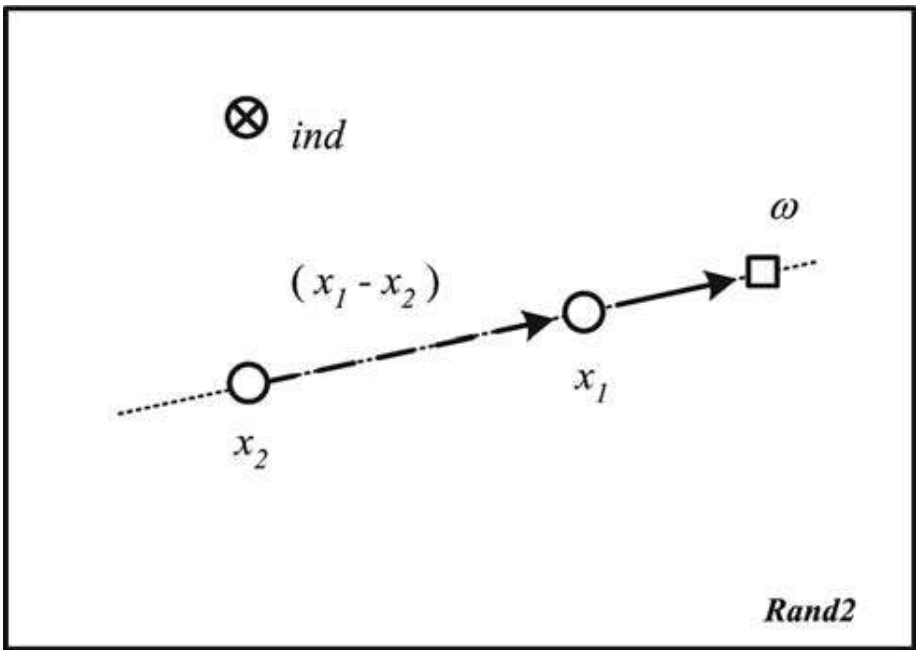


Figure. 6.2 *Rand2* strategy.

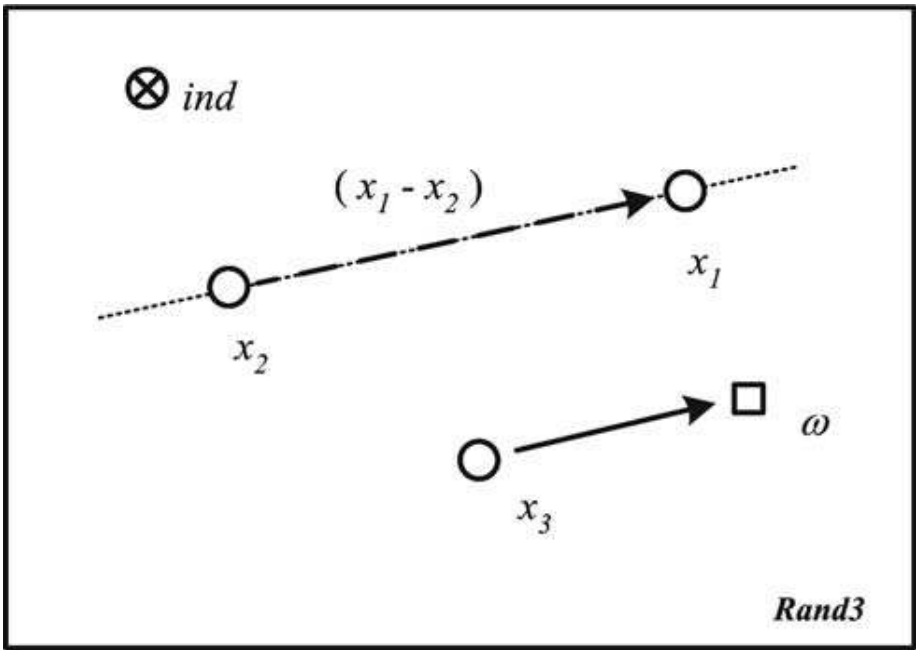


Figure. 6.3 *Rand3* strategy.

6.1.4 Rand4 Strategy

By extracting four random individuals the next strategy can be written as

$$w = x_2 + \wp(x_2 - x_1 + x_4 - x_3)$$

where the difference $\delta = x2 - x1 + x4 - x3$ and the base $\beta = x2$. In this case, the difference is generated by superposition of two random directions $(x2-x1)$ and $(x4 - x3)$. Any of these individuals can be chosen as the base vector: for example, $\beta = x2$ as in (3.15). The vectors $x1$ and $x3$ create the barycenter $Vc' = (x1+x3)/2$ and the same with the vectors $x2$ and $x4$: $Vc'' = (x2+x4)/2$. $Vg = x2$. So, (3.15) may be rewritten as $\omega = x2$

$+\wp\ (x2 - x1 + x4 - x3) = x2+\wp((x2+x4)/2-(x1+x3)/2) = Vg+2\wp$ (Vc''−Vc'). It is clear that $F = 2\wp \equiv \wp = F/2$. We can see that in order to harmonize the strategy with its template it is necessary to divide *F* by 2. The number of possible combinations of four random individuals is $(NP - 1)(NP - 2)(NP - 3)(NP - 4)$. This strategy is shown in Figure. 6.4.

6.1.5 Rand5 Strategy

In this strategy five random individuals are extracted. The first four individuals work as in the *Rand4* strategy, and the fifth individual realizes the base point $\beta = Vg = x5$. Notice that $F = \varphi/2$ too. The search space exploration becomes appreciable; there are

$$(NP -1)(NP -2)(NP -3)(NP -4)(NP -5) = \prod_{i=1}^{5} (NP - i)$$

potential combinations of extracted vectors. The strategy is shown in Figure. 6.5.

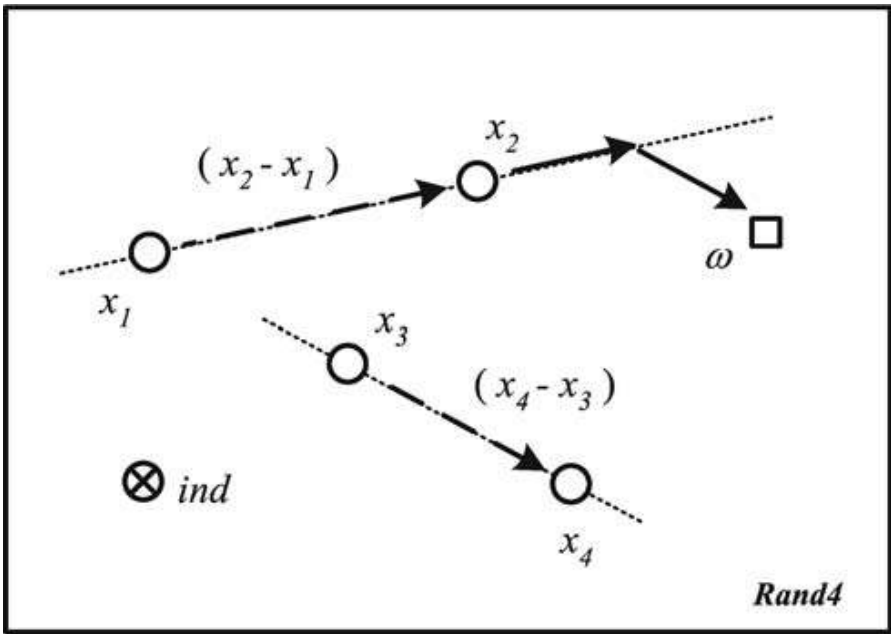


Figure. 6.4 *Rand4* strategy.

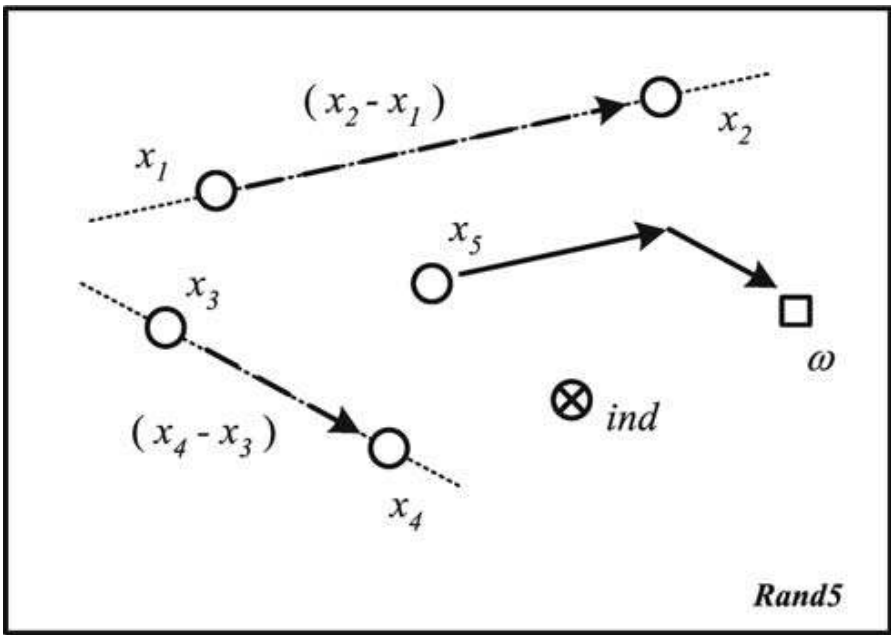


Figure. 6.5 *Rand5* strategy.

6.2 RAND/DIR Strategies

The main principle remains the same as in the RAND group of strategies. Moreover, the information about the objective function is used to calculate the direction of differentiation. In this way, the probability of the optimal choice increases twice. Such an approach is analogous to descent in a direction opposite to the gradient vector.

6.2.1 Rand1/Dir1 Strategy

This strategy uses one extracted individual $x1$ and target individual *ind* for the differentiation formula. With introduced notations it looks like

$$w = V^* + \wp(V^* - V')$$

or, equivalently

$$w = \begin{cases} x_1 + \wp(x_1 - ind) & if\ f(x_1) < f(ind); \\ ind + \wp(ind - x_1) & otherwise. \end{cases}$$

This formula is similar to its generalization with Vc+ =V * and Vc- = V '. Hence $F = \wp$ and Vs = 0. The strategy is shown in Figure. 6.6.

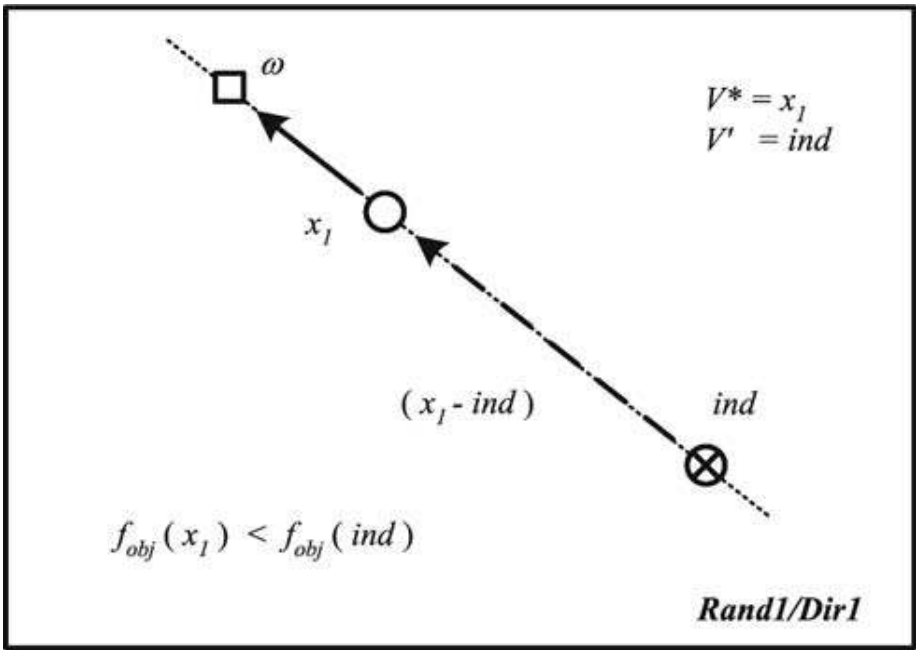


Figure. 6.6 *Rand1/Dir1* strategy.

6.2.2 Rand2/Dir1 Strategy

Following such a tendency illustrating the strategy in which two random individuals $x1$ and $x2$ are extracted. The formula of this strategy is the same but the current individual *ind* is not used. It may be represented also as

$$w = \begin{cases} x_1 + \wp(x_1 - x_2) & \text{if } f(x_1) < f(x_2); \\ x_2 + \wp(x_2 - x_1) & \text{otherwise.} \end{cases}$$

Also, $F = \delta$ and Vs = 0. The strategy is shown in Figure. 6.7.

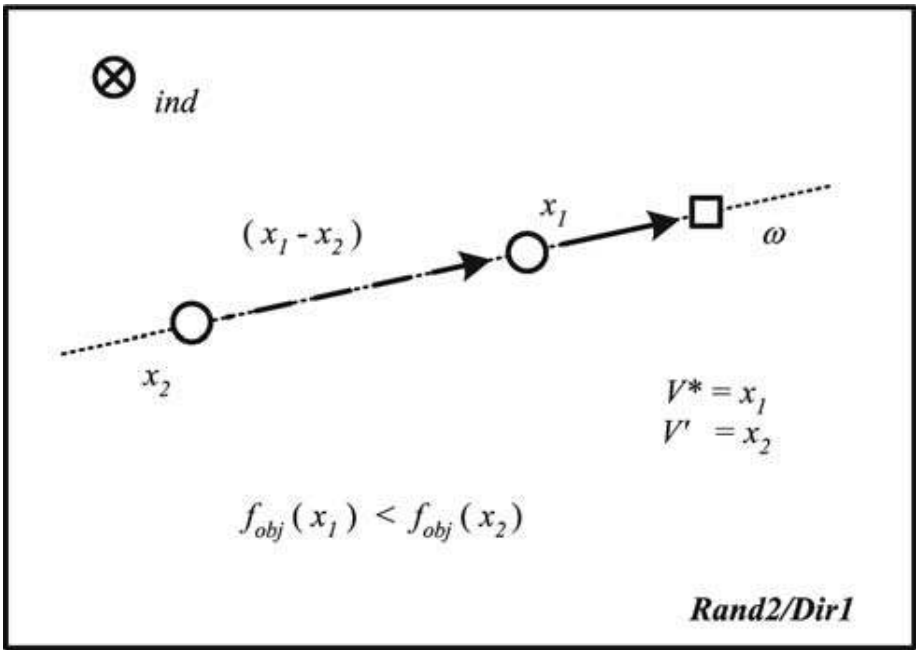


Figure. 6.7 *Rand2/Dir1* strategy.

6.2.3 Rand3/Dir2 Strategy

The next strategy uses three random elements *x*1, *x*2, and *x*3. These elements are sorted according to their values of the objective function so that $f(V^*) \leq f(V'')$ and $f(V^*) \leq f(V')$, where V *,V ‘,V’’ = {x1, x2, x3}. The formula of this strategy is:

$$w = V^* + \wp(2V^* - V' - V'')$$

Note that the superposition of two random directions (V_+-V_-) and (V^* - V'') (difference vector) is used here. The base point β is the individual with the minimal value of the objective function V^*. To adjust this

differentiation's formula with its template (3.7) imagine that the individuals form two barycenters $Vc_- = (V_+ - V_-)/2$ and $Vc_+ = V^*$.
Thus, $\omega = V^* + 2\wp\,(V^* - (V' - V'')/2) = Vc_+ + 2\delta\cdot(Vc_+ - Vc_-)$. There is no average shift vector in this case ($Vs = 0$). So, $\wp = F/2$. The strategy is shown in Figure. 6.8.

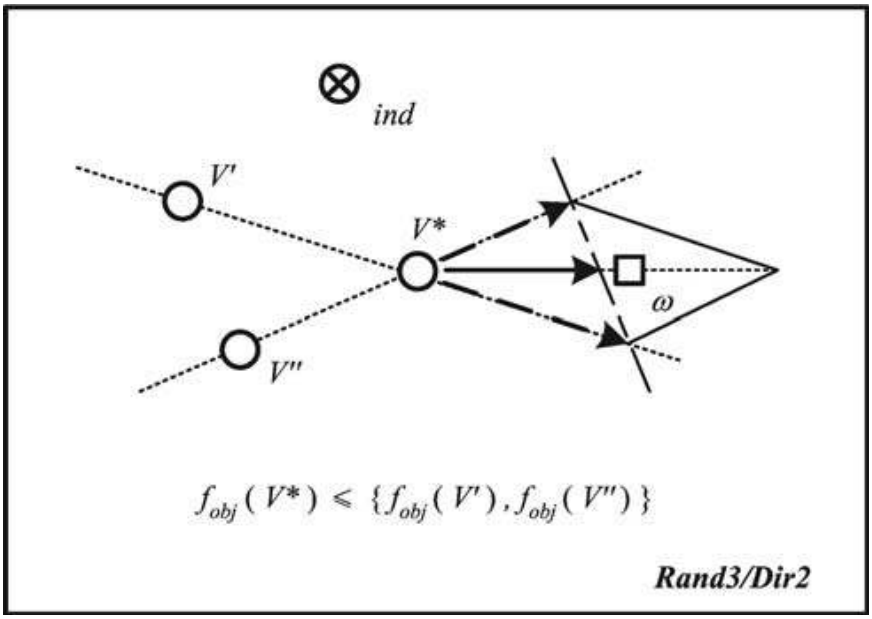


Figure.6.8 *Rand3/Dir2* strategy.

6.2.4 Rand4/Dir2 Strategy

This strategy is based on the choice of four random individuals $x1$, $x2$, $x3$, and $x4$. Let us denote these individuals as $V1$, $V2$, $V3$, and $V4$ so that $f(V1) \leq f(V2)$ and $f(V3) \leq f(V4)$. Hence the differentiation formula is

$$w = V_1 + \wp(V_1 - V_2 + V_3 - V_4)$$

It is obvious that $VC+ = (V1 + V3)/2$ and $VC- = (V2 + V4)/2$. The only distinction is that the base vector $\beta = V1$, but not $VC+$ as in the template (3.7). Such a small difference allows us to simplify the strategy without losing quality. Moreover it is easy to verify that $\wp = F/2$ and $V_S = 0$. This strategy is shown in Figure. 6.9.

6.2.5 Rand4/Dir3 Strategy

This strategy continues to evolve the ideas of the *Rand3/Dir2* one. Here, it is applied in three directions constructed on four randomly extracted individuals. V *,the individual with the minimal value of the objective function.

$$w = V^* + \wp(3V^* - V_1 - V_2 - V_3)$$

It is obvious that $\wp = F/3$ and $VS = 0$. This strategy is shown in Figure. 6.10.

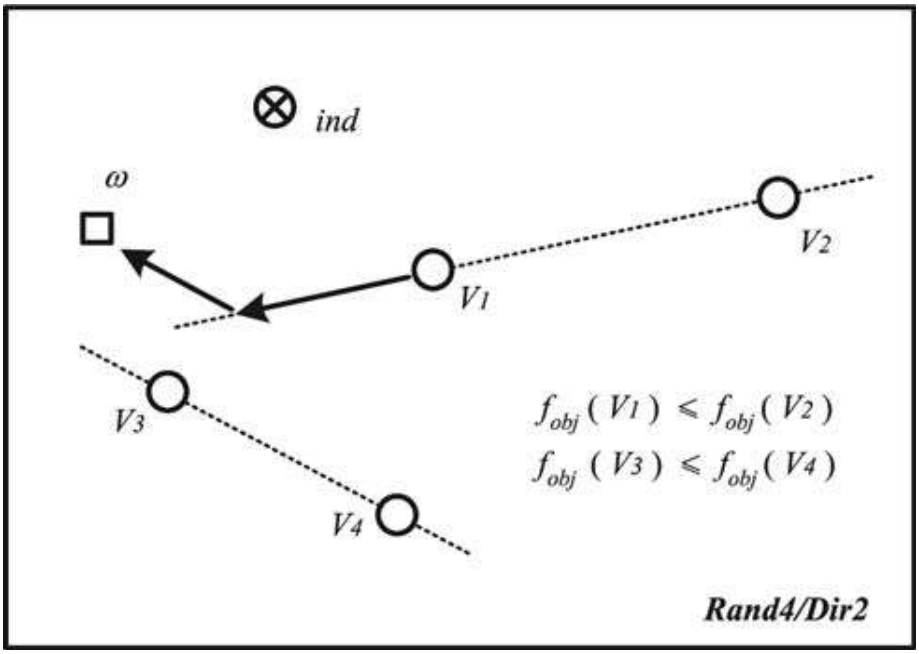


Figure. 6.9 *Rand4/Dir2* strategy.

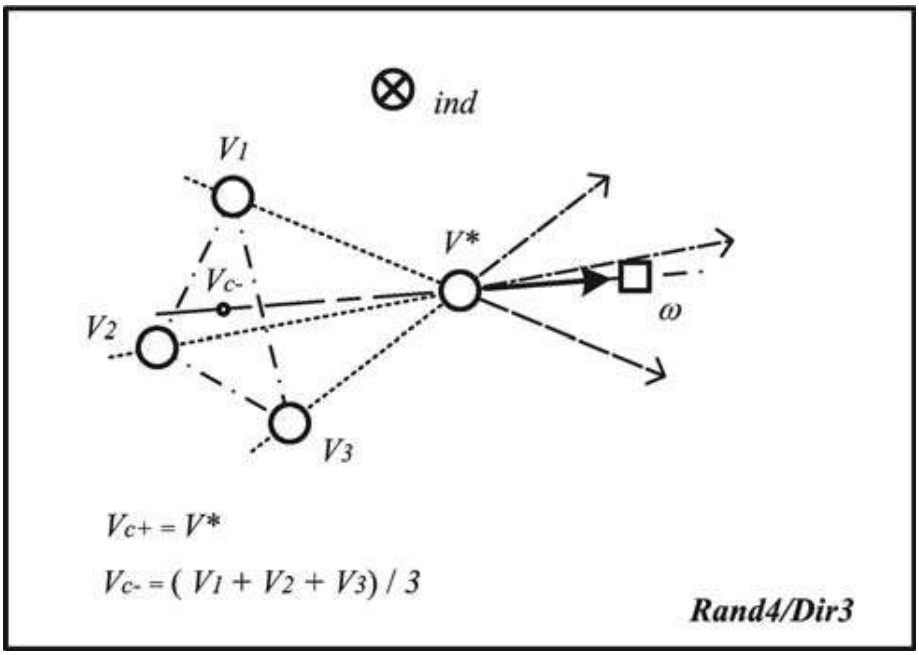


Figure. 6.10 *Rand4/Dir3* strategy.

6.2.6 Rand5/Dir4 Strategy

Continuing to follow the tendency, we illustrate the strategy built on five random individuals, which form four random directions. The purpose of increasing the number of extracted individuals is to determine more precisely the descent direction by better exploiting the information about the objective function and, at the same time, to explore the search space more and more entirely. The differentiation formula that represents this case is

$$w = V^* + \wp(3V^* - V_1 - V_2 - V_3 - V_4)$$

Here, $\wp = F/4$, $V_S = 0$. The strategy is shown in Figure. 6.11.

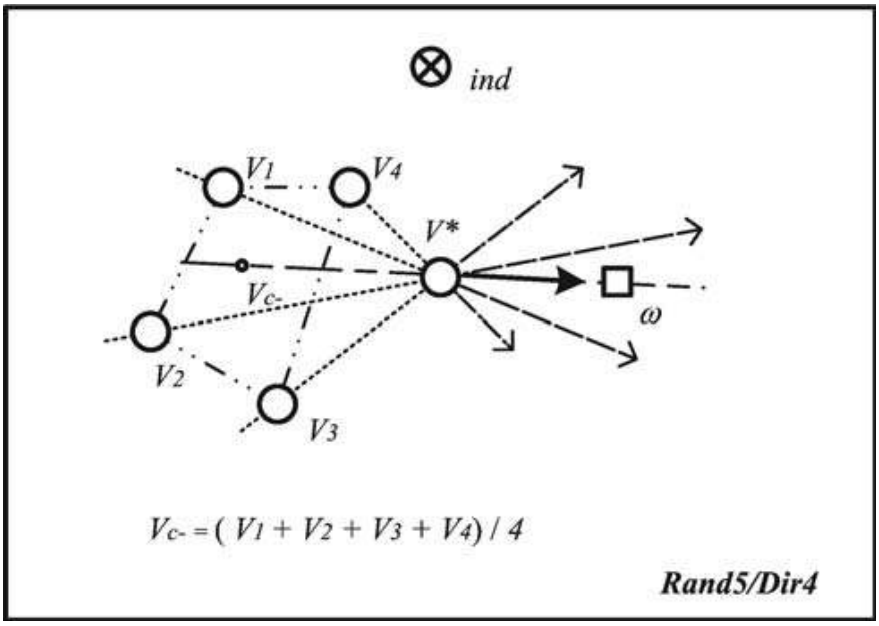


Figure. 6.11 *Rand5/Dir4* strategy.

7.0 PENALTY FUNCTIONS

7.1 Introduction to Constraints

Most optimization problems have constraints. The solution or set of solutions which are obtained as the final result of an evolutionary search must necessarily be feasible, that is, satisfy all constraints. Taxonomy of constraints can be considered and composed of (a) number, (b) metric, (c) criticality and (d) difficulty. A first aspect is number of constraints, ranging upwards from one. Sometimes problems with multiple objectives are reformulated with some of the objectives acting as constraints. Difficulty in satisfying constraints will increase (generally more than linearly) with the number of constraints. A second aspect of constraints is their metric, either continuous or discrete, so that a violation of the constraint can be assessed in distance from satisfaction using that metric. A third consideration is the criticality of the constraint, in terms of absolute satisfaction. A constraint is generally formulated as hard (absolute) when in fact, it is often somewhat soft. That is, small violations would be considered for the final solution if the solution is otherwise superior to other solutions. Evolutionary algorithms are especially capable of handling soft constraints since a population of solutions is returned at each point in the search. This allows the user to select a solution which violates a soft constraint (technically infeasible) over a solution which would be the best, technically feasible solution found. A final aspect of constraints to be considered is the difficulty of satisfying the constraints. This difficulty can be characterized by the size

of the feasible region (*F*) compared to the size of the sample space (*S*). The difficulty may not be known *a priori*, but can be gauged in two ways. The first way is how simple it is to change a solution which violates the constraint to a solution which does not violate the constraint. The second way is the probability of violating the constraint during search. For example, a constraint may be frequently violated but the solution can be easily made feasible. Conversely, a constraint violation may be very difficult to resolve, but occur rarely in the search.

7.2 Methods of Handling Constraints

As originally conceived, evolutionary methods produce new solutions by recombining and / or perturbing existing solutions. In certain instances, an encoding, a reproduction (e.g. crossover) operator and a perturbation (e.g. mutation) operator can be devised such that feasible existing solutions (parents) will always produce feasible new solutions (children). An example using binary integers illustrates this: exchanging bits between two *k*-bit integers will always produce a *k*-bit integer. If a one-to-one correspondence between *k*-bit integers and feasible solutions is established, every newly-produced encoding will correspond to a feasible solution to the original problem. This is an example of a constraint which is easily handled through encoding and operators, so all solutions examined during evolution are feasible, as was demonstrated effectively in Esbensen (1995).

In other cases, it is not clear how an encoding and operators can be defined to preserve feasibility. One approach is to increase the complexity of the evolution operators by adding repair operators, so that they are guaranteed to produce feasible encodings. For problems such as TSP, compensatory conflict-resolution operators (repair operators) could be implemented, so that the resulting child encoding is itself a valid permutation. Several researchers have taken this approach to problems (Liepins et al. 1990, Liepins and Potter 1991, Orvosh and Davis 1994, Tate and Smith 1995A), and it works well without restrictive computational cost. Interestingly, the first three papers repaired the infeasible solution to calculate the fitness, but left the solution unrepaired in the population (with some probability). When an infeasible solution can be readily modified to yield a feasible solution which maintains most or all of the structure of the parent solutions, repair mechanisms can be efficient and effective. However, some repair mechanisms can introduce systematic bias into the search.

Unfortunately, many optimization problems involve constraints for which it is not easy to repair an infeasible solution in order to make it feasible. The repair operators may be prohibitively computationally expensive or they may severely disturb the superior aspects of the parent solutions carried in the children, defeating the fundamental strength of evolution. A method to consider in this case is to allow only feasible solutions to be maintained during evolution. Solutions which violate constraints are immediately discarded upon creation, and not considered further. If constraint violations are encountered relatively infrequently

during evolution (that is, a fairly loosely constrained problem), then discarding infeasible solutions can be an efficient method.

However, in many cases the problem of finding any feasible solution is itself NP-hard (Garey and Johnson 1979). A number of methods have been proposed in the evolutionary computation literature for such highly constrained problems. DeJong and Spears (1989) suggest polynomially reducing other NP-Complete problems to the Boolean Satisfiability problem, for which effective (and compact) GA implementations are known. This method has the drawback that the polynomial reduction involved may be extremely complex, and may greatly increase the size of the problem. Michalewicz (1992) and Michalewicz and Janikow (1991) suggest eliminating the equalities in constraints and formulating special genetic operators which guarantee feasibility. This approach works efficiently for linear constraints and continuous variable domains. A more generic approach borrowed from the mathematical programming literature is that of exterior penalty methods.

7.3 Introduction to Penalty Functions

Penalty functions have been a part of the literature on constrained optimization for decades. Three degrees of penalty functions exist: barrier methods in which no infeasible solution is considered, partial penalty functions in which a penalty is applied near the feasibility boundary, and global penalty functions which are applied throughout the infeasible region (Schwefel 1995). In the area of combinatorial

optimization, the popular Lagrangian relaxation method (Avriel 1976, Fisher 1981, Reeves 1993) is a variation on the same theme: temporarily relax the problem's most difficult constraints, using a modified objective function to avoid straying too far from the feasible region. In general, a penalty function approach is as follows. Given an optimization problem,

$$\begin{aligned} &\min f(x) \\ &s.t.\ \ x \in A \qquad\qquad (P) \\ &\quad\ x \in B \end{aligned}$$

where **x** is a vector of decision variables, the constraints "$\mathbf{x} \in$ A" are relatively easy to satisfy, and the constraints "$\mathbf{x} \in$ B" are relatively difficult to satisfy, the problem can be reformulated as

$$\begin{aligned} &\min \quad f(x) + p(d(x, B)) \\ &s.t. \quad\ x \in A \qquad\qquad (R) \end{aligned}$$

where d(**x**, B) is a metric function describing the distance of the solution vector **x** from the region B, and p(.) is a monotonically non-decreasing penalty function such that $p(0) = 0$. If the exterior penalty function, p(.), grows quickly enough outside of B, the optimal solution of (P) will also be optimal for (R). Furthermore, any optimal solution of (R) will provide an upper bound on the optimum for (P), and this bound will in general be tighter than that obtained by simply optimizing f(**x**) over A.

In practice, the constraints "$\mathbf{x} \in$ B" is expressed as inequality and equality constraints in the form of

$$\begin{aligned} &g_i(x) \le 0 for\ i = 1, ..., q \\ &h_i(x) = 0 \ \ for\ \ i = q+1, ..., m \end{aligned}$$

where q = number of inequality constraints
m - q = number of equality constraints

Various families of functions $p(.)$ and $d(.)$ have been studied for evolutionary optimization to dualize constraints. Different possible distance metrics, $d(.)$, include a count of the number of violated constraints, the Euclidean distance between **x** and B as suggested by Richardson et al. (1989), a linear sum of the individual constraint violations or a sum of the individual constraint violations raised to an exponent, κ. Variations of these approaches have been attempted with different levels of success. Some of the more notable examples are described in the following sections.

It can be difficult to find a penalty function which is an effective and efficient surrogate for the missing constraints. The effort required to tune the penalty function to a given problem instance or repeatedly calculate it during search may negate any gains in eventual solution quality. As noted by Siedlecki and Sklansky (1989), much of the difficulty arises because the optimal solution will frequently lie on the boundary of the feasible region. Many of the solutions most similar to the genotype of the optimum solution will be infeasible. Therefore, restricting the search to only feasible solutions or imposing very severe penalties makes it difficult to find the schemata that will drive the population toward the optimum as shown in the research of (Smith and Tate 1993, Anderson and Ferris 1994, Coit et al. 1995, Michalewicz 1995). Conversely, if the penalty is not severe enough, then too large a region is searched and

much of the search time will be used to explore regions far from the feasible region. Then, the search will tend to stall outside the feasible region. A good comparison of six penalty function strategies applied to continuous optimization problems is given in Michalewicz (1995). These strategies include both static and dynamic approaches, as discussed below, as well as some less generic approaches such as sequential constraint handling (Schoenauer and Xanthakis 1993) and forcing all infeasible solutions to be dominated by all feasible solutions in a given generation (Powell and Skolnick 1993).

7.4 Static Penalty Functions

A simple method to penalize infeasible solutions is to apply a constant penalty to those solutions which violate feasibility in any way. The penalized objective function would then be the unpenalized objective function plus a penalty (for a minimization problem). A variation on this simple penalty function is to add a metric based on number of constraints violated, where there are multiple constraints. The penalty function for a problem with *m* constraints would then be as below (for a minimization problem):

$$f_p(x) = f(x) + \sum_{i=1}^{m} C_i \delta_i \qquad (1)$$

$$where \begin{cases} \delta_i = 1 & if\ constraint\ i\ is\ violated \\ \delta_i = 0 & if\ constraint\ i\ is\ satisfied \end{cases}$$

$fp(\mathbf{x})$ is the penalized objective function, $f(\mathbf{x})$ is the unpenalized objective function, and Ci is a constant imposed for violation of constraint i. This penalty function is based only on the number of constraints violated, and is generally inferior to the second approach based on some distance metric from the feasible region (Goldberg 1989, Richardson et al. 1989).

More common and more effective is to penalize according of distance to feasibility, or the "cost to completion" as termed by Richardson et al. (1989). This was done crudely in the constant penalty functions of the preceding paragraph by assuming distance can be stated solely by number of constraints violated. A more sophisticated and more effective penalty includes a distance metric for each constraint, and adds a penalty which becomes more severe with distance from feasibility. Complicating this approach is the assumption that the distance metric chosen appropriately provides information concerning the nearness of the solution to feasibility, and the further implicit assumption that this nearness to feasibility is relevant in the same magnitude to the fitness of the solution. Distance metrics can be continuous (see for example, Juliff 1993) or discrete (see for example, Patton et al. 1995), and could be linear or non-linear (see for example, Le Riche et al. 1995).

A general formulation is as follows for a minimization problem:

$$f_p(x) = f(x) + \sum_{i=1}^{m} C_i d_i^{\kappa}$$

$$where \;\; d_i \begin{cases} \delta_i g_i(x), & for \; i = 1,.....q \\ |h_i(x)|, & for \; i = q+1,.....m \end{cases} \qquad (2)$$

di is the distance metric of constraint *i* applied to solution **x** and □is user defined exponent, with values of □of 1 or 2 often used. Selection of *Ci* is more difficult. The advice from Richardson et al. (1989) is to base *Ci* on the expected or maximum cost to repair the solution (i.e. alter the solution so it is feasible). For most problems, however, it is not possible to determine *Ci* using this rationale. Instead, it must be estimated based on the relative scaling of the distance metrics of multiple constraints, the difficulty of satisfying a constraint, and the seriousness of a constraint violation, or be determined experimentally.

Many researchers in evolutionary computation have explored variations of distance based static penalty functions (e.g., Baeck and Khuri 1994, Goldberg 1989, Huang et al. 1994, Olsen 1994, Richardson et al. 1989). One example (Thangiah 1995) uses a linear combination of three constant distance based penalties for the three constraints of the vehicle routing with time windows problem. Another novel example is from (Le Riche et al. 1995) where two separate distance based penalty functions are used for each constraint in two GA segregated subpopulations. This "double penalty" somewhat improved robustness to penalty function parameters since the feasible optimum is approached with both a severe and a lenient penalty.

Homaifar et al. (1994) developed a unique static penalty function with multiple violation levels established for each constraint. Each interval is defined by the relative degree of constraint violation. For each interval *l*, a unique constant, *Cil* is then used as a penalty function coefficient.

7.5 Dynamic Penalty Functions

The primary deficiency with static penalty functions is the inability of the user to determine criteria for the C_i coefficients. Also, there are conflicting objectives involved with allowing exploration of the infeasible region, yet still requiring that the final solution be feasible. A variation of distance based penalty functions, which alleviates much of these difficulties, is to incorporate a dynamic aspect which (generally) increases the severity of the penalty for a given distance as the search progresses. This has the property of allowing highly infeasible solutions early in the search, while continually increasing the penalty imposed to eventually move the final solution to the feasible region. A general form of a distance based penalty method incorporating a dynamic aspect based on length of search, t, is as follows for a minimization problem:

$$f_p(x,t) = f(x) + \sum_{i=1}^{m} s_i(t) d_i^{\kappa} \qquad (3)$$

where $s_i(t)$ is a monotonically non-decreasing in value with t. Metrics for t include number of generations or the number of solutions searched. Recent uses of this approach include Joines and Houck (1994) for continuous function optimization and Olsen (1994), who compares several penalty functions, all of which consider distance, but some also consider evolution time. A common concern of these dynamic penalty formulations is that they result in feasible solutions at the end of evolution. If $s_i(t)$ is too lenient, final infeasible solutions may result, and if $s_i(t)$ is too severe, the search may converge to non-optimal feasible solutions. Therefore, these penalty functions typically require problem

specific tuning to perform well. One explicit example of $s_i(t)$ is as follows, from Joines and Houck (1994),

$s_i(t) = (C_i t)^{\alpha}$

where α is constant equal to 1 or 2, as defined by Joines and Houck.

7.6 Adaptive Penalty Functions

While incorporating distance together with the length of the search into the penalty function has been generally effective, these penalties ignore any other aspects of the search. In this respect, they are not adaptive to the ongoing success (or lack thereof) of the search and cannot guide the search to particularly attractive regions or away from unattractive regions based on what has already been observed. A few authors have proposed making use of such search specific information. Siedlecki and Sklansky (1989) discuss the possibility of self-adapting penalty functions, but their method is restricted to binary-string encodings with a single constraint, and involves considerable computational overhead.

Bean and Hadj-Alouane (1992) and Hadj-Alouane and Bean (1992) propose penalty functions which are revised based on the feasibility or infeasibility of the best, penalized solution during recent generations. Their penalty function allows either an increase or a decrease of the imposed penalty during evolution as shown below, and was demonstrated on multiple choice integer programming problems with one constraint. This involves the selection of two constants, β_1 and β_2 ($\beta_1 > \beta_2 > 1$), to adaptively update the penalty function multiplier, and the evaluation of

the feasibility of the best solution over successive intervals of *Nf* generations. As the search progresses, every *Nf* generations the penalty function multiplier is updated based on whether the best solution was feasible during that interval. Specifically, the penalty function is as follows,

$$f_p(x,k) = f(x) + \sum_{i=1}^{m} \lambda_k d_i^{\kappa} \qquad (4)$$

$$\lambda_{k+1} = \begin{cases} \lambda_k \beta_1, & \text{if previous } N_f \text{ generations have infeasible best solution} \\ \lambda_k / \beta_1, & \text{if previous } N_f \text{ generations have feasible best solution} \\ \lambda_k, & \text{otherwise} \end{cases}$$

Smith and Tate (1993) and Tate and Smith (1995B) used both search length and constraint severity feedback in their penalty function which was enhanced by the work of Coit et al. (1995). This penalty function involves the estimation of a near-feasible threshold (*NFT*) for each constraint. Conceptually, the *NFT* is the threshold distance from the feasible region at which the user would consider the search as "getting warm." The penalty function encourages the evolutionary algorithm to explore within the feasible region and the *NFT*-neighborhood of the feasible region, and discourage search beyond that threshold. This formulation is below:

$$f_p(x,t) = f(x) + \left(F_{feas}(t) - F_{all}(t) \right) \sum_{i=1}^{m} \left(\frac{d_i}{NFT_i} \right)^{\kappa} \qquad (5)$$

where $F_{all}(t)$ denotes the unpenalized value of the best solution yet found, and $F_{feas}(t)$ denotes the value of the best feasible solution yet found. The $F_{all}(t)$ and $F_{feas}(t)$ terms serve several purposes.

First, they provide adaptive scaling of the penalty based on the results of the search. Second, they combine with the NFT_i term to provide a search specific and constraint specific penalty.
The general form of *NFT* is:

$$NFT = \frac{NFT_0}{1 + \Lambda} \qquad (6)$$

where NFT_o is an upper bound for *NFT*. Λ is a dynamic search parameter used to adjust *NFT* based on the search history. In the simplest case, Λ can be set to zero and a static *NFT* results. Λ can also be defined as a function of the search, for example, a function of the generation number (t), i.e., $\Lambda = f(t) = \lambda t$. A positive value of λ results in a monotonically decreasing *NFT* (and thus, a larger penalty) and a larger λ more quickly decreases *NFT* as the search progresses, incorporating both adaptive and dynamic elements.

If *NFT* is ill defined *a priori*, it can be set at a large value initially with a positive constant l used to iteratively guide the search to the feasible region. With problem specific information, more efficient search can take place by defining a tighter region or even static values of *NFT*.

7.7 Future Directions in Penalty Functions

Two areas requiring further research are the development of completely adaptive penalty functions which require no user specified constants and the development of improved adaptive operators to exploit characteristics of the search as they are found. The notion of adaptiveness is to leverage the information gained during evolution to improve both the effectiveness and the efficiency of the penalty function used. Another area of interest is to explore the assumption that multiple constraints can be linearly combined to yield an appropriate penalty function. This implicit assumption of all penalty function used in the literature assumes that constraint violations incur independent penalties and therefore, there is no interaction between constraints.

Intuitively, this seems to be a possibly erroneous assumption, and one could make a case for a penalty which increases more than linearly with the number of constraints violated.

8.0 APPLICATIONS

8.1 Unconstraint Benchmark Test Functions

8.1.1 Minimize ES2;

$$f = -\cos(x_1)\cos(x_2)\exp\left[-((x_1-\pi)^2+(x_2-\pi)^2)\right]$$

Subject to;

$$-100 \le x_1, x_2 \le 100$$

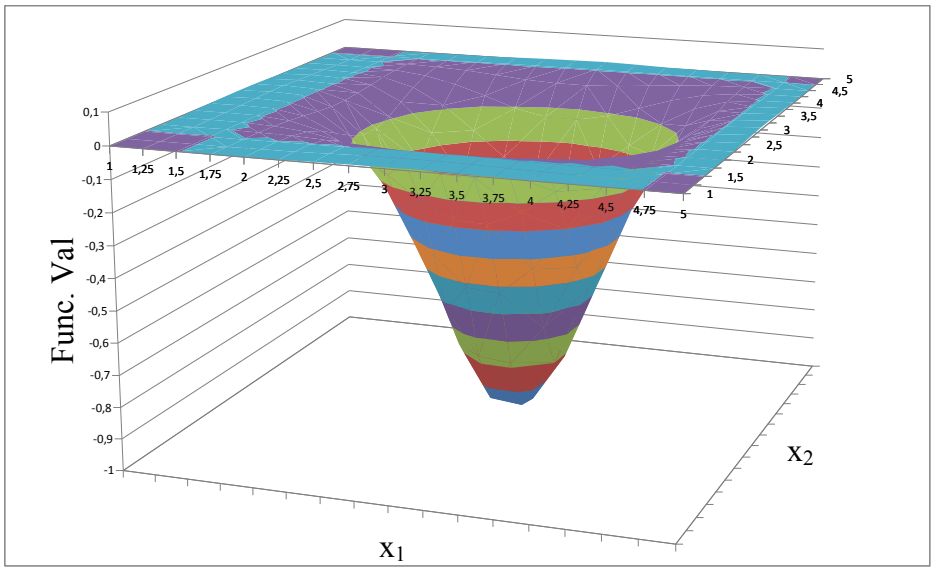


Figure 8.1 ES2 Function

8.1.2 Hyper-Ellipsoid

$$f(x) = \sum_{j=0}^{D-1} 2^j x_j^2$$

$$-100 \le x_j \le 100 \quad j = 0,1,\ldots\ldots,D-1$$

$$f(x^*) = 0, \qquad x^*{}_j = 0, \quad \varepsilon = 1.0x10^{-6}$$

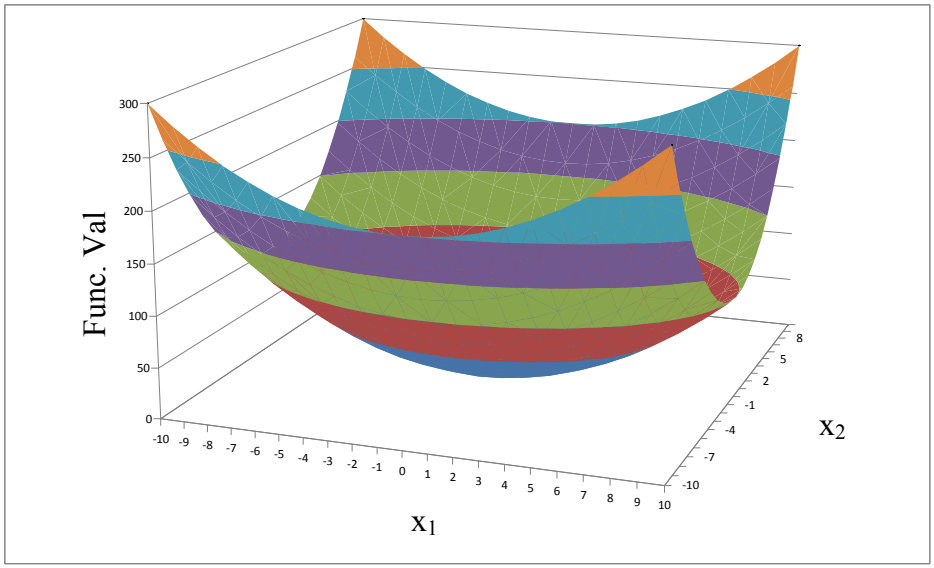


Figure 8.2 Hyper-Ellipsoid

8.1.3 Schwefel's Ridge

$$f(x) = \sum_{k=0}^{D-1}\left(\sum_{j=0}^{k} x_j\right)^2$$

$$-100 \le x_j \le 100 \qquad x_j^* = 0 \qquad \varepsilon = 1.0x10^{-6}$$

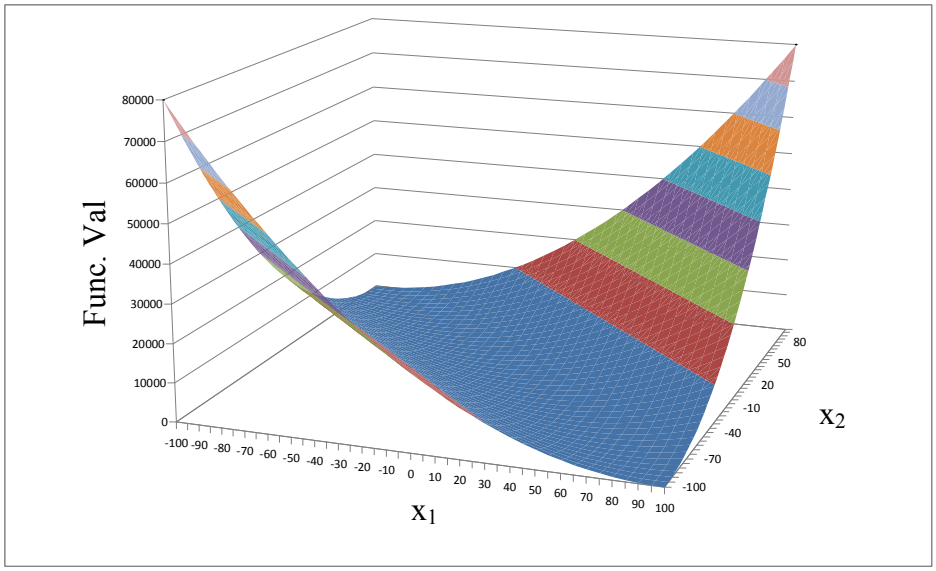


Figure 8.3 Schwefel's Ridge

8.1.4 Neumaier #3

$$f(x) = \sum_{j=0}^{D-1} (x_j - 1)^2 - \sum_{j=1}^{D-1} x_j x_{j-1} \qquad D > 1$$

$$-D^2 \le x_j \le D^2, \quad j = 0,1,\ldots\ldots,D-1$$

$$f(x^*) = -D(D+4)(D-1)/6 \qquad x^*_j = (j+1)(D-j)$$

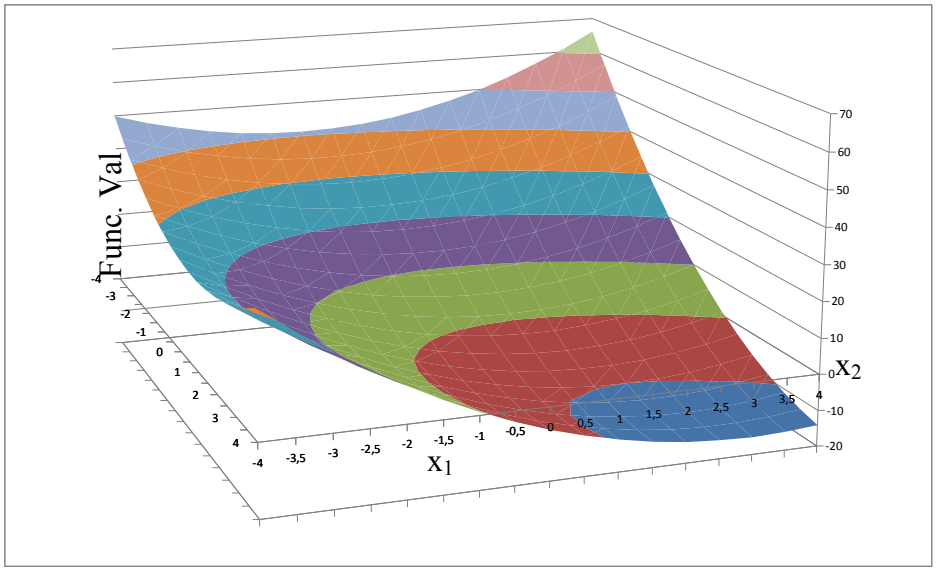


Figure 8.4 Neumaier #3

8.1.5 Salomon

$$f(x) = -\cos\left(2\pi\|x\|\right) + 0.1\|x\| + 1$$

$$\|x\| = \sqrt{\sum_{j=0}^{D-1} x_j^{\,2}}$$

$$-100 \le x_j \le 100 \qquad j = 0,1,\ldots\ldots,D-1$$

$$f(x^*) = 0, \qquad x^*_{\,j} = 0, \quad \varepsilon = 1.0x10^{-6}$$

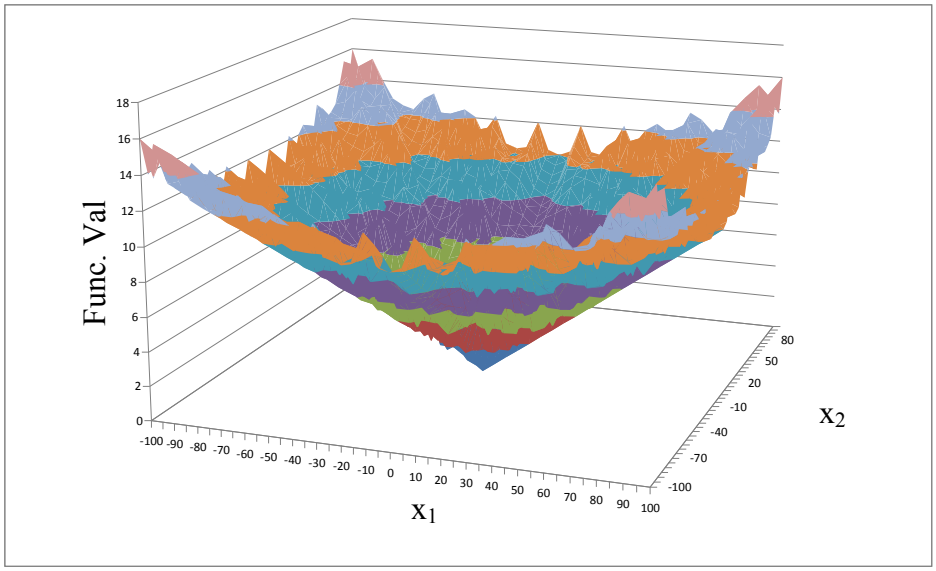


Figure 8.5 Salomon

8.1.6 Whitley

$$f(x) = \sum_{k=0}^{D-1}\sum_{j=0}^{D-1}\left(\frac{y^2{}_{j,k}}{4000} - \cos(y_{j,k}) + 1\right)$$

$$y_{j,l} = 100\left(x_k - x_j{}^2\right)^2 + \left(1 - x_j\right)^2$$

$$-100 \le x_j \le 100, \quad j = 0,1,.....D-1$$

$$f(x^*) = 0, \qquad x^*{}_j = 1, \quad \varepsilon = 1.0x10^{-6}$$

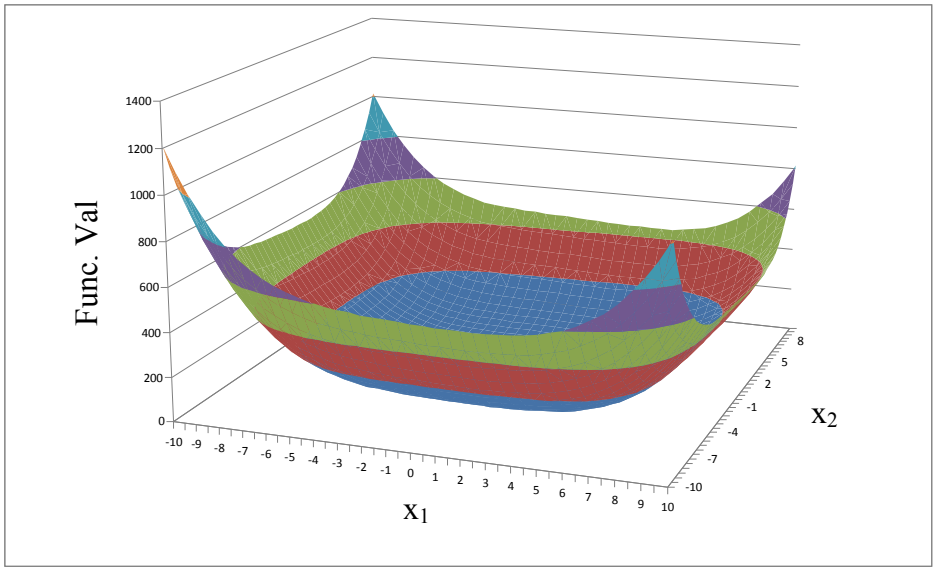


Figure 8.6 Whitley

8.1.7 Ackley Function

- Number of variables: *n* variables.
- Definition:

$$f(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{-\frac{1}{n}\sum \cos(2\pi x_i)}$$

- Search domain: $-15 \le x_i \le 30,\ i = 1, 2, \ldots, n.$
- Number of local minima: several local minima.
- The global minimum: $\mathbf{x}^* = (0, \ldots, 0), f(\mathbf{x}^*) = 0.$
- Function graph: for $n = 2$.

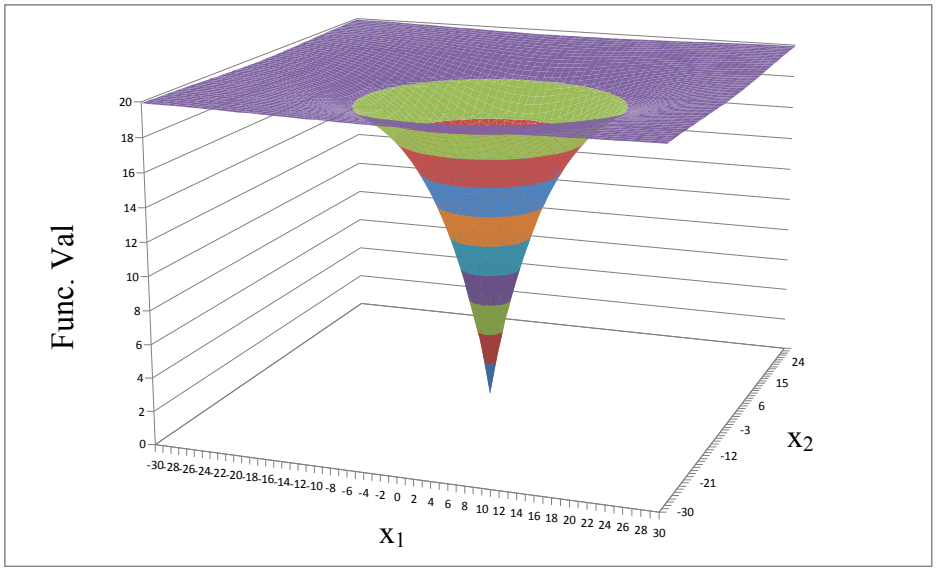


Figure 8.7 Ackley Function

8.1.8 Beale Function

- Number of variables: $n = 2$.
- Definition:

$$f(x) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^{\,2})^2 + (2.626 - x_1 + x_1 x_2^{\,3})^2$$

- Search domain: $-4.5 \le x_i \le 4.5,\ i = 1, 2.$
- The global minimum: $\mathbf{x}^* = (3, 0.5), f(\mathbf{x}^*) = 0.$
- Function graph:

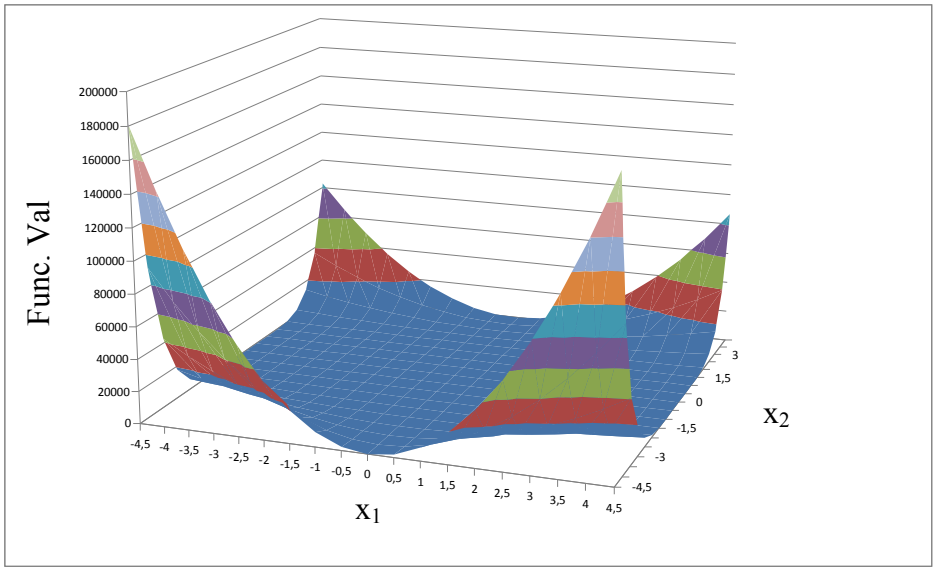


Figure 8.8 Beale Function

8.1.9 Bohachevsky Functions

- Number of variables: $n = 2$.
- Definition:

$$f_1(x) = x_1^{\,2} + 2x_2^{\,2} - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$$

$$f_2(x) = x_1^{\,2} + 2x_2^{\,2} - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$$

$$f_3(x) = x_1^{\,2} + 2x_2^{\,2} - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$$

- Search domain: $-100 \le x_i \le 100,\ i = 1, 2.$
- The global minima: $x^* = (0, 0), f_j(x^*) = 0, j = 1, 2, 3.$
- Function graph: for the first function.

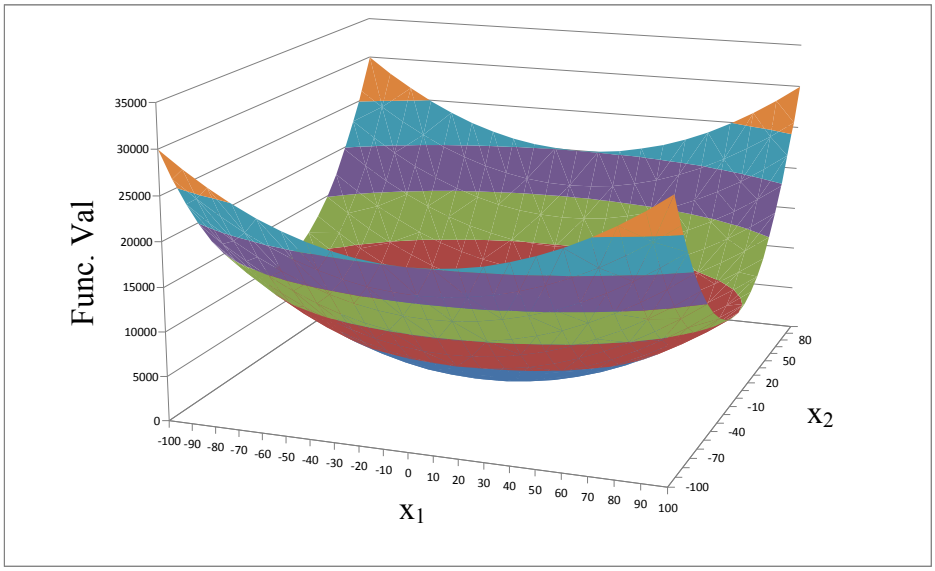


Figure 8.9 Bohachevsky Function

8.1.10 Booth Function

- Number of variables: $n = 2$.
- Definition:

$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

- Search domain: $-10 \le x_i \le 10,\ i = 1, 2.$
- Number of local minima: several local minima.
- The global minimum: $\mathbf{x}^* = (1, 3),\ f(\mathbf{x}^*) = 0.$
- Function graph: for $n = 2$.

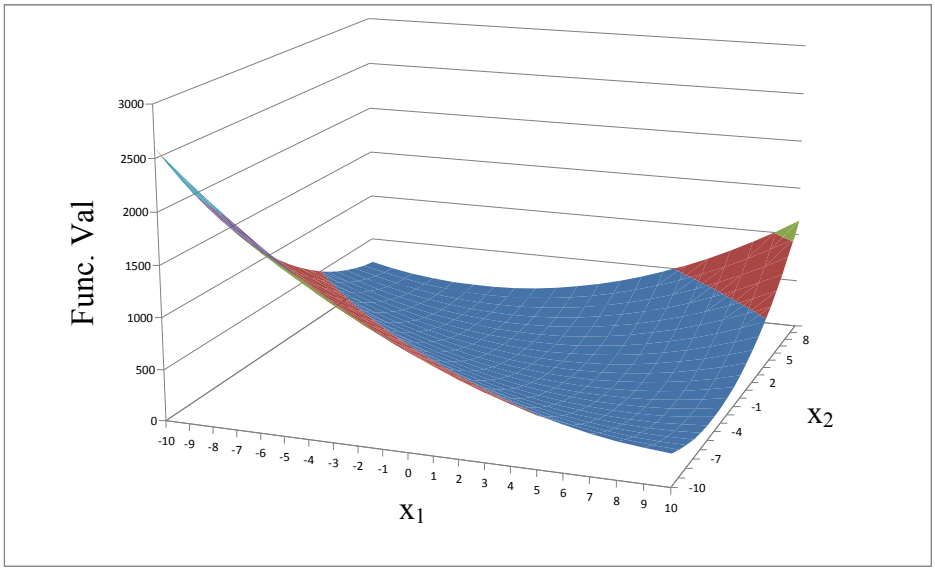


Figure 8.10 Booth Function

8.1.11 Colville Function

- Number of variables: $n = 4$.
- Definition:

$$f(x) = 100(x_1^{\,2} - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^{\,2} - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$$

- Search domain: $-10 \le x_i \le 10$, $i = 1, \ldots, 4$.
- The global minimum: $\mathbf{x}^* = (1, \ldots, 1)$, $f(\mathbf{x}^*) = 0$.

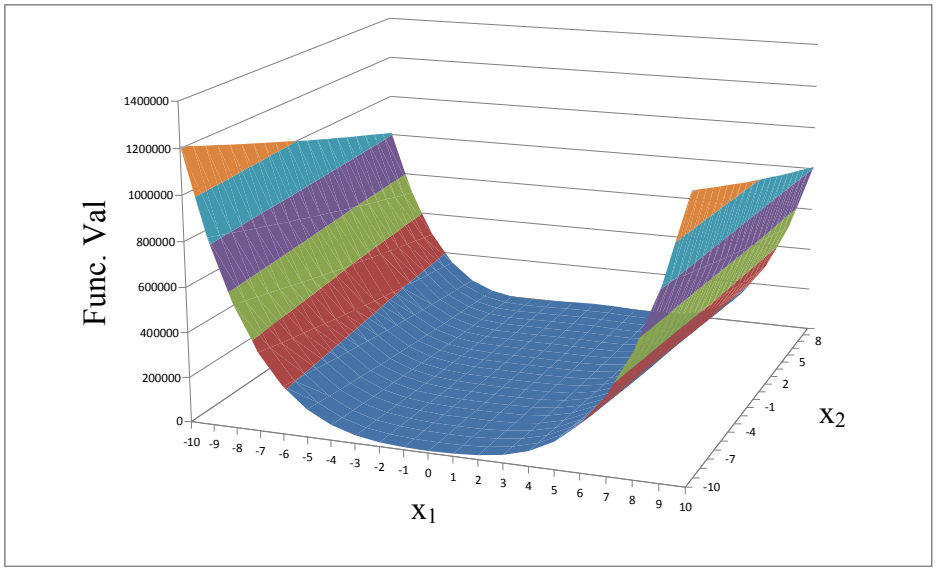


Figure 8.11 Colville Function

8.1.12 Dixon & Price Function

- Number of variables: *n* variables.
- Definition:

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^{\,2} - x_{i-1})^2$$

- Search domain: $-10 \le x_i \le 10,\ i = 1, 2, \ldots, n.$
- The global minima: $f(\mathbf{x}^*) = 0.$
- Function graph: for $n = 2.$

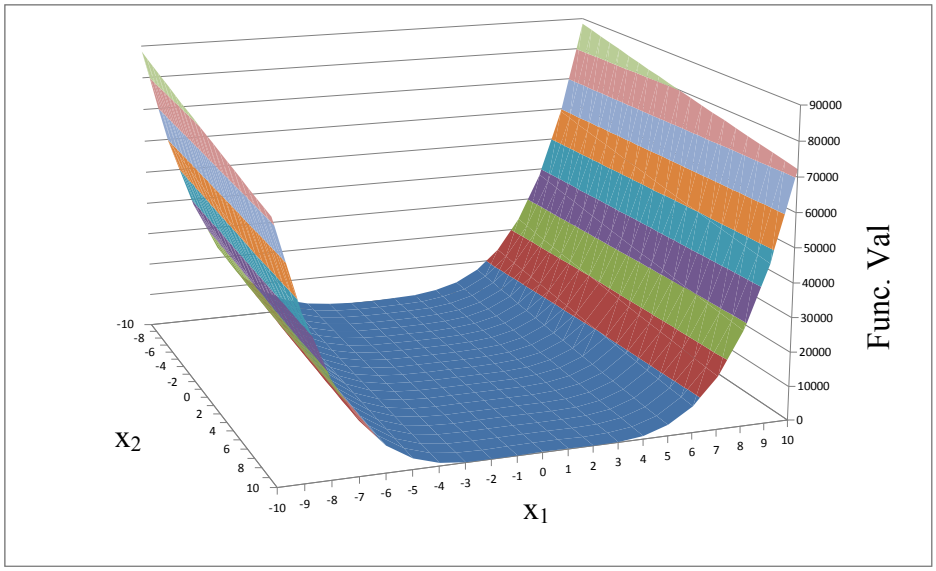


Figure 8.12 Dixon and Prince Function

8.1.13 Easom Function

- Number of variables: $n = 2$.
- Definition:

$$f(x) = -\cos(x_1)\cos(x_2)e^{\left[-(x_1-\pi)^2-(x_2-\pi)^2\right]}$$

- Search domain: $-100 \le x_i \le 100,\ i = 1, 2.$
- Number of local minima: several local minima.
- The global minima: $x^* = (\pi, \pi), f(x^*) = -1.$
- Function graph:

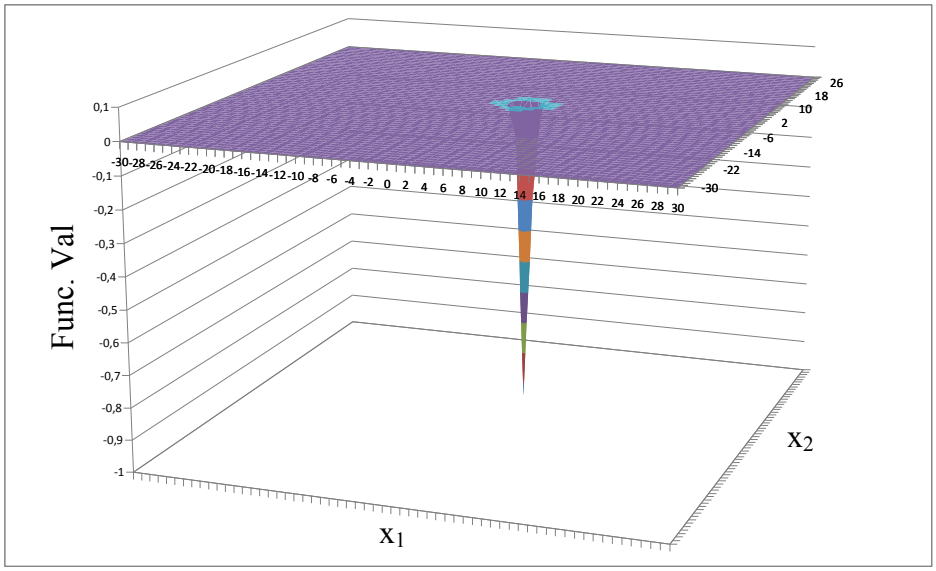


Figure 8.13 Easom Function

8.1.14 Goldstein & Price Function

- Number of variables: n =2.
- Definition:

$$f(x) = (1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))$$
$$(30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$$

- Search domain: $-2 \le x_i \le 2$, $i = 1, 2$.
- Number of local minima: several local minima.
- The global minima: $x^* = (0, 1)$, $f(x^*) = 3$
- Function graph:

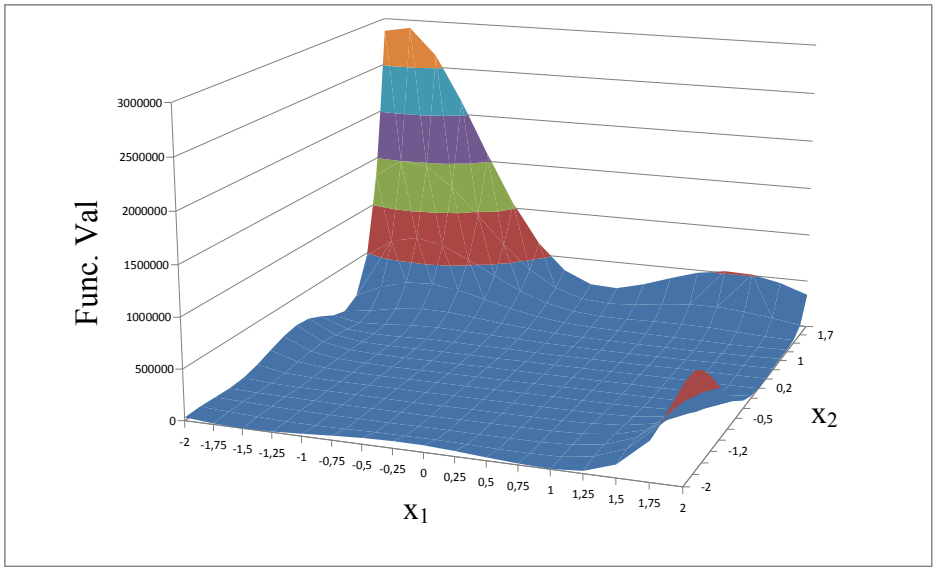


Figure 8.14 Goldstein & Price Function

8.1.15 Hump Functions

- Number of variables: $n = 2$.
- Definition:

$$f(x) = 1.032 + 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

- Search domain: $-5 \le x_i \le 5,\ i = 1, 2.$
- Number of local minima: no local minimum except the global ones.
- The global minima: $\mathbf{x}^* = (0.0898, -0.7126), (-0.0898, 0.7126),$ $f(\mathbf{x}^*) = 0.$
- Function graph:

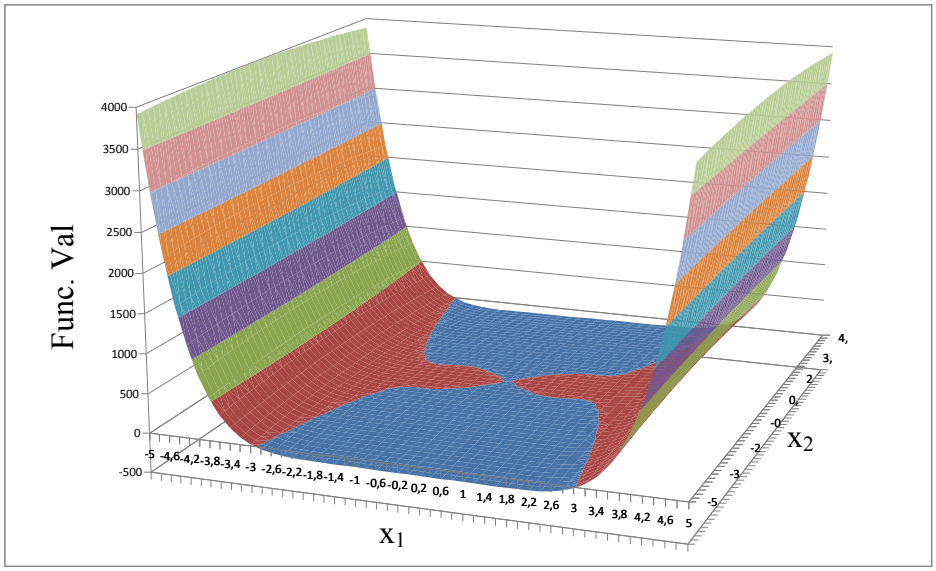


Figure 8.15 Humps Function

8.1.16 Matyas Function

- Number of variables: *2* variables.
- Definition:

$$f(x) = 0.26(x_1^{\ 2} + x_2^{\ 2}) - 0.48x_1x_2$$

- Search domain: $-10 \le x_i \le 10,\ i = 1, 2.$
- Number of local minima: no local minima except the global one.
- The global minima: $x^* = (0, 0), f(x^*) = 0.$
- Function graph:

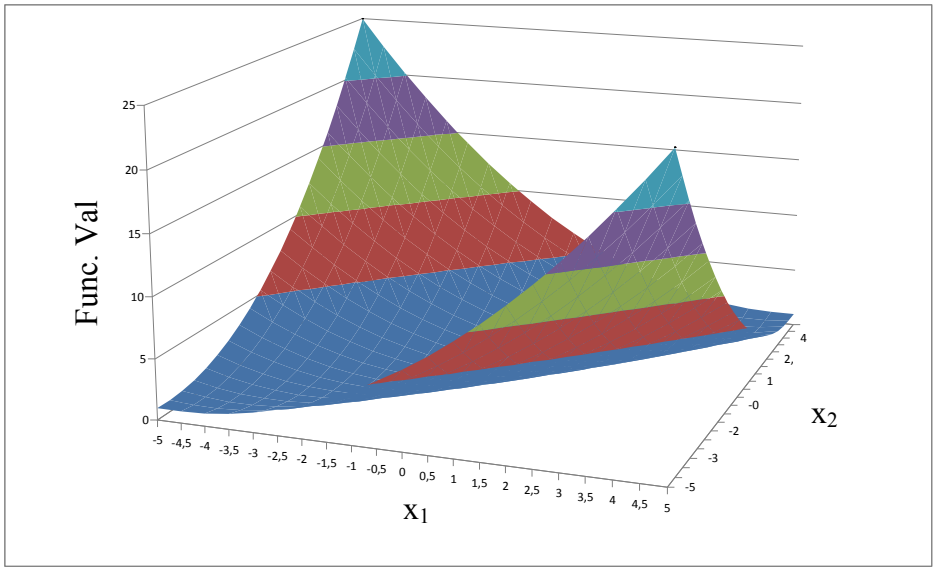


Figure 8.16 Matyas Function

8.1.17 Michalewics Function

- Number of variables: *n* variables.
- Definition:

$$f(x_1, x_2) = -\sum_{j=1}^{2} \sin(x_j)\left(\sin\left(\frac{jx_j^{\,2}}{\pi}\right)\right)^{2m} \quad ; m = 10$$

- Search domain: $0 \le x_i \le \pi$, $i = 1, 2, \ldots, n$.
- Number of local minima: several local minima.
- The global minima: at $n=2$, $f(x^*) = -1.8013$.
  at $n=5$, $f(x^*) = --4.687658$.
  at $n=10$, $f(x^*) = -9.66015$.
- Function graph: for $n = 2$.

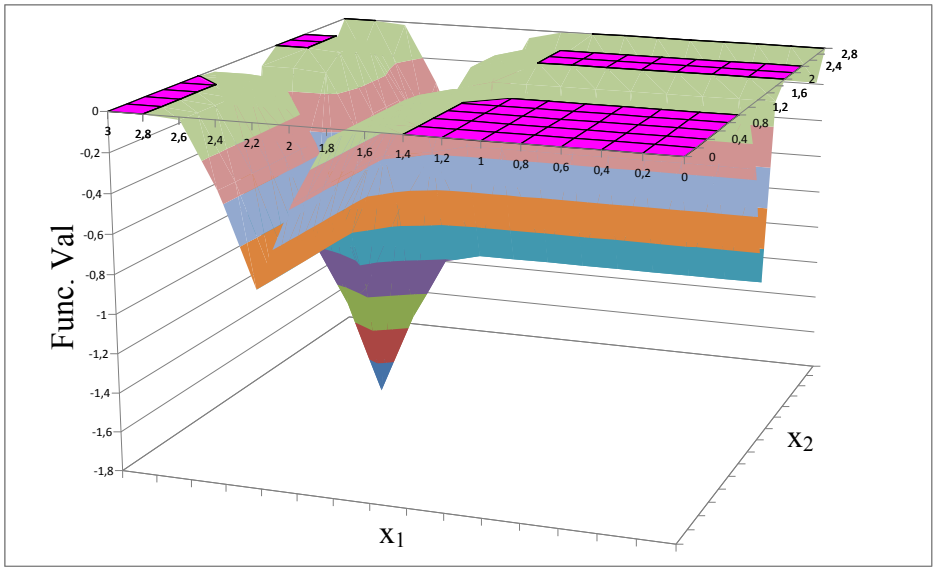


Figure 8.17 Michalewics Function

8.1.18 Rastrigin Function

- Number of variables: *n* variables.
- Definition

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^{\,2} - 10\cos(2\pi x_i))$$

- Search domain: $-5.12 \le x_i \le 5.12,\ i = 1, 2, \ldots, n.$
- Number of local minima: several local minima.
- The global minima: $x^* = (0, \ldots, 0),\ f(x^*) = 0.$
- Function graph: for $n = 2$.

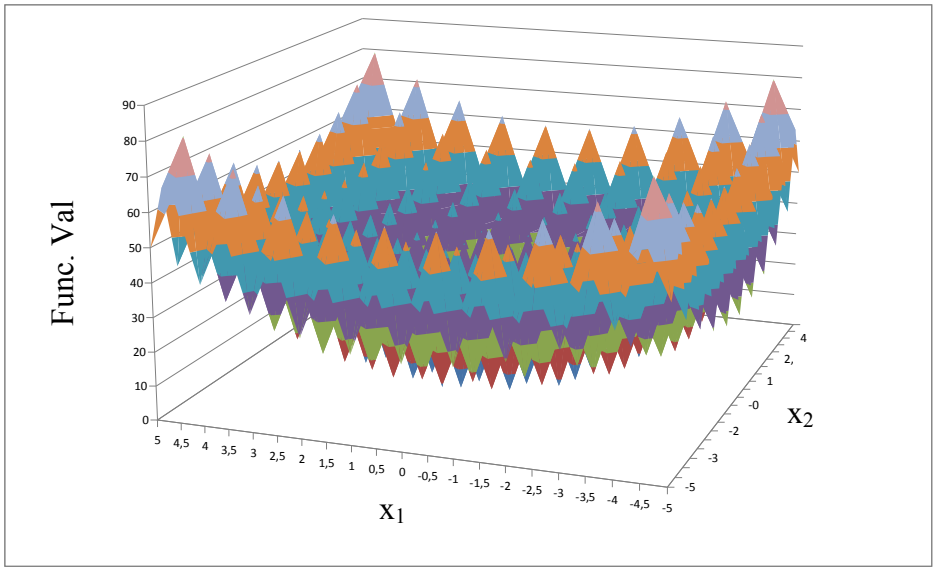


Figure 8.18 Rastrigin Function

8.1.19 Rosenbrock Function

- Number of variables: *n* variables.
- Definition:

$$f(x) = \sum_{i=1}^{n-1}\left[100\left(x_i^{\,2} - x_{i+1}\right)^2 + \left(x_i - 1\right)^2\right]$$

- Search domain: $-5 \le x_i \le 10,\ i = 1, 2, \ldots, n.$
- Number of local minima: several local minima.
- The global minima: $x^* = (1, \ldots, 1),\ f(x^*) = 0.$
- Function graph: for $n = 2$.

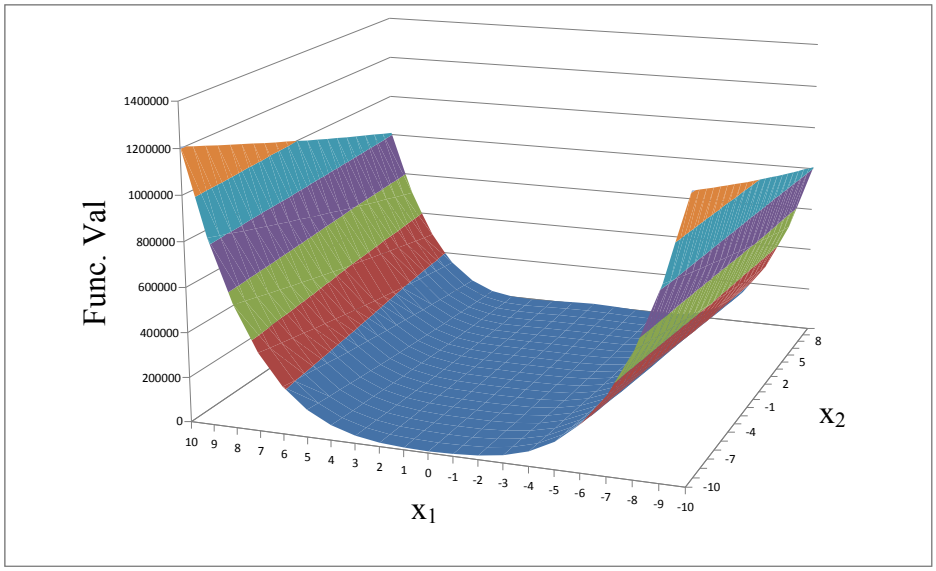


Figure 8.19 Rosenbrock Function

8.1.20 Shubert Function

- Number of variables: $n = 2$.
- Definition:

$$f(x) = \left(\sum_{i=1}^{5} i \cos((i+1)x_1 + i) \right) \left(\sum_{i=1}^{5} i \cos((i+1)x_2 + i) \right)$$

- Search domain: $-10 \le x_i \le 10,\ i = 1, 2.$
- Number of local minima: several local minima.
- The global minima: 18 global minima $f(x^*) = -186.7309$.
- Function graph

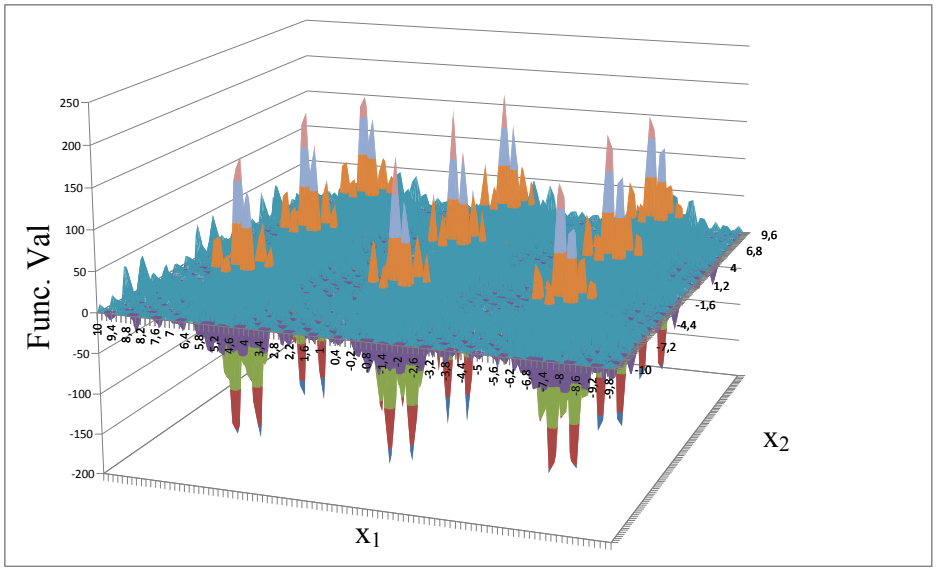


Figure 8.20 Shubert Function

8.1.21 Sphere Function

- Number of variables: *n* variables.
- Definition:

$$f(x) = \sum_{i=1}^{n} x_i^{\,2}$$

- Search domain: $-5.12 \le x_i \le 5.12,\ i = 1, 2, \ldots, n.$
- Number of local minima: no local minimum except the global one.
- The global minima: $x^* = (0, \ldots, 0),\ f(x^*) = 0.$
- Function graph: for $n = 2$.

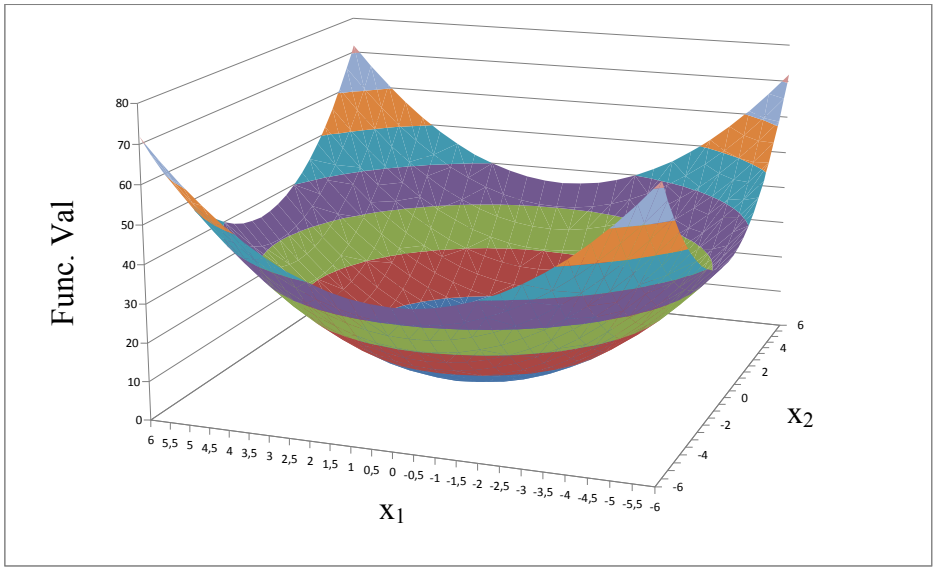


Figure 8.21 Sphere Function

8.1.22 Zakharov Function

- Number of variables: *n* variables.
- Definition:

$$Z_n(x) = \sum_{i=1}^{n} x_i^{\,2} + \left(\sum_{i=1}^{n} 0.5 i x_i^{\,2}\right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i^{\,2}\right)^4$$

- Search domain: $-5 \le x_i \le 10,\ i = 1, 2, \ldots, n.$
- Number of local minima: no local minimum except the global one.
- The global minima: $x^* = (0, \ldots, 0),\ Z_n(x^*) = 0.$
- Function graph: for $n = 2$.

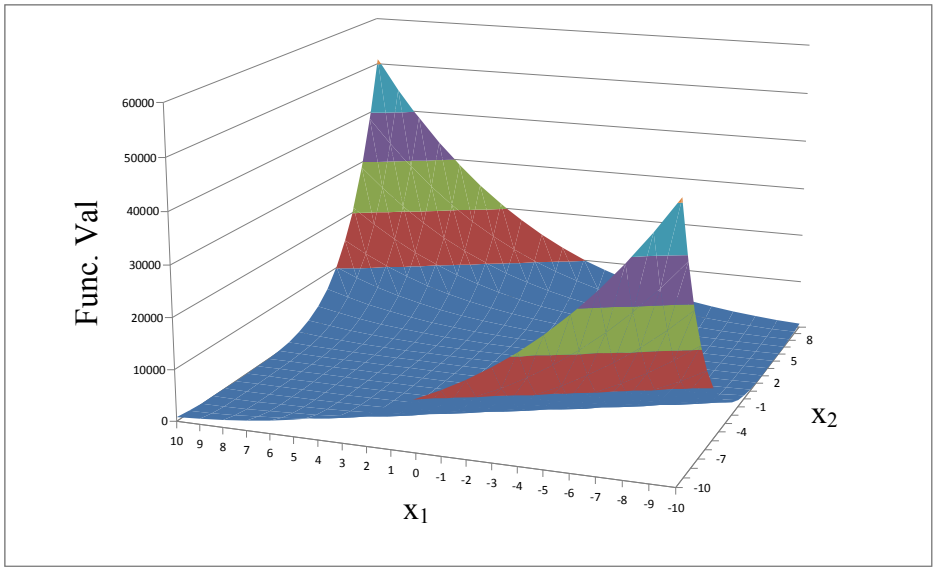


Figure 8.22 Zakharov Function

8.2 Constraint Benchmark Test Functions

8.2.1 G1 Problem

$$\min f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^{\,2} - \sum_{i=5}^{13} x_i$$

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0$$
$$g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0$$
$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0$$
$$g_4(x) = -8x_1 + x_{10} \le 0$$
$$g_5(x) = -8x_2 + x_{11} \le 0$$
$$g_6(x) = -8x_3 + x_{12} \le 0$$
$$g_7(x) = -2x_4 - x_5 + x_{10} \le 0$$
$$g_8(x) = -2x_6 - x_7 + x_{11} \le 0$$
$$g_9(x) = -2x_8 - x_9 + x_{12} \le 0$$
$$x_i \ge 0$$
$$x_i \le 1$$

r Number of variables: *13* variables.

r Search Space: $0 \le x_i \le u_i$, $i = 1, 2, \ldots, n$,

$$u = (1,1,\ldots,1,100,100,100,1)$$

r The global minima: $x^* = (1,1,\ldots,1,3,3,3,1)$, $f(x^*) = -15$.

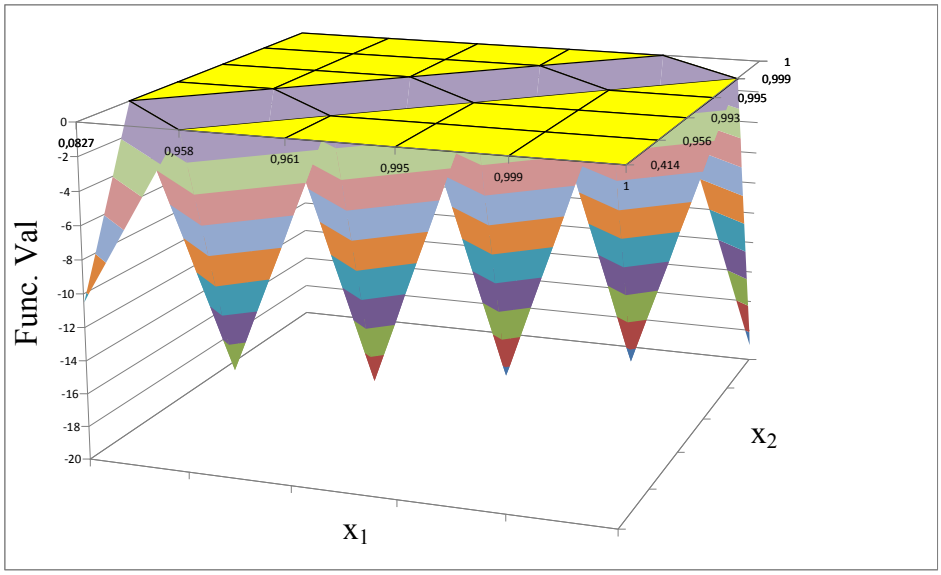


Figure 8.23 G1 Problem

8.2.2 G2 Problem

$$Maxf(x) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^{\,2}}} \right|$$

$$g_1(x) = -\prod_{i=1}^{n} x_i + 0.75 \le 0$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \le 0$$

r Number of variables: n variables.

r Search Space: $0 \le x_i \le 10$, $i = 1, 2, \ldots, n$.

r Best known: at $n = 20$, $f(x^*) = 0.803619$.

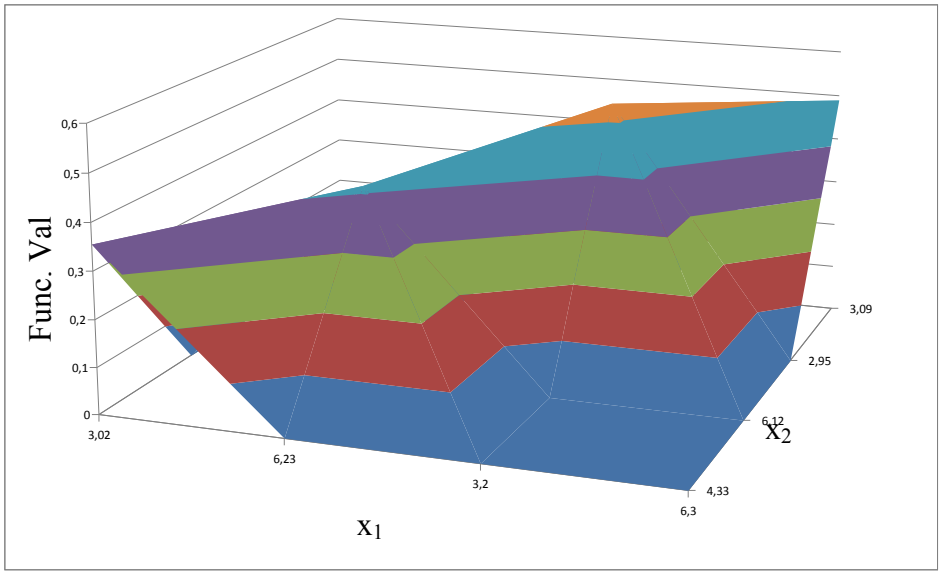


Figure 8.24 G2 Problem

8.2.3 G3 Problem

$$\min f(x) = 5.3578547x_3{}^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

$$g_1(x) = u(x) - 92 \le 0$$
$$g_2(x) = -u(x) \le 0$$
$$g_3(x) = v(x) - 110 \le 0$$
$$g_4(x) = -v(x) + 90 \le 0$$
$$g_5(x) = w(x) - 25 \le 0$$
$$g_6(x) = -w(x) + 20 \le 0$$

where

$$u(x) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5$$
$$v(x) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3{}^2$$
$$w(x) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4$$

r Number of variables: *5* variables.

r Search Space: $l_i \le x_i \le u_i$, $i = 1, \ldots, 5$,

$l = (78,33,27,27,27)$, $u=(102,45,45,45,45)$.

r The global minima: $x^* = (78,33,29.995,45,36.7758)$,

$f(x^*) = -30665.539$.

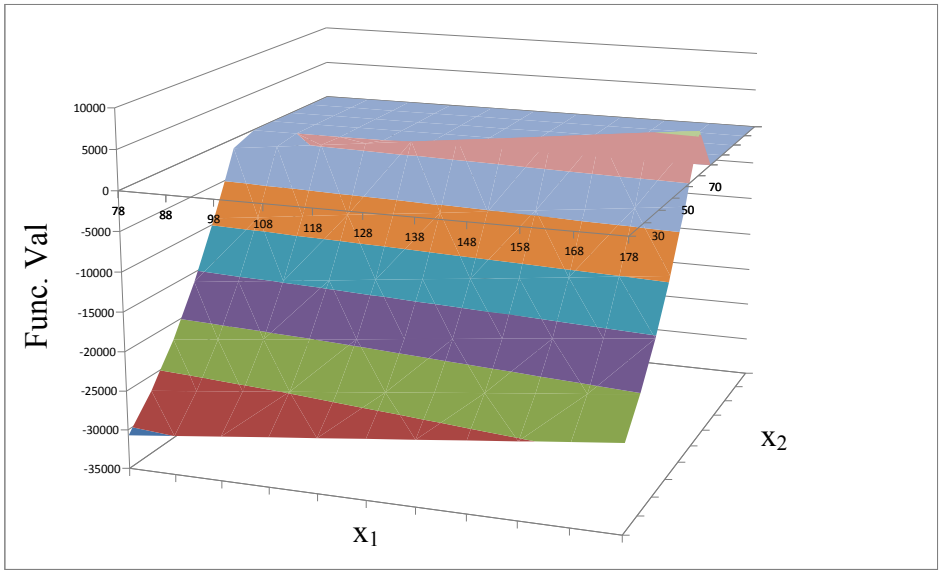


Figure 8.24 G3 Problem

8.2.4 G4 Problem

$$\min f(x) = 3x_1 + 10^{-6} x_1^{\,3} + 2x_2 + \frac{2}{3} 10^{-6} x_2^{\,3}$$

$$g_1(x) = x_3 - x_4 - 0.55 \le 0$$

$$g_2(x) = x_4 - x_3 - 0.55 \le 0$$

$$h_1(x) = 1000\left[\sin(-x_3 - 0.25) + \sin(-x_4 - 0.25)\right] + 894.8 - x_1 = 0$$

$$h_2(x) = 1000\left[\sin(x_3 - 0.25) + \sin(x_3 - x_4 - 0.25)\right] + 894.8 - x_2 = 0$$

$$h_3(x) = 1000\left[\sin(x_4 - 0.25) + \sin(x_4 - x_3 - 0.25)\right] + 1294.8 = 0$$

r Number of variables: *4* variables.

r Search Space: $l_i \leq x_i \leq u_i$, $i = 1, \ldots, 4$,

$l = (0,0,-0.55,-0.55)$, $u=(1200,1200,0.55,0.55)$.

r Best known: $x^* = (679.9453,1026,0.118876,-0.3962336)$,

$f(x^*) = 5126.4981$.

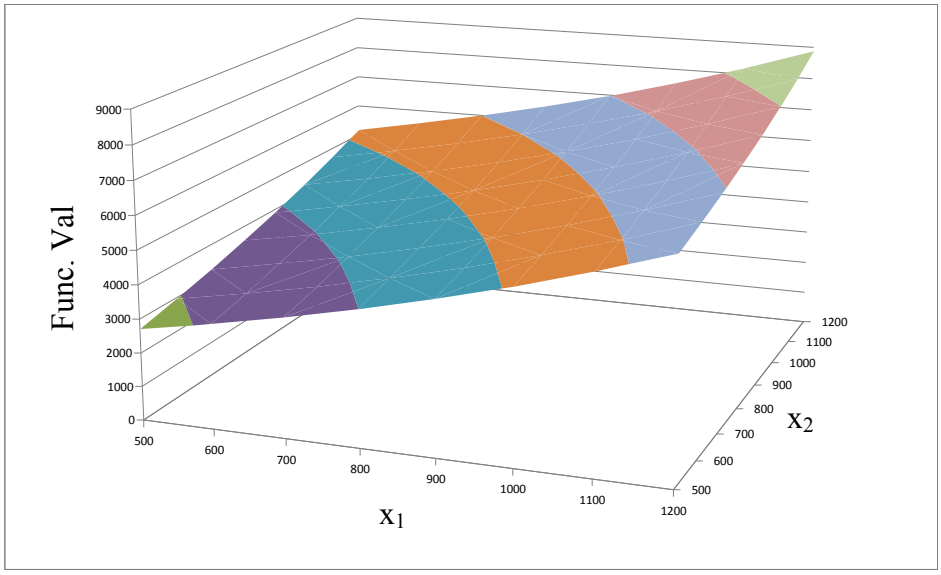


Figure 8.25 G4 Problem

8.2.5 G6 Problem

$$\min f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2$$
$$+ 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

$$g_1(x) = 4x_1 + 5x_2 - 3x_7 + 9x_8 - 105 \le 0$$
$$g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \le 0$$
$$g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \le 0$$
$$g_4(x) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^{\,2} - 7x_4 - 120 \le 0$$
$$g_5(x) = 5x_1^{\,2} + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \le 0$$
$$g_6(x) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^{\,2} - x_6 - 30 \le 0$$
$$g_7(x) = x_1^{\,2} + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \le 0$$
$$g_8(x) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \le 0$$

r Number of variables: *10* variables.

r Search Space: $-10 \le x_i \le 10,\ i = 1, 2, \ldots, 10.$

r The global minima: $x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$,

$$f(x^*) = 24.3062091.$$

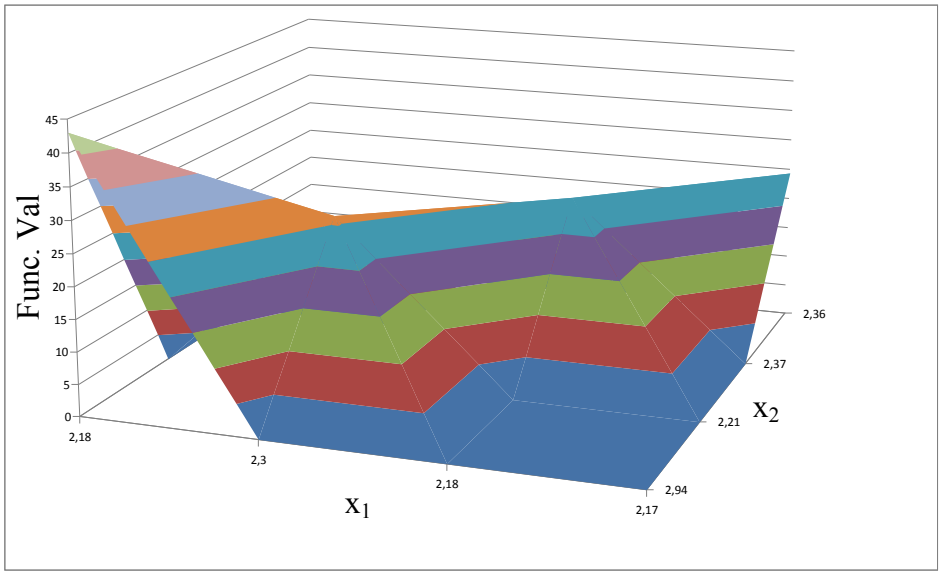


Figure 8.26 G6 Problem

8.2.6 G7 Problem

$$\begin{aligned}\max_{x}\ & f(x) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)},\\ \text{s.t.}\ & g_1(x) = x_1^2 - x_2 + 1 \le 0,\\ & g_2(x) = 1 - x_1 + (x_2-4)^2 \le 0.\end{aligned}$$

r Number of variables: *2* variables.

r Search Space: $0 \le x_i \le 10,\ i = 1, 2.$

r The global minima: x = (1.2279713, 4.2453733) , f (x) = 0.095825.

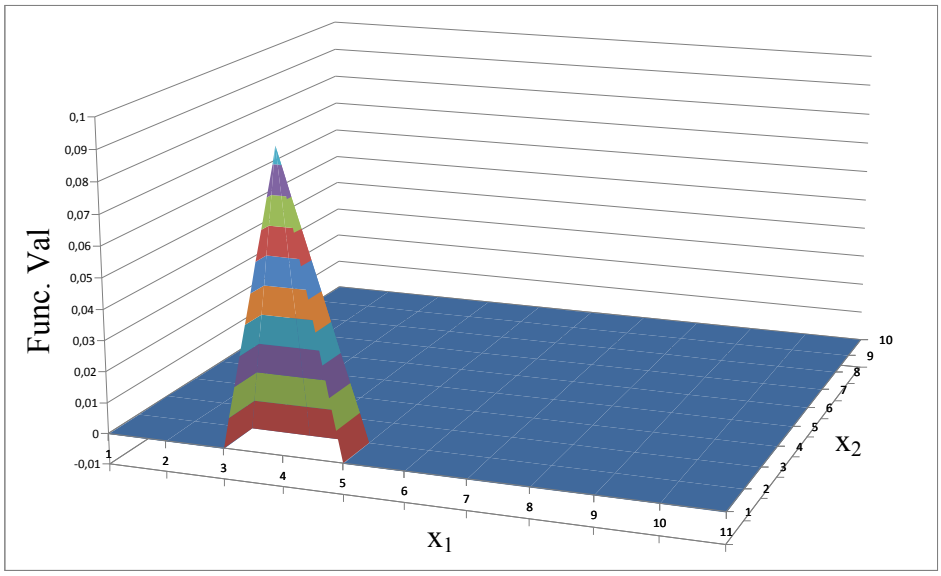


Figure 8.27 G7 Problem

8.2.7 G8 Problem

$$\begin{aligned}\min_{x}\ & f(x) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7\\ & \quad -10x_6 - 8x_7,\\ \text{s.t.}\ & g_1(x) = 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127 \le 0,\\ & g_2(x) = 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 \le 0,\\ & g_3(x) = 23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 \le 0,\\ & g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0.\end{aligned}$$

r Number of variables: *7* variables.

r Search Space: $-10 \leq x_i \leq 10,\ i = 1, 2, \ldots, 7.$

r The global minima: x = (2.330499, 1.951372,−0.4775414, 4.365726,−0.6244870, 1.038131,1.594227), f (x) = 680.6300573.

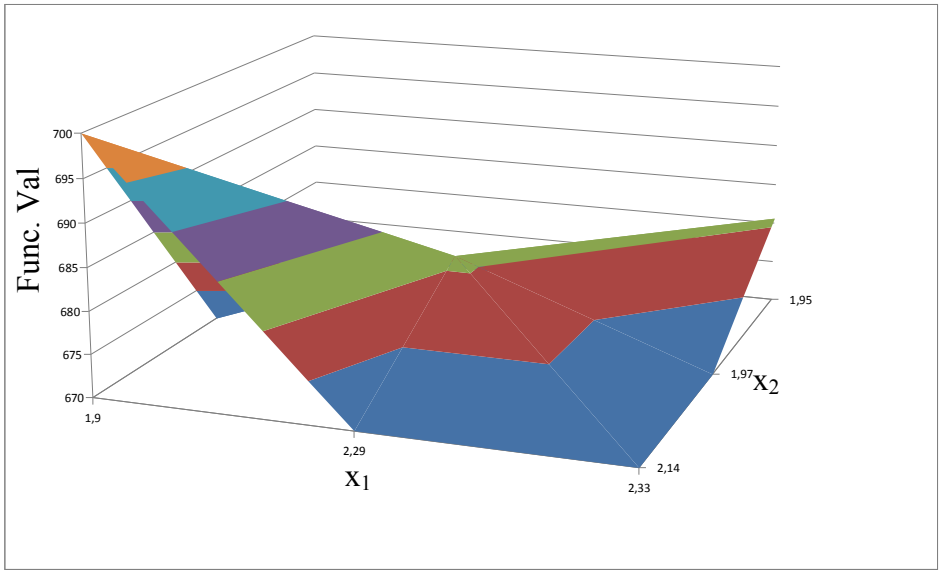


Figure 8.28 G8 Problem

8.2.8 G9 Problem

$$\min f(x) = e^{x_1 x_2 x_3 x_4 x_5}$$

$$h_1(x) = x_1^{\ 2} + x_2^{\ 2} + x_3^{\ 2} + x_4^{\ 2} + x_5^{\ 2} - 10 = 0$$

$$h_2(x) = x_2\ x_3\ - 5x_4\ x_5\ = 0$$

$$h_3(x) = x_1^{\ 3} + x_2^{\ 3} + 1 = 0$$

r Number of variables: *n* variables.

r Search Space: $l_i \le x_i \le u_i$, $i = 1, \ldots, 5$,

$u = (2.3, 2.3, 3.2, 3.2, 3.2)$, $l = -u$.

r The global minima: x = (−1.717143, 1.595709, 1.827247, −0.7636413, −0.763645) , f (x) = 0.0539498.

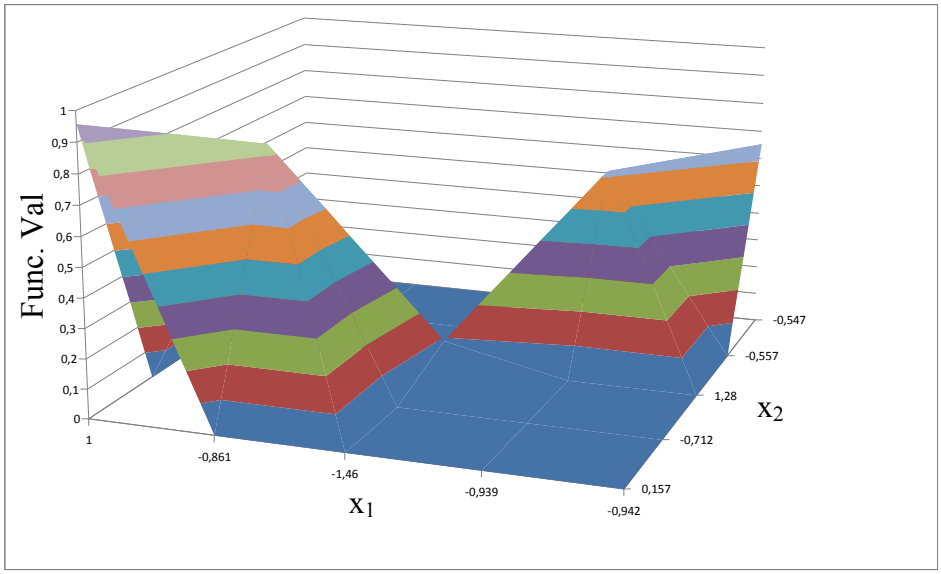


Figure 8.29 G9 Problem

8.2.9 G10 Problem

$$\max f(x) = \left(\sqrt{n}\right)^n \prod_{i=1}^{n} x_i$$

$$h_1(x) = \sum_{i=1}^{n} x_i^{\,2} - 1 = 0$$

- Number of variables: *n* variables.
- Search Space: $0 \le x_i \le 1$, $i = 1, 2, \ldots, n$.
- The global minima: $x^* = (1/n^{0.5}, \ldots, 1/n^{0.5})$, $f(x^*) = 1$.

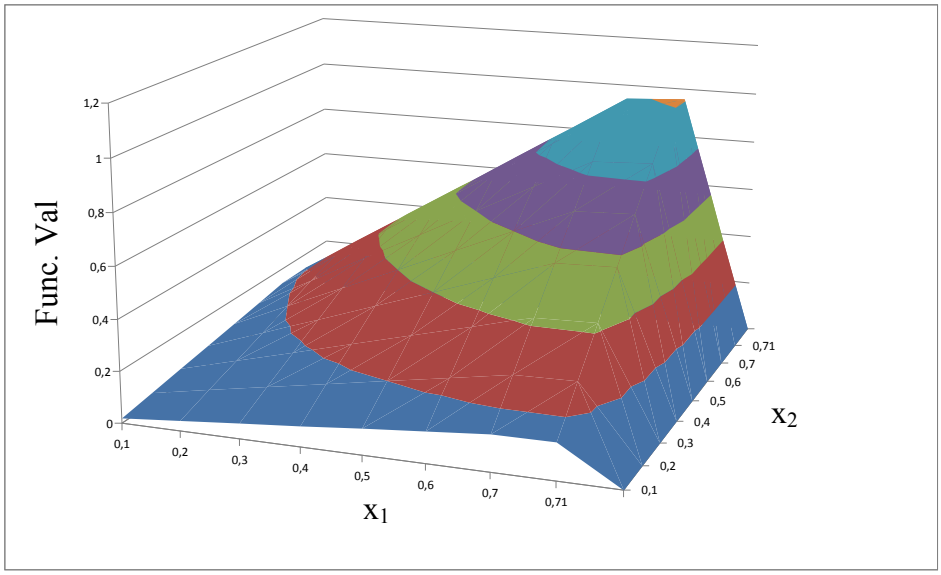


Figure 8.30 G10 Problem

8.2.9 G11 Problem

$$\min f(x) = x_1\ x_2\ x_3$$

$$g_1(x) = -1 + 0.0025(x_4 + x_6) \le 0$$
$$g_2(x) = -1 + 0.0025(-x_4 + x_5 + x_7) \le 0$$
$$g_3(x) = -1 + 0.01(-x_5 + x_8) \le 0$$
$$g_4(x) = 100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \le 0$$
$$g_5(x) = x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \le 0$$
$$g_6(x) = x_3x_5 - x_3x_8 - 1250x_5 + 1250000 \le 0$$

- Number of variables: *8* variables.
- Search Space: $l_i \leq x_i \leq u_i$, $i = 1, \ldots, 8$,

  l = 10*(10, 100, 100, 1, 1, 1, 1, 1),

  u=1000*(10, 10, 10, 1, 1, 1, 1, 1).
- The global minima: x = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979), f (x) = 7049.3307.

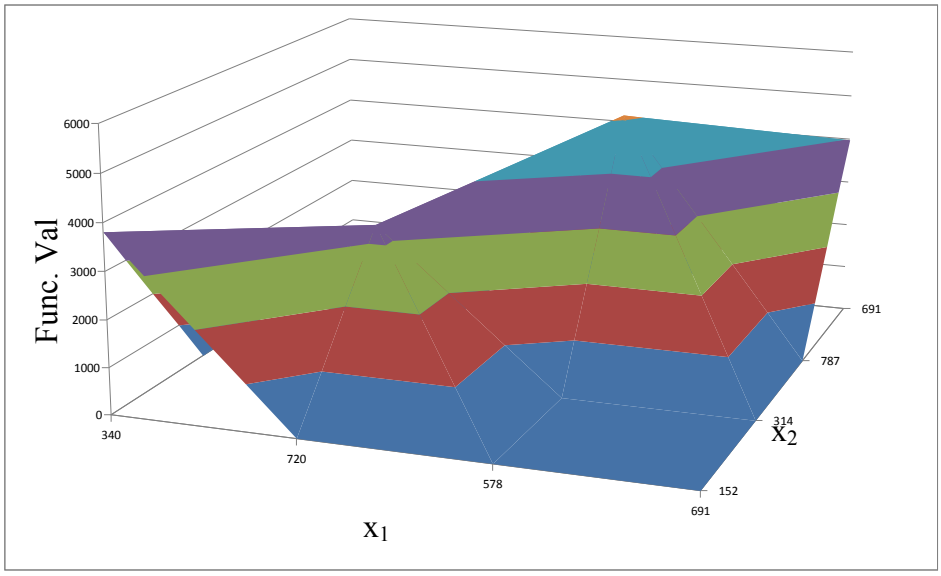


Figure 8.31 G11 Problem

8.3 Chemical Engineering Problems

A large number of process synthesis and design problems in chemical engineering can be modeled as mixed integer nonlinear programming (MINLP) problems. They involve continuous (floating point) and integer variables. A common feature of this class of mathematical problems is the potential existence of non-convexities due to the particular form of the objective function and/or the set of constraints. Due to their combinatorial nature, these problems are considered to be difficult. In recent years, evolutionary algorithms (EAs) are gaining popularity for finding the optimal solution of nonlinear multimodal problems encountered in many engineering disciplines. To illustrate the applicability and efficiency of modified differential evolution (MDE), seven test problems on process synthesis and design have been solved. These problems arise from the area of chemical engineering, and represent difficult nonconvex optimization problems, with continuous and discrete variables. The performance of MDE is compared with that of DE.

8.3.1 Process synthesis problem

This example has a non-linear constraint and has been proposed by Kocis and Grossmann (1988). It has also been solved by other authors (Floudas et al., 1989; Ryoo and Sahinidis, 1995; Cardoso et al., 1997; Costa and Oliviera, 2001)

$$Min\ f(x,y) = 2x + y$$

$$S.t$$

$$1.25 - x^2 - y \le 0$$

$$x + y \le 1.6$$

$$0 \le x \le 1.6$$

$$y \in \{0,1\}$$

The global optimum is (x, y; f) = (0.5, 1; 2).

8.3.2 Process synthesis and design problem

This problem, with a non-linear constraint is proposed by Kocis and Grossmann (1988) and is also studied by Salcedo (1992), Cardoso et al. (1997), Costa and Oliviera (2001)

$$Min\ f(x_1, x_2, y) = -y + 2x_1 + x_2$$

$$S.t$$

$$x_1 - 2\exp(-x_2) = 0$$

$$-x_1 + x_2 + y \le 0$$

$$0.5 \le x_1 \le 1.4$$

$$y \in \{0,1\}$$

The global optimum is $(x_1, x_2, y; f) = (1.375, 0.375, 1; 2.124)$.

8.3.3 Process flow sheeting problem

This problem was first studied by Floudas (1995) and is nonconvex because of the first constraint. It has also been solved by Cardoso et al. (1997), and Costa and Oliviera (2001)

$$Min\ f(x, y) = -0.7y + 5(x_1 - 0.5)^2 + 0.8$$

$$S.t$$

$$-\exp(x_1 - 0.2) - x_2 \le 0$$

$$x_2 + 1.1y \le -1$$

$$x_1 - y \le 0.2$$

$$0.2 \le x_1 \le 1$$

$$-2.22554 \le x_2 \le -1$$

$$y \in \{0,1\}$$

The global optimum is $(x_1, x_2, y; f) = (0.94194, -2.1, 1; 1.07654)$.

8.3.4 Two reactor problem

It has been taken from Kocis and Grossmann (1989). The objective here is to select one between two candidate reactors (as shown in Fig. 1) in order to minimize the production cost. Also, it has been solved by Diwekar et al. (1992), Diwekar and Rubin (1993), Cardoso et al. (1997), and Costa and Oliviera (2001)

$$Min\, f(x, y_1, y_2, v_1, v_2) = 7.5y_1 + 5.5y_2 + 7v_1 + 6v_2 + 5x$$

$$S.t$$

$$y_1 + y_2 = 1$$

$$z_1 = 0.9[1 - \exp(-0.5v_1)]x_1$$

$$z_2 = 0.8[1 - \exp(-0.4v_2)]x_2$$

$$z_1 + z_2 = 10$$

$$x_1 + x_2 = x$$

$$z_1 y_1 + z_2 y_2 = 10$$

$$v_1 \le 10\, y_1$$

$$v_2 \le 10\, y_2$$

$$x_1 \le 20\, y_1$$

$$x_2 \le 20\, y_2$$

$$x_1,\ x_2,\ z_1,\ z_2,\ v_1, v_2 \ge 0$$

$$y_1, y_2 \in \{0,1\}$$

The binary variables $y1$ and $y2$ denote the existence (nonexistence) of reactor 1 and 2 when their value is 1 (0). In the objective function, there

are fixed charges for purchasing reactor 1 (7.5) or reactor 2 (5.5), linear terms in $v1$ and $v2$ (reactor volumes) and the purchase price for raw material x. The two nonlinear equations are the input–output relations for the reactors which define the output flows ($z1$ and $z2$) in terms of the input flows ($x1$ and $x2$) and the volumes. The raw material x is split into the reactor input flows $x1$ and $x2$; a total demand of 10 units must be met by the output flows $z1$, $z2$. The next four inequalities are logical constraints which insure that if a given reactor does not exist (e.g. $y1$ = 0), then the corresponding volume and feed stream are zero. The last constraint requires that either reactor 1 or 2 be selected. The suboptimal solution corresponding to (y1, y2)=(0, 1) has an objective function value of 107.376 at (x1, x2) = (0.0, 15.0) and (v1, v2) = (0.0, 4.479).

The global optimum is:
$(x, y_1, y_2, v_1, v_2; f) = (13.36227, 1, 0, 3.514237, 0; 99.245209)$.

8.3.5 Process synthesis problem

This problem was studied by Floudas et al. (1989), Salcedo (1992), Ryoo and Sahinidis (1995), Cardoso et al. (1997), and Costa and Oliviera (2001). This problem features nonlinearities in both continuous and binary variables and has seven degrees of freedom.

$$Min\, f(x_1, x_2, x_3, y_1, y_2, y_3, y_4) = (y_1 - 1)^2 + (y_2 - 1)^2 + (y_3 - 1)^2 - Ln(y_4 + 1) + (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2$$

$S.t$

$$y_1 + y_2 + y_3 + x_1 + x_2 + x_3 \le 5$$

$$y_3^2 + x_1^2 + x_2^2 + x_3^2 \le 5.5$$

$$y_1 + x_1 \le 1.2$$

$$y_2 + x_2 \le 1.8$$

$$y_3 + x_3 \le 2.5$$

$$y_4 + x_1 \le 1.2$$

$$y_2^2 + x_2^2 \le 1.64$$

$$y_3^2 + x_3^2 \le 4.25$$

$$y_2^2 + x_3^2 \le 4.64$$

$$x_1,\ x_2,\ x_3 \ge 0$$

$$y_1, y_2, y_3, y_4 \in \{0,1\}$$

The global optimum is;

$(x_1, x_2, x_3, y_1, y_2, y_3, y_4; f) = (0.2, 1.28062, 1.95448, 1, 0, 0, 1; 3.557473)$.

8.3.6 Optimal operation of alkylation unit

Alkylation process is common in the petroleum industry. A simplified process flow diagram of an alkylation process is shown in Fig. 3. The process model was described and solved by Sauer, Coville, and Burwick (1964) using successive linear programming. The process model seeks to

determine the optimum set of operating conditions for the process, based on a mathematical model, which allowed maximization of profit.

As shown in Figure 8.32, an olefin feed (100% butane), a pure isobutane recycle and a 100% isobutane make-up stream are introduced in a reactor together with an acid catalyst. The reactor product stream is then passed through a fractionators where the isobutane and the alkylate product are separated. The spent acid is also removed from the reactor. The variables are defined as shown in Table 2 along with the upper and lower bounds on each variable. The bounds represent economic, physical and performance constraints.

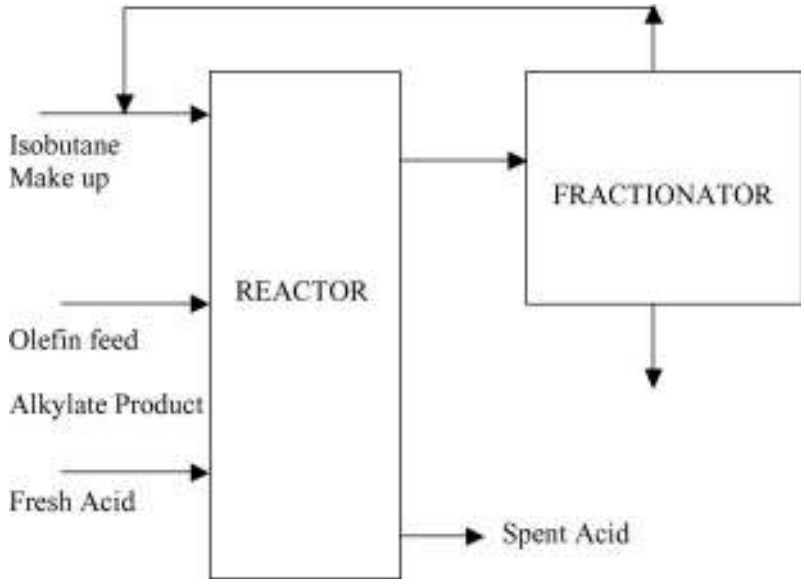


Figure 8.32 Flowsheet of Alkylation units

Profit function

The objective is to improve the octane number of some olefin feed by reacting it with isobutane in the presence of acid. The product of the reaction is distilled and the un-reacted is recycled back to the reactor. The objective function was defined in terms of alkylate product, or output value minus feed and recycle costs. Operating costs were not reflected in the function. The total profit ($ per day), to be maximized (Adjiman et al., 1998), is given as follows:

Table 8.3 Variables and their bounds

Variables and their bounds

Symbol	Variable	Lower Bound	Upper Bound
x_1	Olefin feed rate (barrels/day)	1500	2000
x_2	Acid addition rate (thousands of pounds/day)	1	120
x_3	Alkylate yield (barrels/day)	3000	3500
x_4	Acid strength (wt.%)	85	93
x_5	Motor octane no.	90	95
x_6	External isobutane-to-olefin ratio	3	12
x_7	F-4 performance no.	145	162

$$\textit{Max profit} = 1.715x_1 + 0.035x_1x_6 + 4.0565x_3 + 10x_2 - 0.063x_3x_5$$

Subjected to;

$$0.0059553571x_6^2x_1 + 0.88392857x_3 - 0.1175625x_6x_1 - x_1 \le 0$$

$$1.1088x_1 + 0.1303533x_1x_6 - 0.0066033x_1x_6^2 - x_3 \le 0$$

$$6.66173269x_6^2 + 172.39878x_5 - 56.596669x_4 - 191.20592x_6 - 10000 \le 0$$

$$1.08702x_6 + 0.32175x_4 - 0.03762x_6^2 - x_5 + 56.85075 \le 0$$

$$0.006198x_7x_4x_3 + 2462.3121x_2 - 25.125634x_2x_4 - x_3x_4 \le 0$$

$$161.18996x_3x_4 + 5000x_2x_4 - 489510x_2 - x_3x_4x_7 \le 0$$

$$0.33x_7 - x_5 + 44.333333 \le 0$$

$$0.022556x_5 - 0.007595x_7 - 1 \le 0$$

$$0.00061x_3 - 0.0005x_1 - 1 \le 0$$

$$0.819672\ x_1 - x_3 + 0.819672 \le 0$$

$$24500\ x_2 - 250\ x_2x_4 - x_3x_4 \le 0$$

$$1020.4082\ x_4x_2 + 1.2244898\ x_3x_4 - 100000\ x_2 \le 0$$

$$6.25\ x_1x_6 + 6.25\ x_1 - 7.625\ x_3 - 100000 \le 0$$

$$1.22\ x_3 - x_6x_1 - x_1 + 1 \le 0$$

The maximum profit as reported in Adjiman et al. (1998) is: \$1772.77 per day, and the optimal variable values are $x1 = 1698.18$, $x2 = 53.66$, $x3 = 3031.3$, $x4 = 90.11$, $x5 = 95.0$, $x6 = 10.50$, $x7 = 153.53$.

8.3.7 Heat exchanger network design (HEND)

This problem addresses the design of a heat exchanger network as shown in Fig. 4. It has been taken from Floudas and Pardalos (1990). One cold stream must be heated from 100 F (37.78 ^{0}C) to 500 F (260 ^{0}C) using three hot streams with different inlet temperatures. The goal is to minimize the overall heat exchange area.

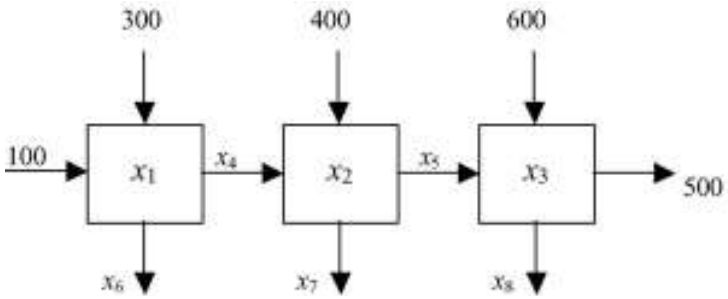


Figure 8.33 Flow sheet of HEND

$Min\ f = x_1 + x_2 + x_3$

Subjected to;

$$100\ x_1 - x_1\ (400 - x_4) + 833.33252\ x_4 - 83333.333 \le 0$$

$$x_2 x_4 - x_2\ (400 - x_5 + x_4) - 1250\ x_4 + 1250\ x_5 \le 0$$

$$x_3 x_5 - x_3\ (100 + x_5) - 2500\ x_5 + 1250000 \le 0$$

$$100 \le x_1 \le 10000$$

$$1000 \le x_2, x_3 \le 10000$$

$$10 \le x_4, x_5 \le 1000$$

The global optimum is:

$(x_1, x_2, x_3, x_4, x_5; f) = (579.19, 1360.13, 5109.92, 182.01, 295.60 ; 7049.25)$.

8.3.8 Optimization of Water Pumping System

A water pumping system [22] consists of two parallel pumps drawing water from a lower reservoir and delivering it to another that is 40 m higher, as shown in figure. In addition to overcoming the pressure difference due to the elevation, the friction in the pipe is $7.2w^2$ kPa, where w is the combined flow rate in kilograms per second. The pressure-flow-rate characteristics of the pumps are:

$$Pump1;\ \Delta p(kPa) = 810 - 25w_1 - 3.75w_1^2$$

$$Pump2;\ \Delta p(kPa) = 900 - 65w_2 - 30w_2^2$$

where w_1 and w_2 are the flow rates through pump 1 and pump 2, respectively.

The system can be represented by four simultaneous equations. The pressure difference due to elevation and friction is:

$$\Delta p = 7.2w^2 + \frac{(40\,m)(1000\,kg/m^3)(9.807\,m/s^2)}{(1000\,Pa/kPa)}$$

$$Pump1;\ \Delta p(kPa) = 810 - 25w_1 - 3.75w_1^2$$

$$Pump2;\ \Delta p(kPa) = 900 - 65w_2 - 30w_2^2$$

$$Mass\,balance;\ \ w = w_1 + w_2$$

The objective here is to minimize D*p* subject to the constraints. Hence,

$$Min\ \Delta p = 7.2w^2 + \frac{(40\,m)(1000\,kg/m^3)(9.807\,m/s^2)}{(1000\,Pa/kPa)}$$

Subjected to;

$$\Delta p(kPa) = 810 - 25w_1 - 3.75w_1^{\,2}$$

$$\Delta p(kPa) = 900 - 65w_2 - 30w_2^{\,2}$$

$$w = w_1 + w_2$$

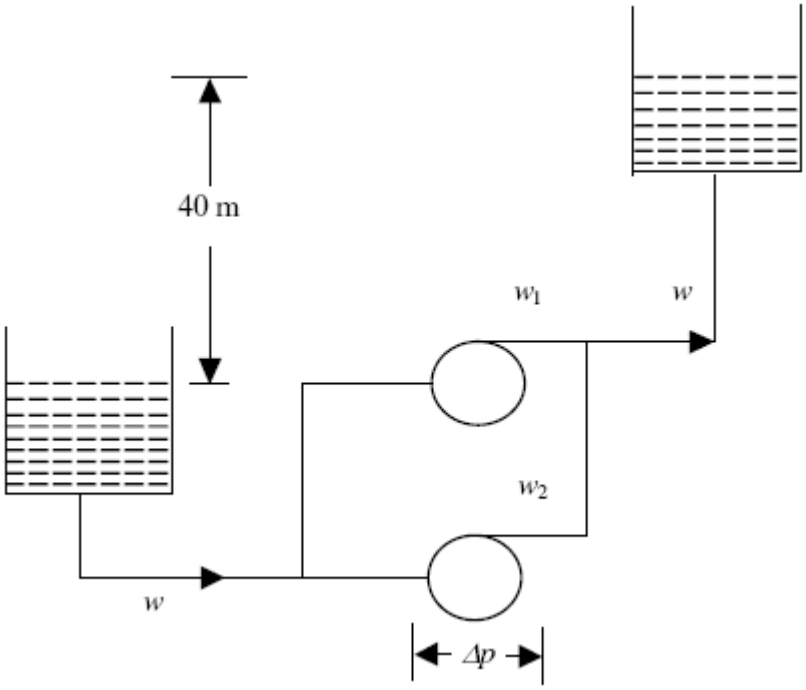


Figure 8.34 Flow sheet of water pumping system

In general, the equality constraints are difficult to deal with. So there is a need to transform equality constraints into inequality constraints by some means or the other. Typically, they are handled by either of the following two methods, viz., (1) eliminating the parameter and hence reducing the dimensions of the problem (2) an equality constraint is formulated into two inequalities by introducing deviation variables on problem parameter. In the present study, one variable is eliminated while the other two equalities are transformed into inequalities using method 1. Hence, the reformulated problem is as follows:

$$Min\ f = x_3 = 150 + 0.5(x_1 + x_2)^2$$

Subjected to;

$$6\ x_1^2 - 30\ x_1 - 249.9999999 + 150 + 0.5(x_1 + x_2)^2 \geq 0$$

$$12\ x_2^2 - 20\ x_2 - 249.9999999 + 150 + 0.5(x_1 + x_2)^2 \geq 0$$

$$0 \leq x_1 \leq 9.422$$

$$0 \leq x_2 \leq 5.903$$

The global optimum obtained is:

$(x_1, x_2; f) = (6.41608, 3.12184; 195.485979)$.

8.3.9 Reactor network design (RND)

This example, taken from Ryoo and Sahinidis (1995), is a reactor network design problem, describing the system shown in figure. It involves the design of a sequence of two CSTR reactors where the

consecutive reaction A→B→C takes place. The goal is to maximize the concentration of product B (x_4 =CB2) in the exit stream. This problem is known to have caused difficulties for other global optimization methods.

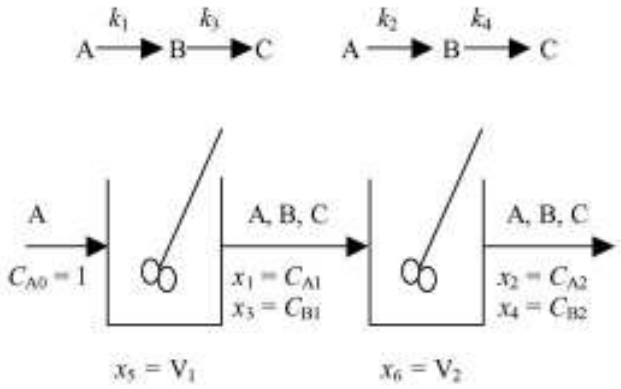


Figure 8.35 Flow sheet of reactor network design

This example constitutes a very difficult test problem, as it possesses a local minimum with an objective function value that is very close to that of the global solution. The local solutions are with f =−0.37461 and f =−0.38808. Interestingly enough, the two local solutions utilize only one of the two reactors whereas the global solution makes use of both reactors.

$k_1 = 0.09755988$

$k_2 = 0.99k_1$

$k_3 = 0.0391908$

$k_4 = 0.9k_3$

$Min\ f = -x_4$

Subjected to;

$$x_1 + k_1 x_1 x_5 = 1$$

$$x_2 - x_1 + k_2 x_2 x_6 = 0$$

$$x_3 + x_1 + k_3 x_3 x_5 = 1$$

$$x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0$$

$$x_5^{0.5} + x_6^{0.5} \le 4$$

$$(0,0,0,0,10^{-5},10^{-5}) \le (x_1, x_2, x_3, x_4, x_5, x_6,) \le (1,1,1,1,16,16)$$

The global optimum is ;

$(x_1, x_2, x_3, x_4, x_5, x_6; f) = (0.771462, 0.516997, 0.204234, 0.388812, 3.036504, 5.096052; -0.388812)$.

This problem can be reformulated by eliminating equality constraint as follows:

$$Max\ f = \frac{k_2 x_6 (1 + k_3) + k_1 (1 + k_2 x_6)}{(1 + k_1 x_5)(1 + k_2 x_6)(1 + k_3 x_5)(1 + k_4 x_6)}$$

Subjected to;

$$x_5^{0.5} + x_6^{0.5} \le 4$$

$$(10^{-5},10^{-5}) \le (x_5, x_6) \le (16,16)$$

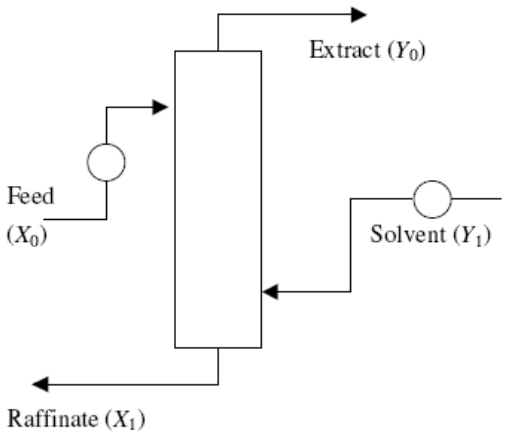
Global optimum is same after reformulation.

8.3.10 Optimization of Extraction Process

Assuming that the concentrations are expressed on a solute free mole basis and that the equilibrium relation between Y and X is a straight, i.e., the phases are insoluble. The model is then given as below:

$$\frac{dX}{dZ} - N_{ox}(X - Y) = 0$$

$$\frac{dY}{dZ} - FN_{ox}(X - Y) = 0$$



where,

F = extraction factor $\left(mv_x / v_y \right)$

m = distribution coefficient (m = 1.5)

NOX = number of transfer units

v_x, v_y = superficial velocity in raffinate, extract phase

X = dimensionless raffinate phase concentration

Y = dimensionless extract phase concentration

Z = dimensionless contactor length

Figure shows the extraction column with boundary conditions X_0 and Y_1. A solution for Y_0 in terms of v_x and v_Y can be obtained, given the values for m, N_{OX}, and the length of the column.

Hartland and Mecklenburgh list the solution for the plug flow model (and also the axial dispersion model) for a linear equilibrium relationship, in terms of F:

$$Y_0 = \frac{F\left[1-\exp\left(N_{ox}\left[1-F\right]\right)\right]}{1-F\exp\left(N_{ox}\left[1-F\right]\right)}$$

For the plug flow and axial diffusion models, Jackson and Agnew [1] summarized a number of correlations for NOX given by an equation of the form:

$$N_{ox} = a\left[\frac{v_x}{v_y}\right]^b (N)^c$$

The correlations obtained by non-linear least squares regression were:

$N = 8.33\ s^{-1}$

$$N_{ox} = 4.85\left[\frac{v_x}{v_y}\right]^{0.24}$$

Maximize the total extraction rate for constant disk rotation speed subject to the inequality constraints:

$$Max\ f = v_y Y_0$$

Inequality constraints;

$$v_x + v_y \leq 0.20$$

Constraints on v_X and v_Y would be upper and lower bounds such as:

$$0.05 \leq v_x \leq 0.25$$
$$0.05 \leq v_y \leq 0.30$$

The global optimum is; $(v_x, v_y; f) = (0.15, 0.05; 0.225)$

8.3 Non Linear Regression Analysis with D.E

Nonlinear regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables. The data are fitted by a method of successive approximations.

In this section include 15 regression analysis problems. Each one has different behavior. For that reason, the differential evolution algorithm is tested perfectly.

8.4.1 Problem 19

Fit the data below the model;

$$y = A_1 * e^{A_2 * x_1}$$

Table 8.2 Experimental data for Problem 19

Y	X_1
3.2939	1
4.2699	2
7.1749	4
9.3008	5
20.229	8

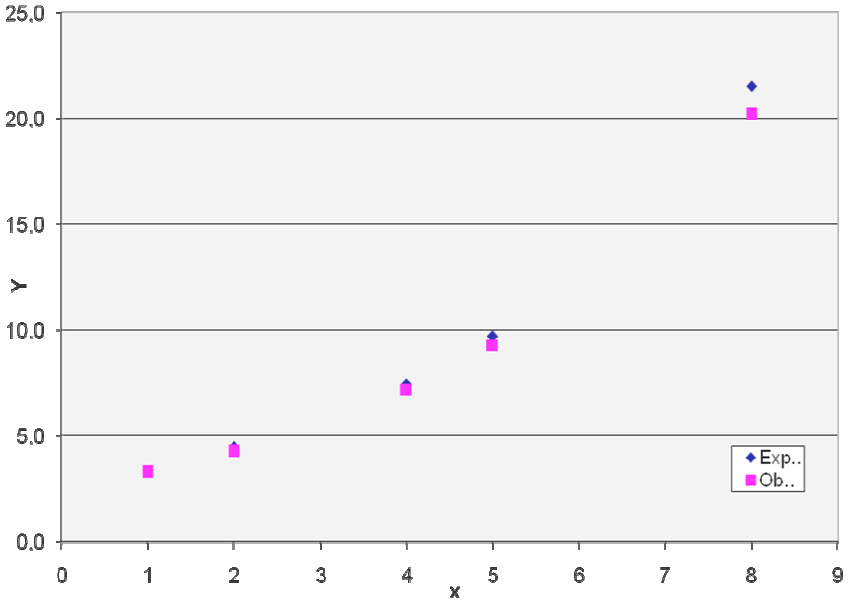


Figure 8.36 Expected v.s Observed graph for problem 19

Table 8.3 Result Table For problem 19

	Result
A_1	2.54
A_2	0.259
Min. Function Value	3.31 E-5

8.4.2 roblem 20

Fit the data below the model;

$y = A_1 * e^{A_2 * x_1}$

Table 8.4 Experimental data for Problem 20

Y	X_1
10	1
11	2
14.6	4
16.5	5
24	8
14.8	4.1

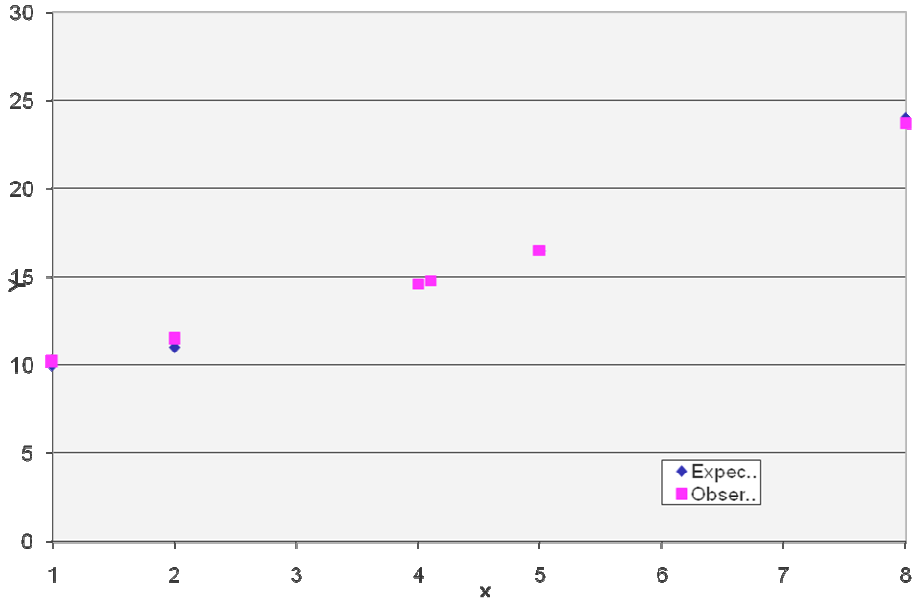


Figure 8.37 Expected v.s Observed graph for problem 20

Table 8.5 Result Table For problem 20

	Result
A_1	9.01
A_2	0.121
Min. Function Value	0.334

8.4.3 Problem 23

Fit the data below the model;

$$y = \frac{A_1 * x_1 * \log\left[\frac{A_2}{x_2}\right]}{e^{A_3 * x_3} + A_4}$$

Table 8.6 Experimental data for Problem 23

Y	**X_1**	**X_2**	**X_3**
0.81028	1	0.1	0.1
8.1028	10	0.1	0.1
12.124	15	0.1	0.1
5.0514	5	0.1	0.1
60.771	75	0.1	0.1
0.68833	1	0.1	1
6.8833	10	0.1	1
10.325	15	0.1	1
3.4417	5	0.1	1
51.625	75	0.1	1
0.30451	1	1	0.1
3.0451	10	1	0.1
4.5676	15	1	0.1
1.5225	5	1	0.1
22.838	75	1	0.1

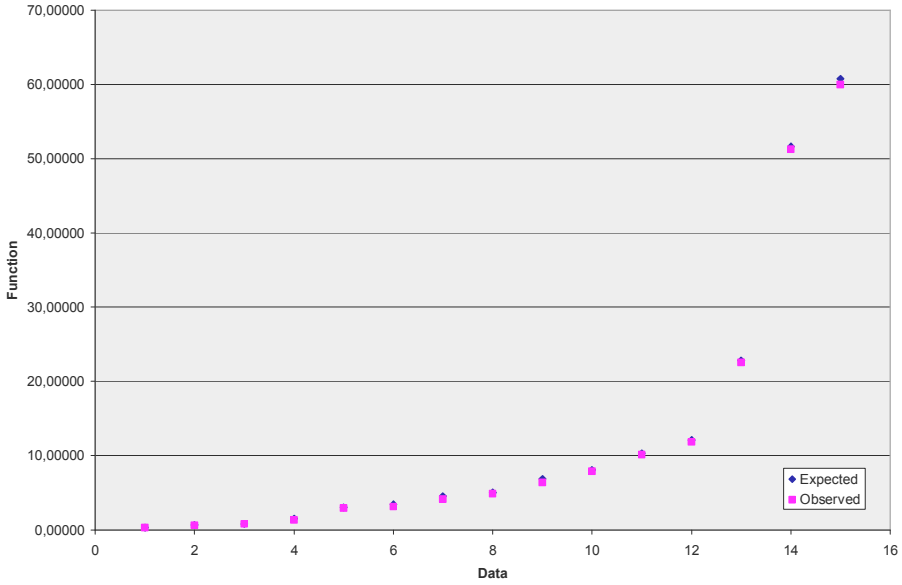


Figure 8.38 Expected v.s Observed graph for problem 23

Table 8.7 Result Table For problem 23

	Result
A_1	0.696
A_2	3.99
A_3	0.478
A_4	2.12
Min. Function Value	0.0997

8.4.4 Problem 24

Fit the data below the model;

$$y = e^{\left(-x_1/x2\right)} * \frac{A_1 + A_2 * x_1 + A_3 * {x_1}^2 + {x_1}^3}{A_4 + A_5 * x_1 + A_6 * {x_1}^2 + {x_1}^3}$$

Table 8.8 Experimental data for Problem 24

Y	X_1
1.9697	0.1
0.3867	0.67
0.226	1.34
0.04611	2.01
0.01877	2.68
0.015805	4.69
0.0072126	5.36
0.0033327	6.03
0.0015553	6.7
0.00073177	4.737
0.00034665	8.04
0.00016516	8.71
0.000079076	9.38
0.000038024	10.05

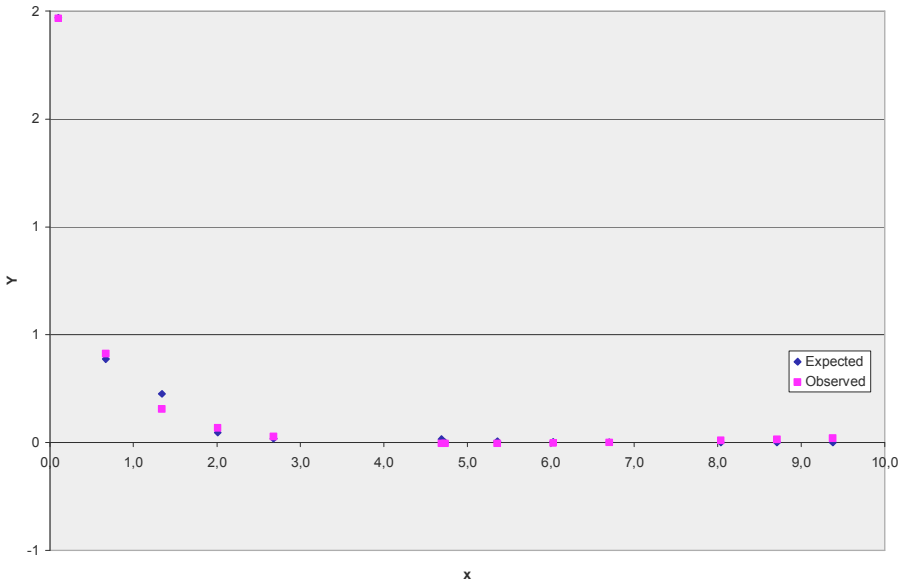


Figure 8.39 Expected v.s Observed graph for problem 24

Table 8.9 Result Table For problem 24

	Result
$\mathbf{A_1}$	8.06
$\mathbf{A_2}$	30
$\mathbf{A_3}$	-11.8
$\mathbf{A_4}$	-0.041
$\mathbf{A_5}$	18.9
$\mathbf{A_6}$	19
Min. Function Value	5.98 E-3

8.4.5 Chwirut2

These data are the result of a NIST study involving ultrasonic calibration. The response variable is ultrasonic response, and the predictor variable is metal distance.

Data: 1 Response Variable (***y***) (y = ultrasonic response)
1 Predictor Variable (***x***) (x = metal distance)
54 Observations

Model: Exponential Class
3 Parameters (β_1 to β_3)

$$y = \frac{\exp[-\beta_1 * x_1]}{\beta_2 + \beta_3 * x_1}$$

Table 8.12 Experimental data for Chwirut2

Data	Y	X	Data	Y	X
1	92.9	0.5	**28**	75.8	0.5
2	57.1	1	**29**	20	2
3	31.05	1.75	**30**	10.42	4
4	11.5875	3.75	**31**	59.5	0.75
5	8.025	5.75	**32**	21.67	2
6	63.6	0.875	**33**	8.55	5
7	21.4	2.25	**34**	62	0.75
8	14.25	3.25	**35**	20.2	2.25
9	8.475	5.25	**36**	7.76	3.75
10	63.8	0.75	**37**	3.75	5.75
11	26.8	1.75	**38**	11.81	3
12	16.46	2.75	**39**	54.7	0.75
13	7.125	4.75	**40**	23.7	2.5
14	67.3	0.625	**41**	11.55	4
15	41	1,25	**42**	61,3	0,75
16	21.15	2.25	**43**	17.7	2.5
17	8.175	4.25	**44**	8.74	4
18	81.5	0.5	**45**	59.2	0.75
19	13.12	3	**46**	16,3	2,5
20	59.9	0.75	**47**	8.62	4
21	14.62	3	**48**	81	0.5
22	32.9	1.5	**49**	4.87	6
23	5.44	6	**50**	14.62	3
24	12.56	3	**51**	81.7	0.5
25	5.44	6	**52**	17.17	2.75
26	32	1.5	**53**	81.3	0.5
27	13.95	3	**54**	28.9	1.75

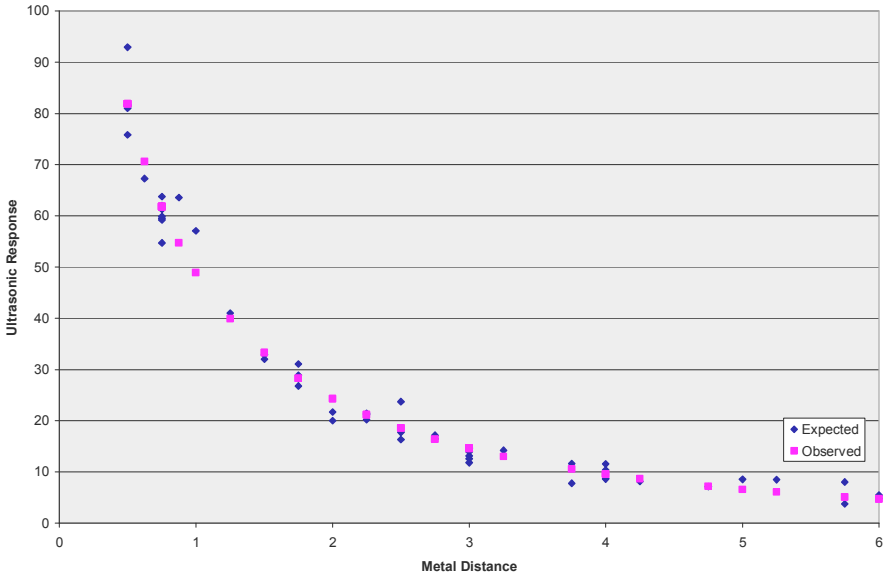


Figure 8.41 Expected v.s Observed graph for Chwirut2

Table 8.13 Result Table for Chwirut2

	Result
β_1	0.167
β_2	0.00516
β_3	0.0122
Min. Function Value	514

8.4.6 The Hougen-Watson Function:

The Hougen-Watson model (Bates and Watts, 1988) for reaction kinetics is a typical example of nonlinear regression model. The rate of kinetic reaction (y) is dependent on the quantities of three inputs: hydrogen (x1), n-pentane (x2) and isopentane (x3). The model is specified as:

$$y = rate = \frac{\beta_1 - \frac{x_3}{\beta_5}}{1 + \beta_2 * x_1 + \beta_3 * x_2 + \beta_4 * x_3}$$

For the given dataset the minimum the graphical presentation of the observed values against the expected values of y suggests that the model fits to the data very well.

Table 8.14 Experimental data for The Hougen-Watson Function

rate	**x(1)**	**x(2)**	**x(3)**
8.55	470	300	10
3.79	285	80	10
4.82	470	300	120
0.02	470	80	120
2.75	470	80	10
14.39	100	190	10
2.54	100	80	65
4.35	470	190	65
13.00	100	300	54
8.50	100	300	120
0.05	100	80	120
11.32	285	300	10
3.13	285	190	120

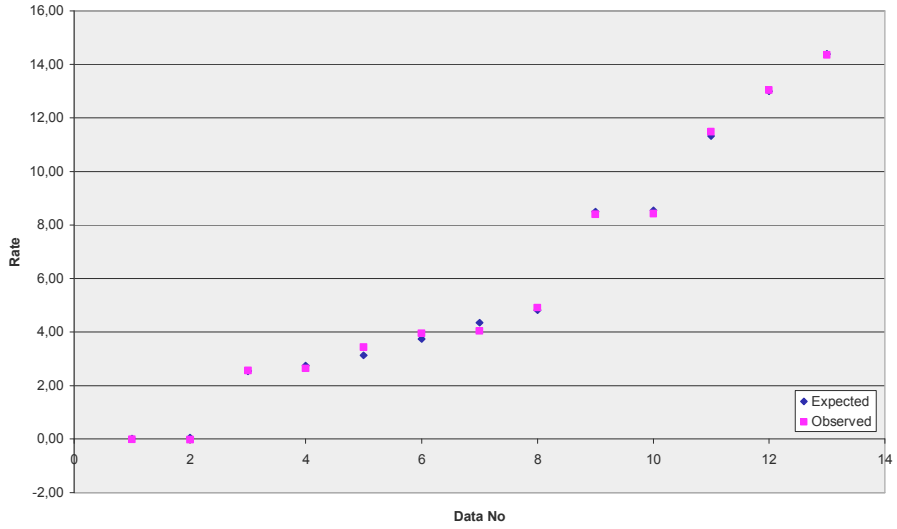


Figure 8.42 Expected v.s Observed graph for The Hougen-Watson Function

Table 8.15 Result Table For The Hougen-Watson Function

	Result
β_1	1.25258511
β_2	0.0627757706
β_3	0.0400477234
β_4	0.112414719
β_5	1.19137809
Min. Function Value	0.298900981

8.4.7 Lanczos Function:

Lanczos (1956) presented several data sets (at different accuracy levels) generated by an exponential function. Using the given dataset of this problem one may estimate the parameters of h (x) = b exp (-b x) + b exp (-b x) + b exp (-b x) + u and check if the values of (0.0951, 1, 0.8607, 3,

1.5576, 5) are obtained. The estimated parameters are very close to the true parameters.

$$y = \beta_1 * \exp[-\beta_2 * x] + \beta_3 * \exp[-\beta_4 * x] + \beta_5 * \exp[-\beta_6 * x]$$

Table 8.16 Experimental data for Lanczos Function

Data	Y	X
1	2.5134	0
2	2.0443	0.05
3	1.6684	0.1
4	1.3664	0.15
5	1.1232	0.2
6	0.9269	0.25
7	0.7679	0.3
8	0.6389	0.35
9	0.5338	0.4
10	0.4479	0.45
11	0.3776	0.5
12	0.3197	0.55
13	0.2720	0.6
14	0.2325	0.65
15	0.1997	0.7
16	0.1723	0.75
17	0.1493	0.8
18	0.1301	0.85
19	0.1138	0.9
20	0.1000	0.95
21	0.0883	1
22	0.0783	1.05
23	0.0698	1.1
24	0.0624	1.15

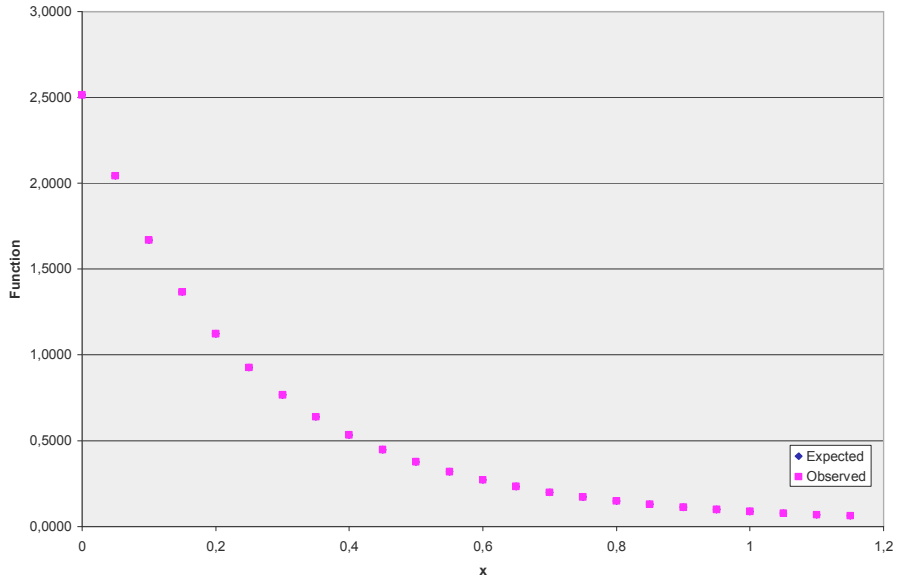


Figure 8.43 Expected v.s Observed graph for Lanczos Function

Table 8.17 Result Table For Lanczos Function

	Result
β_1	1.55
β_2	3.23
β_3	0.936
β_4	5.95
β_5	0.027
β_6	0.121
Min. Function Value	8.756 E-4

8.4.8 The Kirby Function:

Kirby (NIST, 1979) measured response values (y) against input values (x) to scanning electron microscope line width standards. The Kirby function is the ratio of two quadratic polynomials.

$$y = \frac{\beta_1 + \beta_2 * x + \beta_3 * x^2}{1 + \beta_4 * x + \beta_5 * x^2}$$

Table 8.18 Experimental data for The Kirby Function

Data	Y	X	Data	Y	X	Data	Y	X
1	0.0082	9.65	**52**	36.17	160.56	**102**	68.77	256.24
2	0.0112	10.74	**53**	36.84	162.30	**103**	69.59	259.11
3	0.0149	11.81	**54**	38.01	165.21	**104**	70.11	261.18
4	0.0198	12.88	**55**	38.67	166.90	**105**	70.86	264.02
5	0.0248	14.06	**56**	39.87	169.92	**106**	71.43	266.13
6	0.0324	15.28	**57**	40.03	170.32	**107**	72.16	268.94
7	0.0420	16.63	**58**	40.50	171.54	**108**	72.70	271.09
8	0.0549	18.19	**59**	41.37	173.79	**109**	73.40	273.87
9	0.0719	19.88	**60**	41.67	174.57	**110**	73.93	276.08
10	0.0963	21.84	**61**	42.31	176.25	**111**	74.60	278.83
11	0.1291	24.00	**62**	42.73	177.34	**112**	75.16	281.08
12	0.1710	26.25	**63**	43.46	179.19	**113**	75.82	283.81
13	0.2314	28.86	**64**	44.14	181.02	**114**	76.34	286.11
14	0.3227	31.85	**65**	44.55	182.08	**115**	76.98	288.81
15	0.4809	35.79	**66**	45.22	183.88	**116**	77.48	291.08
16	0.7084	40.18	**67**	45.92	185.75	**117**	78.08	293.75
17	1.022	44.74	**68**	46.30	186.80	**118**	78.60	295.99
18	1.458	49.53	**69**	47.00	188.63	**119**	79.17	298.64
19	1.952	53.94	**70**	47.68	190.45	**120**	79.62	300.84
20	2.541	58.29	**71**	48.06	191.48	**121**	79.88	302.02
21	3.223	62.63	**72**	48.74	193.35	**122**	80.19	303.48
22	3.999	67.03	**73**	49.41	195.22	**123**	80.66	305.65
23	4.852	71.25	**74**	49.76	196.23	**124**	81.22	308.27
24	5.732	75.22	**75**	50.43	198.05	**125**	81.66	310.41
25	6.727	79.33	**76**	51.11	199.97	**126**	82.16	313.01
26	7.835	83.56	**77**	51.50	201.06	**127**	82.59	315.12
27	9.025	87.75	**78**	52.12	202.83	**128**	83.14	317.71
28	10.267	91.93	**79**	52.76	204.69	**129**	83.50	319.79
29	11.578	96.10	**80**	53.18	205.86	**130**	84.00	322.36
30	12.944	100.28	**81**	53.78	207.58	**131**	84.40	324.42
31	14.377	104.46	**82**	54.46	209.50	**132**	84.89	326.98
32	15.856	108.66	**83**	54.83	210.65	**133**	85.26	329.01

33	17.331	112.71	**84**	55.40	212.33	**134**	85.74	331.56
34	18.885	116.88	**85**	56.43	215.43	**135**	86.07	333.56
35	20.575	121.33	**86**	57.03	217.16	**136**	86.54	336.10
36	22.320	125.79	**87**	58.00	220.21	**137**	86.89	338.08
37	22.303	125.79	**88**	58.61	221.98	**138**	87.32	340.60
38	23.460	128.74	**89**	59.58	225.06	**139**	87.65	342.57
39	24.060	130.27	**90**	60.11	226.79	**140**	88.10	345.08
40	25.272	133.33	**91**	61.10	229.92	**141**	88.43	347.02
41	25.853	134.79	**92**	61.65	231.69	**142**	88.83	349.52
42	27.110	137.93	**93**	62.59	234.77	**143**	89.12	351.44
43	27.658	139.33	**94**	63.12	236.60	**144**	89.54	353.93
44	28.924	142.46	**95**	64.03	239.63	**145**	89.85	355.83
45	29.511	143.90	**96**	64.62	241.50	**146**	90.25	358.32
46	30.71	146.91	**97**	65.49	244.48	**147**	90.55	360.20
47	31.35	148.51	**98**	66.03	246.40	**148**	90.93	362.67
48	32.52	151.41	**99**	66.89	249.35	**149**	91.20	364.53
49	33.23	153.17	**100**	67.42	251.32	**150**	91.55	367.00
50	34.33	155.97	**101**	68.23	254.22	**151**	92.20	371.30

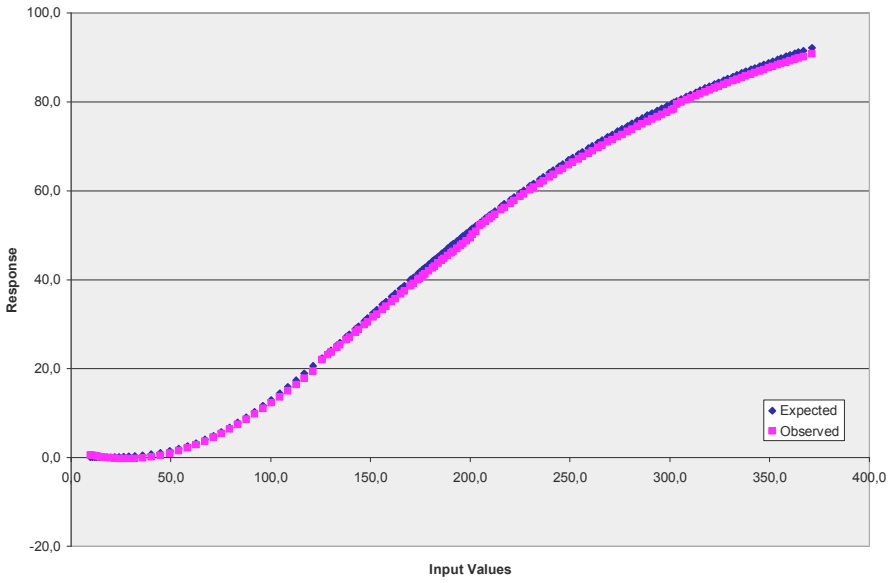


Figure 8.44 Expected v.s Observed graph for The Kirby Function

Table 8.19 Result Table for the Kirby Function

	Result
β_1	1.67450632
β_2	-0.13927398
β_3	0.00259611813
β_4	-0.00172418116
β_5	2.16648026E-5
Min. Function Value	3.90507396

8.4.9 The ENSO Function:

This function (Kahaner. et al.. 1989) relates y. monthly averaged atmospheric pressure differences between Easter Island and Darwin. Australia and time (x). The difference in the atmospheric pressure (y) drives the trade winds in the southern hemisphere (NIST. USA). The function is specified as

$$y = \beta_1 + \beta_2 * \cos\left[2 * \pi * x / 12\right] + \beta_3 * \sin\left[2 * \pi * x / 12\right] + \beta_5 * \cos\left[2 * \pi * x / \beta_4\right] +$$
$$\beta_6 * \sin\left[2 * \pi * x / \beta_4\right] + \beta_8 * \cos\left[2 * \pi * x / \beta_7\right] + \beta_9 * \cos\left[2 * \pi * x / \beta_7\right]$$

Arguments to the sin(.) and cos(.) functions are in radians

Table 8.20 Experimental data for The Enso Function

Data	**Y**	**X**	**Data**	**Y**	**x**	**Data**	**Y**	**x**
1	12.9	1	**57**	10.9	57	**113**	14.3	113
2	11.3	2	**58**	11.7	58	**114**	14.5	114
3	10.6	3	**59**	11.4	59	**115**	8.5	115
4	11.2	4	**60**	13.7	60	**116**	12	116
5	10.9	5	**61**	14.1	61	**117**	12.7	117
6	7.5	6	**62**	14	62	**118**	11.3	118

7	7.7	7	**63**	12.5	63	**119**	14.5	119
8	11.7	8	**64**	6.3	64	**120**	15.1	120
9	12.9	9	**65**	9.6	65	**121**	10.4	121
10	14.3	10	**66**	11.7	66	**122**	11.5	122
11	10.9	11	**67**	5	67	**123**	13.4	123
12	13.7	12	**68**	10.8	68	**124**	7.5	124
13	17.1	13	**69**	12.7	69	**125**	0.6	125
14	14	14	**70**	10.8	70	**126**	0.3	126
15	15.3	15	**71**	11.8	71	**127**	5.5	127
16	8.5	16	**72**	12.6	72	**128**	5	128
17	5.7	17	**73**	15.7	73	**129**	4.6	129
18	5.5	18	**74**	12.6	74	**130**	8.2	130
19	7.6	19	**75**	14.8	75	**131**	9.9	131
20	8.6	20	**76**	7.8	76	**132**	9.2	132
21	7.3	21	**77**	7.1	77	**133**	12.5	133
22	7.6	22	**78**	11.2	78	**134**	10.9	134
23	12.7	23	**79**	8.1	79	**135**	9.9	135
24	11	24	**80**	6.4	80	**136**	8.9	136
25	12.7	25	**81**	5.2	81	**137**	7.6	137
26	12.9	26	**82**	12	82	**138**	9.5	138
27	13	27	**83**	10.2	83	**139**	8.4	139
28	10.9	28	**84**	12.7	84	**140**	10.7	140
29	10.4	29	**85**	10.2	85	**141**	13.6	141
30	10.2	30	**86**	14.7	86	**142**	13.7	142
31	8	31	**87**	12.2	87	**143**	13.7	143
32	10.9	32	**88**	7.1	88	**144**	16.5	144
33	13.6	33	**89**	5.7	89	**145**	16.8	145
34	10.5	34	**90**	6.7	90	**146**	17.1	146
35	9.2	35	**91**	3.9	91	**147**	15.4	147
36	12.4	36	**92**	8.5	92	**148**	9.5	148
37	12.7	37	**93**	8.3	93	**149**	6.1	149
38	13.3	38	**94**	10.8	94	**150**	10.1	150
39	10.1	39	**95**	16.7	95	**151**	9.3	151
40	7.8	40	**96**	12.6	96	**152**	5.3	152
41	4.8	41	**97**	12.5	97	**153**	11.2	153
42	3	42	**98**	12.5	98	**154**	16.6	154
43	2.5	43	**99**	9.8	99	**155**	15.6	155
44	6.3	44	**100**	7.2	100	**156**	12	156

45	9.7	45	**101**	4.1	101	**157**	11.5	157
46	11.6	46	**102**	10.6	102	**158**	8.6	158
47	8.6	47	**103**	10.1	103	**159**	13.8	159
48	12.4	48	**104**	10.1	104	**160**	8.7	160
49	10.5	49	**105**	11.9	105	**161**	8.6	161
50	13.3	50	**106**	13.6	106	**162**	8.6	162
51	10.4	51	**107**	16.3	107	**163**	8.7	163
52	8.1	52	**108**	17.6	108	**164**	12.8	164
53	3.7	53	**109**	15.5	109	**165**	13.2	165
54	10.7	54	**110**	16	110	**166**	14	166
55	5.1	55	**111**	15.2	111	**167**	13.4	167
56	10.4	56	**112**	11.2	112	**168**	14.8	168

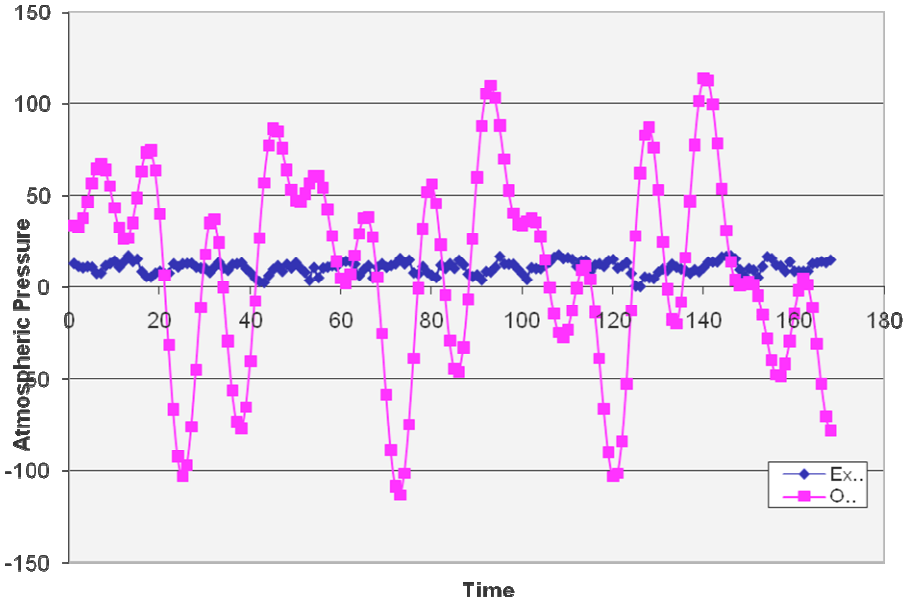


Figure 8.45 Expected v.s Observed graph for The Enso Function

Table 8.21 Result Table for the Enso Function

	Result
β_1	10.5107492
β_2	3.0762128
β_3	0.532801425
β_4	26.8876144
β_5	0.212322867
B_6	1.49668704
β_7	44.3110885
β_8	-1.62314288
β_9	0.525544858
Min. Function Value	1.31 E+6

8.4.10 Roszman Function

These data are the result of a NIST study involving quantum defects in iodine atoms. The response variable is the number of quantum defects. and the predictor variable is the excited energy state. The argument to the ARCTAN function is in radians

Data: 1 Response (y = quantum defect)

1 Predictor (x = excited state energy)

25 Observations

$$y = \beta_1 - \beta_2 x - \frac{\arctan\left[\frac{\beta_3}{x - \beta_4}\right]}{\pi}$$

Table 8.22 Experimental data for The Roszman Function

Data	y	x
1	0.252429	-4868.68
2	0.252141	-4868.09
3	0.251809	-4867.41
4	0.297989	-3375.19
5	0.296257	-3373.14
6	0.295319	-3372.03
7	0.339603	-2473.74
8	0.337731	-2472.35
9	0.33382	-2469.45
10	0.38951	-1894.65
11	0.386998	-1893.4
12	0.438864	-1497.24
13	0.434887	-1495.85
14	0.427893	-1493.41
15	0.471568	-1208.68
16	0.461699	-1206.18
17	0.461144	-1206.04
18	0.513532	-997.92
19	0.506641	-996.61

20	0.505062	-996.31
21	0.535648	-834.94
22	0.533726	-834.66
23	0.568064	-710.03
24	0.612886	-530.16
25	0.624169	-464.17

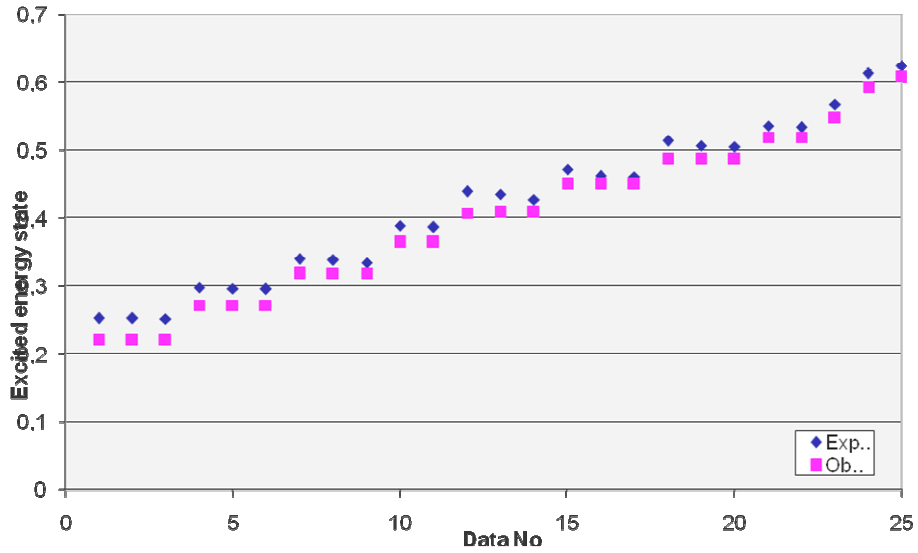


Figure 8.46 Expected v.s Observed graph for Roszman Function

Table 8.23 Result Table for the Roszman Function

	Result
β_1	0.202
β_2	-0.0000061
β_3	1180
β_4	-148
Min. Function Value	0.615

8.4.11 Ratkowsky

This model and data are an example of fitting sigmoidal growth curves taken from Ratkowsky (1983). The response variable is the dry weight of onion bulbs and tops. and the predictor variable is growing time.

Data: 1 Response (y = onion bulb dry weight)
1 Predictor (x = growing time)
15 Observations

$$y = \frac{\beta_1}{1 + \exp\left[\beta_2 - \beta_3 x\right]}$$

Table 8.24 Experimental data for The Ratkowsky Function

Data	Y	X
1	16.08	1
2	33.83	2
3	65.8	3
4	97.2	4
5	191.55	5
6	326.2	6
7	376.87	7
8	520.53	8
9	590.03	9
10	651.92	10
11	724.93	11
12	699.56	12
13	689.96	13
14	637.56	14
15	717.4	15

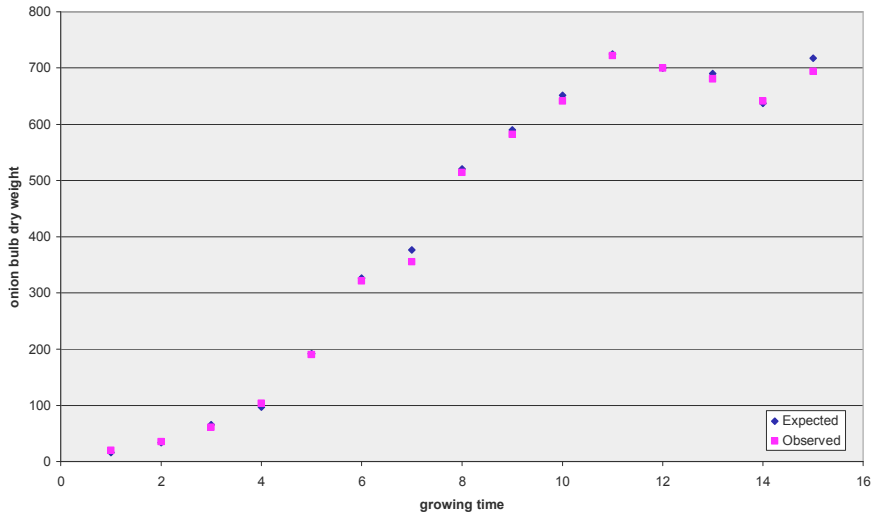


Figure 8.47 Expected v.s Observed graph for Ratkowsky Function

Table 8.25 Result Table for the Ratkowsky Function

	Result
β_1	72.5
β_2	2.62
β_3	0.0674
Min. Function Value	0.806

8.4.12 BoxBOD Function

These data are described in detail in Box. Hunter and Hunter (1978). The response variable is biochemical oxygen demand (BOD) in mg/l. and the predictor variable is incubation time in days.

Data: 1 Response (y = biochemical oxygen demand)
1 Predictor (x = incubation time)
6 Observations

$$y = \beta_1 * (1 - \exp[-\beta_2 * x])$$

Table 8.26 Experimental data for Box BOD Function

Y	X
109	1
149	2
169	3
191	5
213	7
224	10

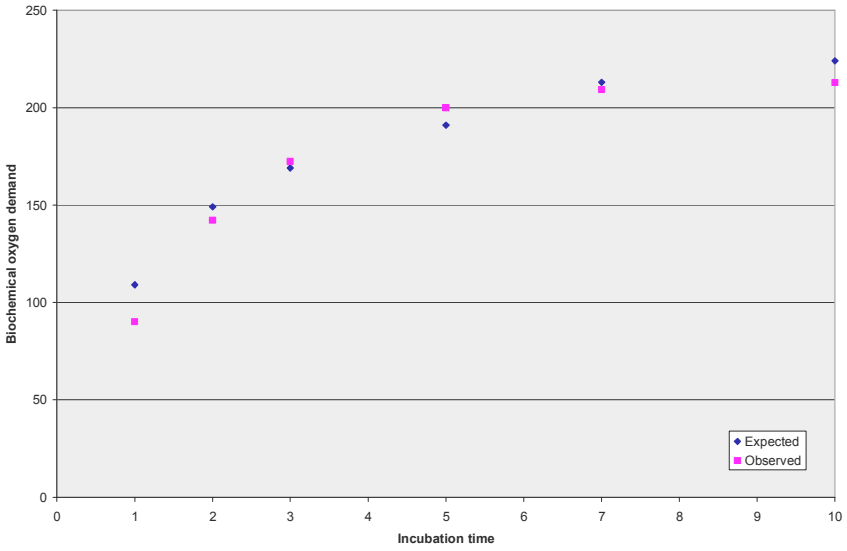


Figure 8.48 Expected v.s Observed graph for Box BOD Function

Table 8.27 Result Table for the Box BOD Function

	Result
β_1	213.8094
β_2	0.547237
Min. Function Value	1168.0085

8.4.13 Eckerle Function

These data are the result of a NIST study involving circular interference transmittance. The response variable is transmittance. and the predictor variable is wavelength.

Data: 1 Response Variable (y = transmittance)
1 Predictor Variable (x = wavelength)
35 Observations

$$y = \frac{\beta_1}{\beta_2}\exp\left[\frac{-(x-\beta_3)^2}{2*\beta_2{}^2}\right]$$

Table 8.28 Experimental data for Eckerle Function

Data	X	Y	Data	X	Y	Data	X	Y
1	400	0.000158	**13**	442	0.040168	**25**	460	0.03372
2	405	0.00017	**14**	444	0.071256	**26**	462	0.019402
3	410	0.000235	**15**	445	0.126446	**27**	463	0.011783
4	415	0.00031	**16**	447	0.207341	**28**	465	0.007436
5	420	0.000492	**17**	448	0.290237	**29**	470	0.002273
6	425	0.000871	**18**	450	0.344562	**30**	475	0.00088
7	430	0.001742	**19**	451	0.369805	**31**	480	0.000458
8	435	0.00464	**20**	453	0.366853	**32**	485	0.000235
9	436	0.00659	**21**	454	0.310673	**33**	490	0.000159
10	438	0.00973	**22**	456	0.207815	**34**	495	0.000114
11	439	0.0149	**23**	457	0.116435	**35**	500	0.000071
12	441	0.023731	**24**	459	0.061676			

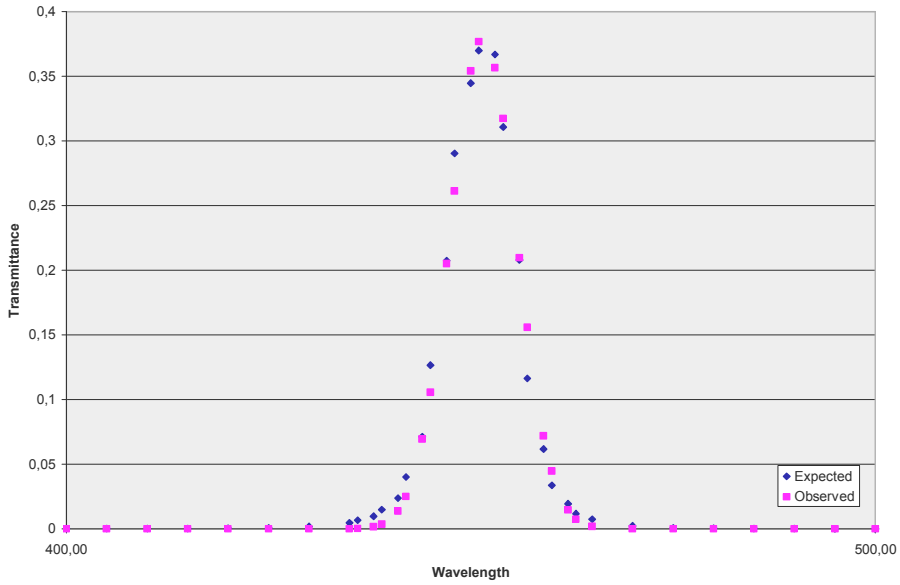


Figure 8.49 Expected v.s Observed graph for Eckerle Function

Table 8.29 Result Table for the Eckerle Function

	Result
β_1	1.554382
β_2	4.088832
β_3	451.5412
Min. Function Value	1.46358E-3

8.4.15 Misra Function

These data are the result of a NIST study regarding dental research in monomolecular adsorption. The response variable is volume. and the predictor variable is pressure.

Data: 1 Response (y = volume)
1 Predictor (x = pressure)
14 Observations

$$y = \beta_1 * \left[\frac{1}{\sqrt{1 + 2 * \beta_2 * x}} \right]$$

Table 8.30 Experimental data for Misra Function

Data	X	Y
1	77.6	10.07
2	114.90	14.73
3	141.10	17.94
4	190.80	23.93
5	239.90	29.61
6	289.00	35.18
7	332.80	40.02
8	378.40	44.82
9	434.80	50.76
10	477.30	55.05
11	536.80	61.01
12	593.10	66.40
13	689.10	75.47
14	760.00	81.78

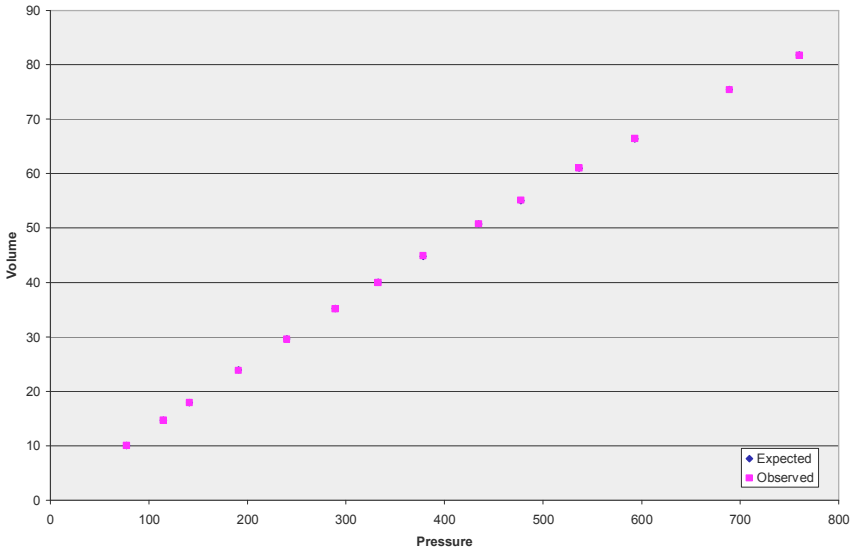


Figure 8.50 Expected v.s Observed graph for Misra Function

Table 8.31 Result Table for the Misra Function

	Result
β_1	6.364272 E-2
β_2	2.0813627 E-4
Min. Function Value	4.096683 E-2

9.0 CONCLUSIONS

Many engineering optimization problems contain multiple optimal solutions, among which one or more may be the absolute minimum or maximum solutions. These absolute optimum solutions are known as global optimal solutions and other optimum solutions are known as local optimal solutions. Ideally, we are interested in the global optimal solutions because they correspond to the absolute optimum objective function value. Most of the traditional optimization algorithms based on gradient methods have the possibility of getting trapped at local optimum depending upon the degree of non-linearity and initial guess. Unfortunately, none of the traditional algorithms are guaranteed to find the global optimal solution, but population based search algorithms are found to have a better global perspective than the traditional methods.

In the recent past, non-traditional search and optimization techniques based on natural phenomena (evolutionary computation) such as genetic algorithms (GA), evolution strategies (ESs), simulated annealing (SA), and differential evolution (DE) to name a few, have been developed to overcome the problems. Among their advantages are:

(1) They do not require the objective function to be continuous and/or differentiable,

(2) They do not require extensive problem formulation (in case of traditional methods such as Integer programming, geometric programming, branch and bound methods, etc., special mathematical formulation is required for solving a problem),

(3) They are not sensitive to starting point,
(4) They usually do not get stuck into so called local optima,
(5) They are more likely to find out a function's true global optimum.

These advantages enhance their application to various fields. They have been successfully applied in many engineering design problems.

This thesis presented an investigation of the "Random Search Algorithm" types. The target study was on Differential Evolution and its applications on optimization of nonlinear chemical processes. The first part of this study introduced basics of Random Search Algorithms and first impress of Differential Evolution in general.

The second section was thought of as a study of population based algorithms. Since the main differences between the traditional and population based algorithm is clearly progressiveness, and it is even not worth to compare.

Initialization, the first main part of the Differential Evolution algorithm, was also introduced in this section. Initialization likes a cloud which includes a possible solution. Generally, the term "population generation" was used to explain this phenomenon. This population was generated randomly, hence all generated possible solution points in the population differ from each other. And also, in this chapter it was worked on the effects of far initializing Differential Evolution with a uniformly random population. For this purpose, the population generator was modified by multiplying the generating equation,

($x_{j,i,0} = b_{j,L} + h * rand_j(0,1) * (b_{j,U} - b_{j,L})$) by an "h" factor. The "h" value was 1, 0.1, and 0.001 respectively. When "h" value was 1 the cloud is considered normal. On the other hand when the "h" value was 0.1 or 0.001 the cloud shrank. That means that far initialization occured. Table 2.1 reports the average number of function evaluations taken to find a point whose objective function value differs from the optimum objective function value by less than a preset minimum. Finding such a point within the maximum allowed number of generations constitutes a success; otherwise, the trial was considered to be a failure. Only "successes" are given in the results of Table 2.1. The fraction of successful trials, P, records the impact of failures. Results are 10-trial averages obtained using classic Differential Evolution with F = *Cr* = 0.9 and $r0 \neq r1 \neq r2 \neq i$ (distinct indices).

As Table 2.1 shows, far initialization's effect on the sphere, ridge, Rosenbrock, Chebyshev, Michalewicz and Griewangk functions was minimal. In most cases, far initialization penalizes these six functions with a very slight increase in the average number of function evaluations and a very slight decrease in the estimated probability of success. For both the sphere and ridge functions, this result was not surprising. Both functions were unimodal and convex, so neither poses obstacles to the population's expansion toward the minimum. Rosenbrock's function was also uni-modal, but unlike the sphere it is nonconvex. At least in the case of Rosenbrock's function, nonconvexity did not impede Differential Evolution's ability to locate the minimum when far initialized.

Unlike the sphere, ridge or Rosenbrock functions, the remaining functions in Table 2.1 are all multimodal. Table 2.1 shows that except for a slight increase in the number of function evaluations, diminishing the value of *h* did not significantly impact Differential Evolution's ability to converge on the Chebyshev optimum. Similarly, far initialization did not significantly affect Differential Evolution's performance on either Michalewicz or Griewangk function. Differential Evolution became unreliable, however, when far initializing Langerman function and failed altogether on the Ackley, Rastrigin and Schwefel functions once $h = 0.01$. For these highly multimodal functions, the entire initial population can land inside a single, nonoptimal, local basin of attraction when *h* becomes too small. If competing basins are sufficiently far apart, then classic Differential Evolution cannot generate difference vectors large enough to escape the local basin. Thus, it was important to use a bounding box of sufficient size when initializing multimodal functions with a uniform random distribution.

Also, this section included the other main part of the algorithm, namely, mutation, crossover and selection respectively.

Mutation and crossover are the heart of the algorithm, because of generating possible solution vectors. For a clear understanding, this procedure may be explained in a different perspective. To create a population including different members with different qualities, some of them were selected in random and mutated. Hence, two populations were created. Each population has a corresponding mutated population with the same name. Then the genes of the first population member and the

mutated one's were crossed and a new member was obtained. Then selection comes, and the two members having the same name were compared based on some criteria and the better one was selected as the new population.

In the fifth section of this thesis included another crucial point of the Differential Evolution Algorithm, mutation scale factor (F) and crossover constant (Cr). The performance of DE and MDE algorithms depends on key parameters, namely, CR, *F*, and NP were investigated. To study the effect of key parameters, two highly multimodal multidimensional test functions were used which were Ackley's (Equation 4.1) and Rastrigin's (Equation 4.2) functions, respectively. Also, a basic key parameter set (CR = 0.85, *F* = 0.9, and NP = 30) was first chosen. Then, all the numerical simulations were carried out around this basic set taking maximum number of generation to be 250. A systematic numerical simulation was performed keeping two key parameters constant while varying the third at a time.

Figures 4.1–4.4 show the effect of CR, *F*, and NP, respectively, on the performance of MDE and DE using the two functions as mentioned above. Each point on the figures represents the average of 10 independent experiments.

It was evident from these figures that both DE and MDE are affected in a similar way, i.e., the variation of function value with CR/*F*/NP is same qualitatively. It was important to note that variation of function value with CR, *F*, and NP is different for the same problem but

it is same for DE and MDE. Also, the variation of function value with CR and *F* was problem dependent, i.e., different problems may have different trends of function value vs. CR/*F*/NP. Therefore, for comparison purposes, it is essential to keep the same setting of key parameters for both DE and MDE although this setting can be different for different problems.

It was also aimed to improve this study by studying on modification in the mutation part. In the fifth and sixth section of this thesis, these strategies were investigated and tested. To study the effect of these modifications, four chemical engineering process optimization problems were used. These were reactor network design problem, process flow sheeting problem, and heat exchanger network design. Basic key parameter set (CR = 0.85, *F* = 0.9, and NP = 10*D) was chosen. Then, all the numerical simulations were carried out around this basic set taking maximum number of generation to be 1000.

Reactor network design problem (Figure 9.1) involves the design of a sequence of two CSTR reactors where the consecutive reaction A→B→C takes place. The goal is to maximize the concentration of product B (x_4) in the exit stream.

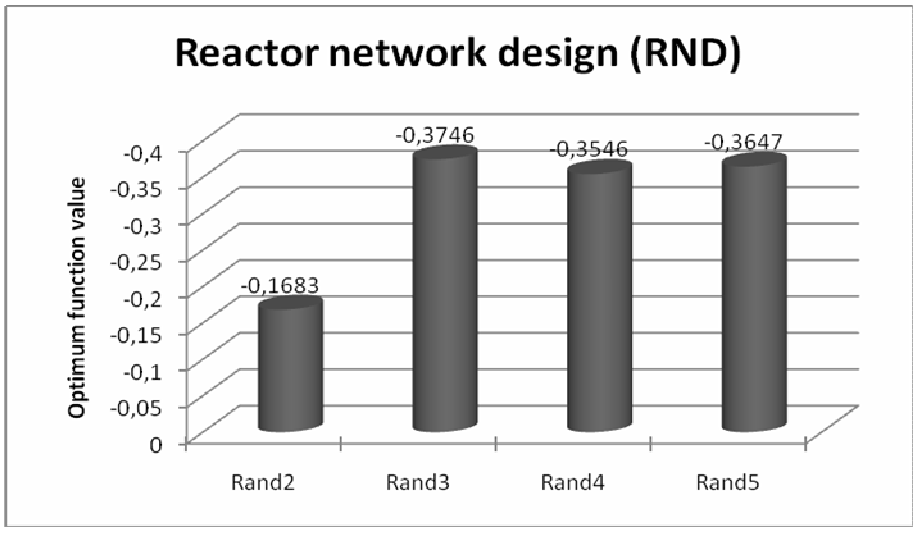


Figure 9.1 Comparison of the modifications in RND Problem.

Process flowsheeting problem (Figure 9.2) was non-convex because of the first constraint. And also this problem had binary variables. The goal was to minimize the objective function value.

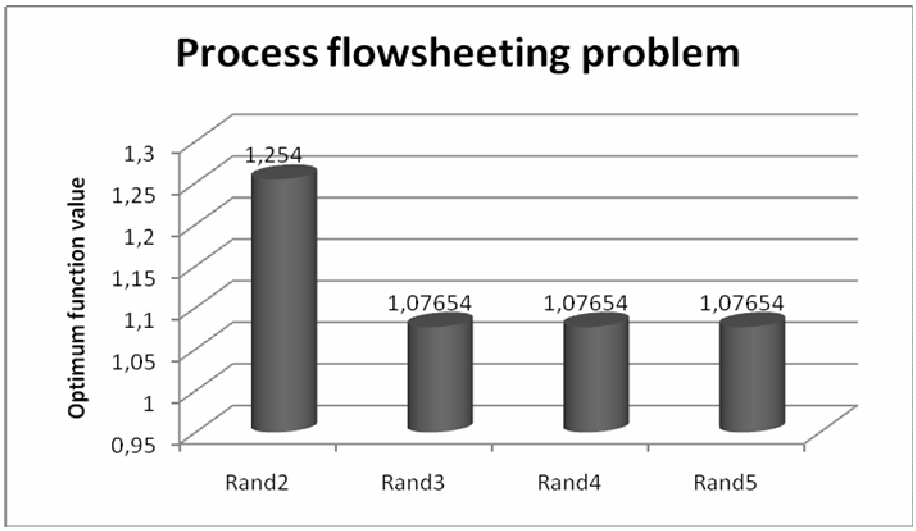


Figure 9.2 Comparison of the modifications in Process Flow sheeting Problem.

This problem addresses the design of a heat exchanger network (Figure 9.3) as shown in Figure 8.33. One cold stream must be heated from 37.78^0C to 260^0C using three hot streams with different inlet temperatures. The goal was to minimize the overall heat exchange area.

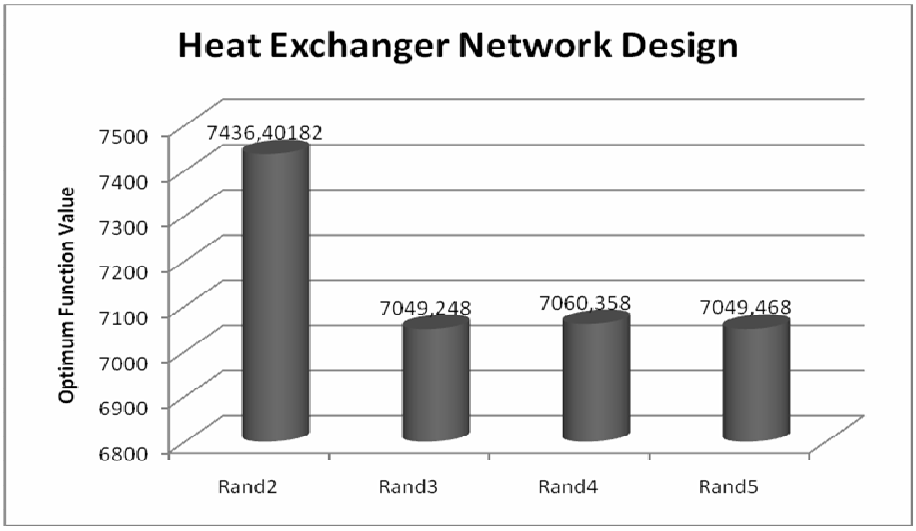


Figure 9.3 Comparison of the modifications in H.E Network Problem

Figures 9.1–9.3 show the effect of modifications on the mutation part of the Differential Evolution Algorithm using the three functions as mentioned above. Each bar on the figures represents the arithmetic average of 10 independent experiments. After this working, firstly as can be seen that except the random3 modification the all remaining gave unstable result, especially random2 modification. For the random2 modification, this result was not surprising, because of arithmetic recombination (crossover). Hence this easily eliminated from the others.

When worked on process flowsheeting problem, as could be seen that, random 3, 4, and 5 modification algorithms gave exactly equal

results. That was also not surprising. Because, the model of this problem had only one nonlinear constraint, hence it was not a complex model.

The figures easily show that the best response for these problems were with random3 modification. Consequently, random3 algorithm was used for all the investigation in this study.

Table 9.1 Effects of the penalty constant, P, on Chem. Eng. Problems.

	Penalty Constant				
	10	**100**	**1000**	**10000**	**100000**
Process synthesis prob.-1	3.256	2.000	2.000	2.000	2.000
Process synthesis and design problem	4.752	2.129	2.124	2.124	2.124
Process flow sheeting problem	10.356	6.548	2.541	1.077	1.077
Two-reactor problem(10^2)	12.063	8.544	4.003	1.000	0.992
Process synthesis prob.-2	18.364	12.332	5.896	3.601	3.557
Optimal operation of alkylation unit(10^2)	1.026	8.454	12.025	16.874	17.728
Heat exchanger network design(10^3)	0.456	0.979	1.450	7.001	7.049
Optimization Of Water Pumping System(10)	2.058	5.990	14.539	19.125	19.549
Reactor network design	1.256	0.889	-0.184	-0.367	-0.388
Optimization Of Extraction Process	0.625	0.428	0.225	0.225	0.225

Most optimization problems have constraints. The solution or set of solutions which are obtained as the final result of an evolutionary search must necessarily be feasible, that is, it should satisfy all the

constraints. For this reason, the penalty method was used to handle the constraints. Penalty methods are motivated by the desire to use unconstrained optimization techniques to solve constrained problems. This was achieved by adding a penalty multiplied by a parameter. This is the penalty constant (P) that forces the solution to feasibility and subsequent optimum.

The effect of penalty parameter, P on Differential Evolution was studied by a parametric investigation. Ten chemical engineering problems were selected for this work. All of this problems were solved five times for P=10, 100, 1.000, 10.000, and 100.000 respectively. Table 9.1 shows these results. When a problem was highly dependent on constraints, then the violation was totally too large. Hence the penalty parameter had not to be a great number. On the other hand, when a problem was not highly dependent on constraints, then the sum of violations was relatively small than the objective function. So, penalty parameter had to be a large number.

In the eight section of this thesis, four different groups of problems were taken into account. These were unconstrained benchmark test problems, constrained benchmark test problems, chemical engineering problems, and nonlinear regression analysis problems respectively. Hence, totally 63 problems were studied in this thesis.

First of all, the unconstrained benchmark test problems were investigated. The algorithm (program) was run 10 times for each of the

test problems to determine the percentages of success (ps). As a result, the differential evolution algorithm has shown 100 percentages of success.

Then the constrained benchmark problems were investigated. Also this group of problems gave excellent results and the percent of success was 100.

The chemical engineering problems were investigated separately. Specially, highly nonlinear systems were selected for testing the program performance and efficiency. Almost all the problems gave good results.

All of these results are shown in Section 8.

Regression analysis is a tool which is required in most of the disciplines, from physics to engineering, from mathematics to economy. Generally Microsoft Office Excel is used for this purpose. But, the users have to determine starting values for each parameter which, affect the solution in great deal and leads to different results. For these reasons, as a final step in this thesis nonlinear regression analysis was investigated. The major difference between the Differential Evolution Algorithm and other nonlinear regression solver tools is the sensitivity of the other nonlinear regression solvers to starting points. The DE Algorithm is not sensitive to starting points. Consequently, Differential Evolution has been found to be very suitable for nonlinear regression analysis.

The results have been shown in Section 8.

Population based random search algorithms are new with respect to Newtonian approaches. Hence, a new field of research was introduced and a theoretical-practical background was established in this thesis. However, some parts of this thesis need further improvement. For example, the response time of the DE algorithm may be shotened by some technical and theoretical interventions.

BIBLIOGRAPHY

1. Bo Peng Bo Liu Fu-Yi Zhang , Ling Wang (2007), Differential evolution algorithm-based parameter estimation for chaotic systems, Chaos, Solutions and Fractals, article in press
2. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London, 1999, pp. 79–108.
3. D. Zaharie, Critical values for the control parameters of differential evolution algorithms, in: R. Matousek, P. Osmera (Eds.), Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing, Bruno, Czech Republic, Bruno University of Technology, Faculty of Mechanical Engineering, Bruno, Czech Republic, 2002, pp. 62–67.
4. Kenneth V. Price · Rainer M. Storn Jouni A. Lampinen(2006) Differential Evolution APractical Approach to Global Optimization Springer-Verlag
5. Kirkpatrick, S., Gelatt, C. D., & Vechhi, M. P. (1983), Optimization by simulated annealing, *Science*, 220(4598), 671–680.

6. Kohout M., Schreiber I., Kubicek M., *A Computational Tool for Nonlinear Dynamics in Process Systems Engineering*, Computers and Chemical Engineering 29, 1265, 2005
7. M.M. Ali, A. To¨ rn, Population set based global optimization algorithms: Some modifications and numerical studies, Computers and Operations Research 31 (10) (2004) 1703–1725.
8. M.M. Ali, C. Khompatraporn, Z.B. Zabinsky, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, Journal of Global Optimization, in press.
9. Marquardt W. & Mönnigmann M., *Constructive Nonlinear Dynamics in Process System Engineering*, Computers and Chemical Engineering 29, 2005
10. Onwubolu, G. C., & Babu, B. V. (2004). New optimization techniques in engineering. Heidelberg, Germany: Springer-Verlag.
11. P. Kaelo, M.M. Ali, A (2006), Numerical study of some modified differential evolution algorithms, European Journal of Operational Research 169, 1176–1184.

12. Price, K., & Storn, R. (1997). Differential evolution: A simple evolution strategy for fast optimization, *Dr. Dobb's Journal*, 22, 18–24 and 78.
13. R. Storn, K. Price, Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
14. Rakesh Angira Alladwar Santosh (2006), Optimization of dynamic systems: A trigonometric differential evolution approach, Computers and Chemical Engineering, 31, 1055–1063
15. http://en.wikipedia.org/wiki/Differential_evolution
16. http://www.icsi.berkeley.edu/~storn/code.html
17. http://www.itl.nist.gov/div898/strd/nls/nls_main.shtml
18. http://www.nlreg.com

RESUME

Ercüment DENİZ was born in Tekirdağ in 1982. She graduated from Tekirdağ Anatolian High School in 2001. He received the degree Bachelor of Science from Chemical Engineering Department of Ege University in 2006. At the same year, He was accepted to Chemical Engineering Department of Ege University for the degree of Master of Science. The subject of her thesis was “Differential Evolution for Optimization of Nonlinear Chemical Processes”. He has completed her M.Sc. thesis successfully in first week of July 2008.