國立成功大學
機械工程學系
碩　士　論　文

利用反應曲面輔助DIRECT演算法處理耗時工程設計問題

Surrogate-Assisted DIRECT Algorithm for Optimization Problems with Expensive Engineering Simulations

研 究 生：王東泰
指導教授：詹魁元博士

中 華 民 國 一 百 零 二 年 六 月

利用反應曲面輔助DIRECT演算法處理耗時工程設計問題

Surrogate-Assisted DIRECT Algorithm for Optimization Problems with Expensive Engineering Simulations

研 究 生：王東泰　　Student: Dung-Tai Wang

指導教授：詹魁元博士　　Advisor: Dr. K.-Y. Chan

國立成功大學

機械工程學系

碩士論文

A Thesis

Submitted to Department of Mechanical Engineering

National Cheng Kung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Dept. of Mech. Eng.

June 2013

Tainan, Taiwan

中華民國一百零二年六月

利用反應曲面輔助DIRECT演算法處理耗時工程設計問題

學生：王東泰　　指導教授：詹魁元博士

國立成功大學機械工程學系

摘　　要

在處理工程上的設計的問題時，高精確度的電腦模型模擬是一個重要的步驟，然而隨著要求的設計細節越多，電腦要花更長的時間完成運算，如果要將這些高運算成本的電腦模型整合於最佳化程序中，在現實的時間限制下，往往是一個非常大的挑戰。本論文修改一個全域最佳化演算法 DIviding RECTangles (DIRECT)，以期能更廣泛應用在實際的工程設計問題，更能在極少量的演算次數上便可達到理想的最佳化結果。 DIRECT 演算法的概念來自於 Shubert 演算法，Shubert 演算法是利用 Lipschitz 常數來進行最佳化的搜尋，但如何決定 Lipschitz 常數是一個複雜的問題，一旦Lipschitz常數計算有瑕疵，該演算法變化無法收斂。DIRECT 使用不同的方式決定 Lipschitz 常數，取而代之的是在每一個最佳化搜尋迴圈中利用比較局部 (local)與全域 (global) 的特徵，自然的產生一系列 Lipschitz 常數，然而現行的DIRECT有許多使用上的問題，爲求空間等份切割而只取終點代表，無法鄰近邊界，對於工程上多數位居邊界的最佳化結果使用上有所限制。再者，現行DIRECT無法針對拘束條件的違反加以量化並斟酌接受其結果，使演算法效率無法提昇。本論文利用反應曲面輔助提出一修正DIRECT演算法，故名爲 Surrogate-Assisted DIRECT。我們利用現有樣本建立空間曲面，依序完成子空間最佳化及空間切割的程序，取代本原本 DIRECT 演算法只取中點的特性，確保在子空間最佳化時能得到最近似理論值之結果，提昇整體演算效率。針對非線性拘束條件，我們利用一個來自懲罰 (penalty) 的概念，在每一個最佳化搜尋迴圈時，計算目標函數值及違背限制式的最大違背量來進行篩選以選定之母空間，以確定下一迴圈欲運行的子空間。我們利用現行文獻上15個數學範例進行比較，比較結果呈現出在少量的演算次數下，S.A. DIRECT有比較好的目標函數值，針對有非線性拘束條件的問題，S.A. DIRECT結果更是遠優於原本的 DIRECT 演算法。我們也利用一個皮帶輪系統的工程設計問題來呈現S.A. DIRECT的優勢，此問題的電腦工程模擬單次需耗時12小時以上，這是一般最佳化問題無法完成的任務，使用S.A. DIRECT可成功達成最佳設計，更進一步驗證本篇論文所提出的方法可以進行實際的應用。

Surrogate-Assisted DIRECT Algorithm for Optimization Problems with Expensive Engineering Simulations

Student: Dung-Tai Wang Advisor: Dr. K.-Y. Chan

Department of Mechanical Engineering
National Cheng Kung University

ABSTRACT

The use of high-fidelity models in the early stages of engineering design ensures accuracy but at the cost of computation. These expensive models present challenges in optimization routines, especially for practical engineering problems with a tight time constraint. This study modifies the global algorithm DIviding RECTangles (DIRECT) to improve engineering designs within a very small number of function evaluations. As an update to the standard Shubert algorithm, DIRECT does not require the calculation of the Lipschitz constant; instead, a set of Lipschitz constants are obtained sequentially by compromising the local versus global characteristics as the algorithm iterates. We use subspace optimum instead of the center as the local characteristic with the aid of surrogate modeling to ensure boundary optimum is obtained. We also use a penalty-inspired filter concept is used to make a trade-off between objective function values and constraint violations during design iteration. Numerical comparisons show that the proposed method gives performance improvements with few function evaluations. With an increase of function counts, the proposed method yields the same outcomes as those of the original DIRECT. A belt-tensioner-pulley design problem is used to demonstrated the applicability of the proposed method to engineering problems.

誌　　謝

承蒙恩師 詹教授魁元兩年多來的指導與教誨，從進入系統最佳化實驗室以來，除了在課業及研究上給予指導與幫助，其中更經歷人生中的許多大事，老師均給予許多支持與寶貴的建議與啓發，使這兩年的生活十分充實，師恩浩蕩，永誌於心。

同時，也要感謝口試委員－邱教授顯堂與陳教授家豪給予本研究以及論文許多指導與建議，使得本論文之內容可以臻於完善。

這兩年的日子中，要感謝成大系統最佳化實驗室的成員們。感謝學長姐們：yen-chih、子頡、季儒、勝昌、佳豪、彬儀、力豪、伊倩、Joseph Millogo，於研究與生活上給予許多建議與幫助。感謝同屆的同學：俏君、典運一起修課、做研究以及在實驗室各種事務上的幫助。感謝實驗室學弟妹們：庭玉、明證、值榕、肇余 、佑安、煜駿、 祐晨的鼓勵與陪伴。與新進學弟妹：彥彬、竹育、彥豪、承勳，帶給實驗室歡樂的氣氛。另外，感謝設計組各實驗室的同學們，無論修課、討論研究都讓碩士生活成爲我的美好回憶。

最後，感謝親愛的父母與家人，謹以此論文獻給我最摯愛的家人。

Table of Contents

List of Tables

List of Figures

List of Symbols

a_0	the average rate of change of the objective function.
a_j	the average rate of change of the jth constraint function.
c_j	the weight coefficient of the jth inequality constraint.
d	half of the longest diagonal.
$d_{\mathrm{c},i}^k$	the size of the kth subspace from the ith parent space.
f_*	the current best function value.
$f_{\mathrm{c},i}^k$	a sample of the kth result obtained by SQP from the ith parent space in S.A. DIRECT.
g_j^r	the violation value of the rth subspace violating the jth constraint.
l_i	the ith design variable lower bound.
lb_i	the distance between sample and the ith lower bound.
m	the number of all constraints.
n	the dimension (or number) of design variables.
$\mathcal{S}_{\mathrm{p},i}$	the ith parent space, selected from all subspaces in S.A. DIRECT.
$\mathcal{S}_{\mathrm{c},i}^k$	the kth subspace (or child space), produced from the ith parent space in S.A. DIRECT.
s_0	the sum of observed rates of change of the objective function.
s_j	the sum of observed rates of change of the jth constraint function.
u_i	the ith upper bound.
ub_i	the distance between sample and the ith upper bound.
$\mathbf{x}_{\mathrm{c},i}^k$	the kth sampling point obtained from the ith parent space.

θ	the relation of global and local characteristic with respect to the current optimum.
ε	the balance parameter, used for adjusting the process of selection, to avoid an excessively small subspace.
ϵ	$\epsilon = \varepsilon \times f_{min}$, used for adjusting the process of selection, to avoid an excessively small subspace.

Chapter 1 Introduction and Motivation

1.1 Introduction

In industry, engineering problems from concept to manufacture always need many design modifications. Which design modification is acceptable and suitable, it bases on the performance evaluations for design modifications. High-fidelity simulation models in engineering problems are commonly used in industry. Without these analysis models, the performance evaluations for design modifications require in-field testing, which is expensive and time-consuming. Capital costs and personnel training costs make these tests too expensive to assist design in the early stage of the product development process. Computer simulations are thus commonly used in performance evaluations. While simple analytical models can generally efficiently provide useful design insights into product design, high-fidelity computer simulations can provide thorough performance evaluations of design with great detail. The accuracy of high-fidelity models is higher than that of simplified analytical models at the cost of computation.

1.2 Motivation and Objective

In engineering practice, the engineering problems are always complex and time constraint. Unfortunately, the simulation cost for high-fidelity models could be too expensive to assist design in practice because industrial engineering problems have a very tight time constraint. Consider the design example of a belt-tensioner-pulley system with two pulleys and one steel cord reinforced belt with six v-shape grooves. A tensioner is added to provide enough tension within the system. Due to the geometric constraints of the system, locations of pulleys and the tensioner cannot be arbitrarily determined. Based on our preliminary investigation, a thorough belt-pulley-tensioner analysis that considers the layer materials and reinforcements of a multi-v belt with bending stiffness requires at least 12 hours on a high-end computer[1].

[1]CPU: Pentium Intel I7-3820, Memory: ADATA DDR3-8G-1333 ×8, motherboards: ASUS P9X79 PRO, Graphic Card: GIGABYTE N520-D3-1G, Hard Disc: WD 1.5TB (black title), Solid-State Disc Micron M4 256G, and Power: SeaSonic 430W (Bronze)

Less than two simulations can be obtained within a day and less than 14 analyses within a week. The associated high cost of high-fidelity simulations prohibits a serious of design modification. In addition, discrete testing outcomes are unable to provide suitable design performance implications that are generally continuous in nature.

In industry, design engineers require an optimization tool that can handle expensive simulations. A good optimizer algorithm should be able to :

1 Finish at any given time frame. In other words, given the time required for each performance simulation and the total time for design study, the algorithm should be able to terminate at the required time. In most cases, this condition translates to the termination criterion by function count.

2 Suspend the iteration progress, the most current design solution. All intermediate results should be feasible at the time the algorithm is suspended. In industry, this requirement is particularly practical when the actual time for each design study is unknown and subject to change at any time. Checking the most current progress at an any time can help designers make timely decisions.

3 Resume after being suspended. In practice, the time required for a design study may be extended based on higher-level management's comments. The extended time for design should also resume after being suspended and reveal in optimization outcomes. Therefore, the ability to use extended simulation time to improve outcomes is an important factor.

4 Improved outcome with increased of running time. When the time for a design study is extended, improved outcomes are expected. The algorithm should keep track of the best solution and ensure that the outcomes improved with time. As an additional requirement, the algorithm should avoid becoming trapping in a local optimum.

5 Allow simulation-based analysis without analytical functions. High-fidelity engineering simulations generally do not have explicit functions. Performing gradients of black-box simulations requires finite differencing and increases the computational burden. For a practical optimizer, the need for gradient information should be eliminated. However, that does not mean that rigorous mathematical foundations are not essential. Any practical

optimization algorithm should build upon rigorous academic developments with serious convergence arguments.

6 Have few parameters to tune. The ability of a practical algorithm to obtain converged solutions should not rely on the fine-tuning of algorithm parameters. The fewer number of tuning parameters of an algorithm, the more practical it can be in industry.

	DIRECT	SQP	GA
1	yes	yes	yes
2	yes	no	yes
3	yes	yes	yes
4	yes	no	yes
5	yes	no(gradient)	yes
6	3	60	32

Table 1.1: Comparison of DIRECT, SQP, and GA.

A comparison of the three most commonly used optimization algorithms in the literature, namely DIviding RECTangles algorithm (DIRECT)[2], Sequential Quadratic Programming (SQP)[3], and Genetic Algorithm (GA)[4], are compared with respect to desired features listed above. Table 1.1, DIRECT satisfies 1 to 5, and has three tuning parameters. SQP does not satisfy in 2, 4, and 5, because it may be infeasible in algorithm processing, SQP is a local search algorithm according to gradient. SQP has about 60 tuning parameters, such as TolX[5]. GA satisfy 1 to 5, and has about 32 tuning parameters, such as CrossoverFraction[6] and MigrationFraction[7]. Although the three algorithms have some parameters, most of them are defaults or the code has already been set. From Table 1.1, the most suitable algorithm is DIRECT.

The DIRECT algorithm is a well studied global searching algorithm. It is applied widely in many disciplines, including airfoil design [1], aircraft design [2], image processing [3], complex

[2]gclsolve in matlab is used here

[3]fmincon in matlab is used here

[4]ga in matlab is used here

[5]Termination tolerance on x, a positive scalar. The default value for the active-set and trust-region-reflective algorithms is 1e-6; for the interior-point algorithm the default is 1e-10.

[6]The fraction of the population at the next generation, not including elite children, that is created by the crossover function. The default is 0.8

[7]Scalar between 0 and 1 specifying the fraction of individuals in each subpopulation that migrates to a different subpopulation. The default is 0.2

computer simulation [4], expansion of electromagnetic fields [5], synchronous parallel hybrid optimization [6] [7], fuel control and performance of electric cars [8], face recognition [9], determination of the size and number of navigation stations [10], gear design [11], determination of the size of the hard disc drive air bearing [12], and gas transportation pipeline design [13]. DIRECT converts the boundary constraint space(lower bound and upper bound of design variables) into an initial parent space and then divides the parent space into three smaller subspaces and samples each center. According to the relative ratio of local and global characteristic of each subspace to current optimal subspace, it selects new parent spaces from all subspaces, again until the terminal condition. The concept is how to select the trisected parent space from all subspace, comes from Shubert algorithm, which uses a fixed Lipschitz constant. Estimating the Lipschitz constant is not straight forward and a a wrong Lipschitz constant value could alter the result completely. DIRECT therefore calculates Lipschitz constant by each subspace relative to current optimal subspace, then selects parent spaces by the Lipschitz concept during algorithm iteration. The DIRECT algorithm is terminated when the number of sampling points exceeds the set threshold and stores all sampling points. Hence, the DIRECT algorithm can be stopped at any time.

However center point of each subspace is not suitable for engineering problems, because optimal engineer problems are often on the boundary. Therefore, an infinite number of points must be sampled. Subspace optimum will replace sampling the center. And aide of surrogate modeling (Kriging model) ensures boundary optimum will be obtained. Adding a new filter, a penalty-inspired filter, by trades-off between local characteristics (objective function value) and constraint characteristics (constraints violation value) after selecting parent spaces by Lipschitz concept. Filtering selected parent spaces avoid many parent spaces, further decreasing the number of the sampling point. The proposed modified method is Surrogate-Assisted (S.A.) DIRECT.

1.3 Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 describes DIRECT algorithm in detail. Chapter 3 introduces the proposed algorithm including new strategies and modifications.

Chapter 4 presents numerical comparisons with boundary constraint problems and inequality constraint problems. Chapter 5 applies the proposed method to a belt-tensioner-puller design problem. Chapter 6 presents the conclusions and suggestions of this thesis.

Chapter 2 DIRECT algorithm and its modifications

2.1 DIRECT Algorithm

The DIRECT algorithm is an update of the Shubert algorithm that does not specify a fixed Lipschitz constant. By doing so, the difficulty in obtaining an accurate Lipschitz constant is eliminated along with the convergence problem accompanying inaccurate Lipschitz constant values. The algorithmic process of DIRECT can be summarized as shown in Fig.2.1. Consider simple bound-constrained optimization problem in Eq.(2.1). f is a single objective function of design variables $\mathbf{x}$ and parameters $\mathbf{p}$. $\mathbf{x}$ is bounded between $\mathbf{x}_{\text{lb}}$ and $\mathbf{x}_{\text{ub}}$. The first step of the DIRECT algorithm, the problem definition, is to model the problem with a proper formulation and to identify design variables and parameters of interest. The bounds on all design variables should also be provided.

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}, \mathbf{p}) \\ &\text{s.t. } \mathbf{x}_{\text{lb}} \leq \mathbf{x} \leq \mathbf{x}_{\text{ub}} \end{aligned} \tag{2.1}$$

The bounds on design variables determine the size of the feasible design space, denoted as $\mathcal{S}$. The original DIRECT algorithm then iterates between selecting best candidates for further subspace division, dividing space into several subspace, and sampling in each subspaces. In what follows, Each step in Fig.2.1 is described below.

This thesis is concerned with the optimization of practical engineering problems, which usually have both equality and inequality constraints. Since most DIRECT applications in the literature consider unconstrained problems only, the DIRECT algorithm for unconstrained problems is discussed first. The extensions to inequality constrained and equality constrained problems will be discussed later.

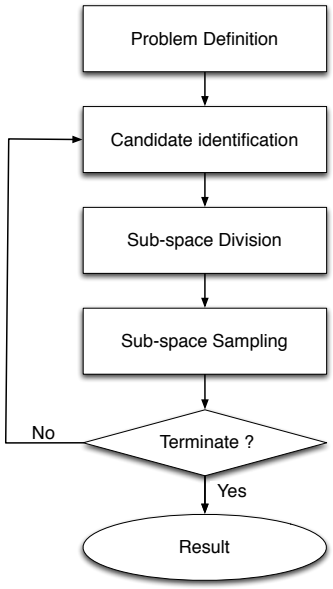


Figure 2.1: DIRECT flowchart.

2.2 Candidate Identification

2.2.1 Concept

As an initialization step, the bound-constrained feasible space $\mathcal{S}$ is selected as the candidate space. To clearly describe the candidate identification step in DIRECT, let us first separate the feasible design space $\mathcal{S}$ into the parent space $\mathcal{S}_{\mathrm{p}}$ and (child) subspace $\mathcal{S}_{\mathrm{c}}$. To relate $\mathcal{S}_{\mathrm{p}}$ with its possibly multiple children subspace $\mathcal{S}_{\mathrm{c}}$, let us further state that the sets $\mathcal{S}_{\mathrm{c},i}$ and $\mathcal{S}_{\mathrm{c},j}$ are mutually exclusive $\forall i \neq j$. $\{\mathcal{S}^k_{\mathrm{c},i} \forall k \in c_i\}$ also form the exhaustive set of the parent space $\mathcal{S}_{\mathrm{p},i}$. c_i is the number of (children) subspaces for the ith parent space. For example, the three subspaces of the parent space $\mathcal{S}_{\mathrm{p},i}$ are $\{\mathcal{S}_{\mathrm{c},i}\} = \{\mathcal{S}^1_{\mathrm{c},i}, \mathcal{S}^2_{\mathrm{c},i}, \mathcal{S}^3_{\mathrm{c},i}\}$. Each subspace is associated with a sample to represent the objective function value of the subspace. Let the sample for the subspace $\mathcal{S}^k_{\mathrm{c},i}$ be $\mathbf{x}^k_{\mathrm{c},i}$ and the corresponding objective function value be $f^k_{\mathrm{c},i}$. The DIRECT algorithm identifies the candidates based on the values of $f^k_{\mathrm{c},i}$ as well as the size $\mathcal{S}^k_{\mathrm{c},i}$, measured by the longest diagonal length, $d^k_{\mathrm{c},i}$. Inspired by the Shubert algorithm, which uses a known and fixed Lipschitz constant, DIRECT uses the Lipschitz concept without a Lipschitz constant by making a trade-off between local objective function $f^k_{\mathrm{c},i}$ and global information $d^k_{\mathrm{c},i}$ via a figure called the $f - d$ diagram, shown in Fig.2.2.

The x-axis in Fig.2.2 is the largest diagonal distance $d^k_{\mathrm{c},i}$ of the subspace $\mathcal{S}^k_{\mathrm{c},i}$, representing

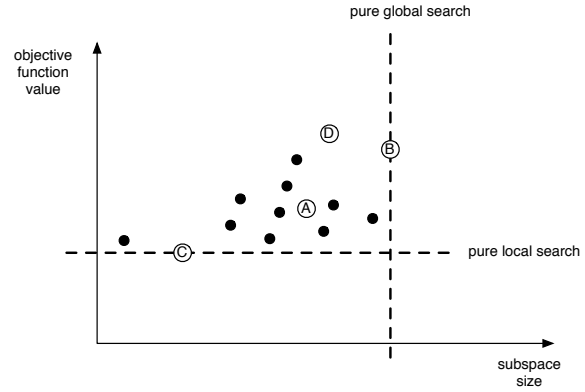


Figure 2.2: f-d diagram in candidate identification.

the global information of sample $\mathbf{x}_{\mathrm{c},i}^k$, the y-axis is the objective function value representing the local information. The Lipschitz constant is the slope of the line in Fig.2.2 that shows the compromise between local and global information of a Shubert optimization. When the Lipschitz constant is zero, the minimal $d_{\mathrm{c},i}^k$ is reached at point C. When a very large Lipschitz constant is used, the maximal $d_{\mathrm{c},i}^k$ is asymptotically found at point B in Fig.2.2. Between points B and C, points A and D do not have as small an objective function value that of C and not as large a subspace that of B. The design point A is a compromise between B and C while the design point D is not as superior as B in both aspects. Therefore, among the four design outcomes $\{A, B, C, D\}$, designs $\{A, B, C\}$ outperform D at the kth iteration. DIRECT then iteratively generates more design points as shown with solid dots in Fig.2.2. The selection of non-dominated design points using the $f - d$ diagram is the candidate selection process in DIRECT. The non-dominated points in the $f - d$ diagram are found by connecting the designs with a minimum slope, as given in Eq.(2.2), where n is the number of designs in the $f - d$ diagram and i is the current candidate.

$$\theta = \frac{\Delta f}{\Delta d} = \frac{f_i - f_*}{d_i - d_*},\ k = 1, \cdots, n \tag{2.2}$$

The initial f_* is the one with the minimum objective function in the $f - d$ diagram. Once a candidate is obtained, the algorithm keeps looking for more candidates. For example, let us look at the case in Fig.2.3 with $k = A, B, C, D$. Since f_B has the smallest objective function value, $f_* = f_B$. Calculating the θ values of designs A, C, and D with respect to B, design D is found to have the minimal θ. Therefore the candidates in Fig.2.3 are B and D. Design C, despite having a better objective function value than that of D, is dominated by the line connecting B-D.

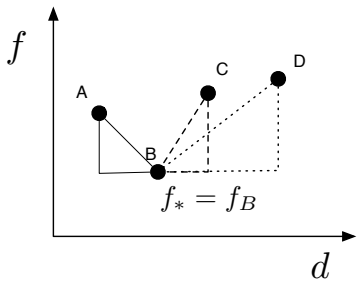


Figure 2.3: Non-dominated points in $f - d$ diagram.

Connecting the non-dominated subspaces results in the lower bound envelope in the $f - d$ diagram. These non-dominated subspaces are the candidates for further exploration. Since the slope obtained by connecting two adjacent non-dominated points in the $f - d$ diagram shows the trade-offs between global and local characteristics, such as the Lipschitz constants. One can view the construction of the envelope as the selection of varying Lipschitz constants.

2.2.2 Modifications to candidate identification

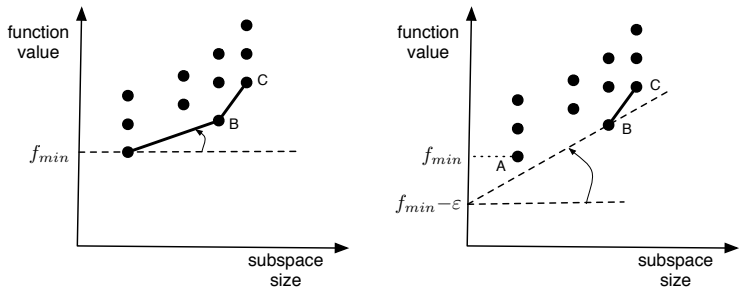


Figure 2.4: Convex hull of f-d diagram in candidate identification.

Candidate identification is the lower-right convex hull, that starts from f_{min} sweeps via the dash line the first parent space is A then B and C in Fig.2.4 left, DIRECT modifies small in candidate identification to avoid very small improvements by $\varepsilon = \epsilon|f_{\min}|$, in Fig.2.4 right, that starts from $(f_{min} - \varepsilon)$ will have two parent space B and C, not include A. Therefore the different balance parameter makes different the first parent space. The following lists some modifications of ϵ and ε:

- Modification of the balance parameter(ϵ)

1. C.D. Perttenun, D.R. Jones, and B.E. Stuckman estimated ϵ using $\epsilon = \max(\frac{10^{-4}}{f_{min}}, 10^{-8})$ [14].
2. D.R. Jones estimated ϵ using a fixed $\epsilon(10^{-4})$, and suggested the $\epsilon : 10^{-7} \sim 10^{-3}$ [11].
3. D.E. Finkel and C.T. Keley estimated ϵ using alternative value ($\epsilon : 0\ 10^{-2}$) during the algorithm iteration. Initial ϵ is 0. When the result did not improve over $0.01\%(10^{-4})$ after several loops, the $\epsilon(\leq 10^{-2})$ was updated. When the improvement was over $0.01\%(10^{-4})$, the ϵ returned to 0 [15].

- Modification of ε

1. D.E. Finkel and C.T. Keley estimated ε using $\epsilon|f_* - f_{\text{median}}|$, where $\epsilon = 10^{-4}$, f_* is current minimum function value and f_{median} is the median in all function value (f) [16].
2. O. Liu estimated ε using $\epsilon|f_* - f_{\text{aver}}|$, where f_* is the current minimum function value, and f_{aver} is the average of all known function value (f) between first and third quartiles ($f|f_{Q_1} \leq f \leq f_{Q_3}$, where Q_1 and Q_3 are first and third quartiles of all f, respectively) [17].

These modifications are about candidate identification, their numerical comparisons are superior to the original DIRECT or not. And all do not present about problem with constraints. They did not present in problems with constraints.

2.3 Space Division and Sampling the Center of Subspace

2.3.1 Concept

An identified candidate subspace from the $f - d$ diagram will become the parent space $\mathcal{S}_{\text{p}}$. DIRECT takes advantage of the unique center point in multiple dimensions and uses the objective function in the center point as the f value in the $f - d$ diagram. Each parent space is further divided into three subspaces. This trisection procedure ensures that the previous center point remains as the center of one of the subspaces. The new subspaces are then evaluated in terms

of their objective function values at the center and their largest diagonal length. The $f-d$ diagram is then updated with the new information from child spaces and the algorithm iterates between selecting candidates and dividing spaces until the termination criterion is satisfied.

2.3.2 Modifications to space division and sampling

DIRECT trisects and samples the center. Sampling the center, ignoring other positions. The following lists some sampling modifications.

- N. Goldberg, T.G. Kolde, and A.S. Yoshimura added two points per loop (random or optimum algorithm) besides the center of subspaces with three strategies [18]:

strategy 1 : Besides the center, subspace has the adding point, the represent of subspace was the smaller function value.

strategy 2 : Including strategy 1, The original DIRECT algorithm cuts the longest side of parent space is replaced by cutting the side in random.

strategy 3 : Because of adding two points every loop, subspace has adding point does not sample the center, decreasing the number of sampling point.

- Sergeyev et al. sampled the vertices of diagonals of the subspace [19]. Figure 2.5 show the first two iterations, initial the lower boundary (a) and upper boundary (b) are known. After trisecting the first parent space. Besides sampling a and b, 3 and 4 are sampled (on the left side of Fig.2.5, because in the left subspace sampling point a is known and its vertices of diagonal is 3, so sampling point 3 is sampled. In the right subspace likes the left subspace, sampling point 4 is sampled. The center subspace includes 3 and 4 does not sample any point). In the second loop, the left subspace (on the left side of Fig.2.5) is selected to parent space. Trisect the parent space on the right side of Fig2.5 and sample the sampling point 5 and 6. Then the two subspaces are selected to be parent spaces (gray subspaces on the right side of Fig.2.5).

  In the candidate subspace, the function value on behalf of subspaces, f, is replaced by F (the mean of the function values is the two sampling point on the vertices of diagonals of subspace, $F_i = \frac{f(b_i)-f(a_i)}{2}$).

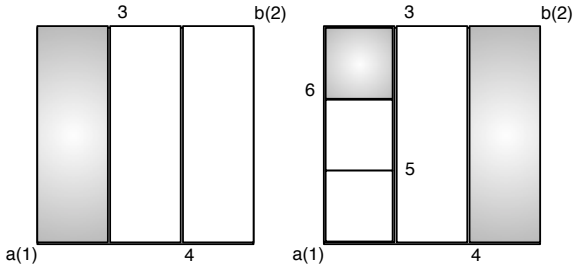


Figure 2.5: Sampling the vertices of diagonals of subspace.

2.4 Termination

2.4.1 Concept

The DIRECT algorithm terminates after evaluating a user-defined number of function evaluations. It is thus known how long a given optimization problem will run. In practical engineering problems where we could know the computational time of an analysis, DIRECT lets the user decide the running time based on time constraints. The more function evaluations, the better results DIRECT can achieve.

2.4.2 Modifications to termination

If theoretical optimum is known, the original DIRECT algorithm add the terminal condition, $p \leq 0.1\%$ (p: percent error), the following list about p.

1. J.M. Gablonsky and C.T. Kelley estimated p using [20]:

$$p = 100 \begin{cases} \dfrac{f_* - f_{\text{global}}}{|f_{\text{global}}|}, & f_{\text{global}} \neq 0 \\ f_* - f_{\text{global}}, & f_{\text{global}} = 0 \end{cases}$$

   ,where f_* is the current minimum function value and f_{global} is the theoretical optimum.

2. O. Liu estimated p using [17]:

$$p = 100 \begin{cases} \dfrac{f_* - f_{\text{global}}}{|f_{\text{global}} - b|}, & f_{\text{global}} \neq 0 \\ f_* - f_{\text{global}}, & f_{\text{global}} = 0 \end{cases}$$

The concept came from linear algebra: $\min(f(x)) \rightarrow \min(af(x)+b)$, where f_* is the current minimum function value, f_{global} is the theoretical optimum, and b any number (the user decides it).

In practice, the theoretical optimum is unknown. Using the above modifications is not convenient.

2.5 Demonstration

Takes the two design variables for example in the first three loops, Figure 2.6 the left side are candidate identification and the right side are trisection and sampling of the first three iterations. In iteration 1 (or loop 1), there is only design space, chooses it to be parent space is trivial, then trisects it and samples the center of the three subspaces. Iteration 2, there are three subspaces, chooses one subspace (gray subspace on the middle left of Fig.2.6, iteration 2) to be parent space by $f-d$ filter, then trisects it and samples the center of the subspaces. Iteration 3, there are five subspaces, chooses two subspaces (gray subspaces on the left bottom of Fig.2.6, iteration 2) to be parent spaces by f-d relation, then trisects them and samples the center of the subspace.
Expanding to N variables, the following the general sequence:

1. Normalizes the design space to initial parent space, going to 3.
2. Selects parent spaces from all subspaces by $f-d$ filter for further trisecting them and sampling each center of subspaces.
3. Trisects each parent space along the longest side, and samples the center of subspaces.
4. Compares all function value of sampling points (f) and updates the subset.
5. Calculates the total function counter, the DIRECT algorithm is stopped when function counter is over the user sets, otherwise return step 2.

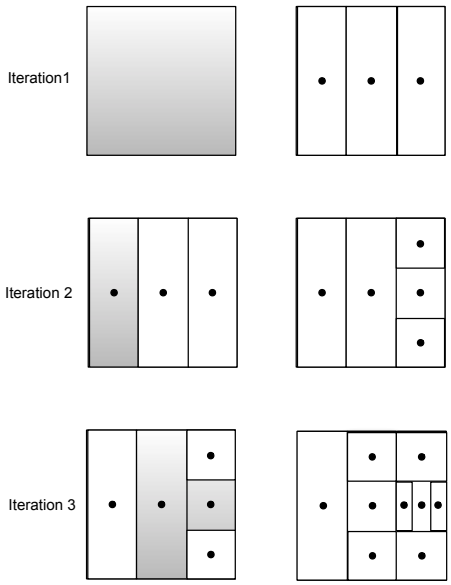


Figure 2.6: DIRECT algorithm.

2.6 DIRECT in constrained optimization

The original DIRECT algorithm handles the bound constraints with a rectangular feasible space. The extensions to general nonlinear inequality and equality constraints that are commonplace in engineering practice have been studied in various forms. Most studies in the literature treat constraints in DIRECT as hidden (removing or replacing the infeasible design) or as penalties.

2.6.1 Methods to Handle Inequality Constraints

The standard DIRECT algorithm uses the $f - d$ filter (f-d diagram) determine the candidate spaces without considering the feasibility of these spaces subject to nonlinear inequality constraints. Existing methods for dealing with general nonlinear constraints include:

1. Removing infeasible designs: this method removes infeasible design points by replacing the objective function with a large number, such as 10^6 [13]. By doing so, infeasible designs are implicitly removed from the $f - d$ diagram to ensure that no subspace with infeasible an midpoint can be an optimum. Through this removing infeasible designs, DIRECT will reexamine areas where it has already found a feasible point before considering infeasible

subspaces for further division.

2. Creating extra information from infeasible designs: this method separates feasible spaces from infeasible subspaces. At each design iteration, one of the infeasible space is selected randomly and three equally separated samples are taken as the additional information. This extra information from infeasible spaces combined with the existing feasible design is used in the candidate selection of the next iteration. However, the random selection of infeasible subspaces in this method has been shown to be ineffective [13].

3. Expanding subspaces: this method expands an infeasible space by a factor of two and searches for existing feasible design points in the expanded subspace. The smallest (best) feasible objective function value in the expanded subspace replaces the original infeasible midpoint since it is the best nearby objective [13]. An objective supplement might be added to the best objective function value to reveal the fact that the midpoint in the original subspace is not as good as the best nearby feasible point. If the expanded subspace contains no feasible design points, this method removes the subspace information from the $f - d$ diagram by assigning a large objective function value.

4. Treating infeasibility as a penalty: this method adds the infeasibility of a design to its objective function as a penalty. To distinguish an infeasible subspace with a relatively 'good' objective function value from one with a 'bad' objective function value, the composite objective function with a penalty becomes:

$$h(f) = \max(f - f_*, 0) + \sum_{j=1}^{m} c_j \times \max(g_j, 0) \tag{2.3}$$

where f_* is the best objective function value in the current $f - d$ diagram, c_j is a positive weighting coefficient for the jth inequality constraint and g_j is the jth constraint.
Studies have considered the feasibility of a subspace by replacing the objective function values in the $f - d$ diagram with the concept of penalties. Recall the candidate selection in Fig.2.2 and 2.3. The concept of adding a penalty to an infeasible design is done by

updating Eq.(2.2) to Eq.(2.4):

$$\theta = \begin{cases} \dfrac{\sum_{j=1}^{m} c_j \times \max(g_{rj}, 0)}{d_k} & \text{, infeasible...(a)} \\ \dfrac{\max(f_k - f_*, 0) + \sum_{j=1}^{M} c_j \times \max(g_{k,j}, 0)}{d_k} & \text{, feasible...(b)} \end{cases} \tag{2.4}$$

DIRECT combines the objective function and constraint function into an auxiliary function. There two situations:

(a) If all sampling points are infeasible, only the constraint is considered (the objective function is neglected) and only one parent space that has minimum Eq.2.4-a is selected.

(b) Otherwise (at least one sampling point is feasible), Eq.2.2 is updated to Eq.2.4-a and the f-d ($-d$ filter) diagram becomes the h-d diagram ($h - d$ filter).

In order to understand how to calculate constraint coefficient c_j (the jth constraint coefficient), at first, calculates the average rate of change of object function (a_0) and the average rate of change of the jth constraint (a_j), suppose that the center of subspace violations the jth constraint ($g_j > 0$), seems to be moving a distance (equal g_j/a_j) to get rid of the j constraint. If the motion makes the object function worse (increase), it increases by $a_0 \times (g_j/a_j) = (a_0/a_j) \times (g_j)$, likes to set a converting way from the infeasible zone to feasible zone by increasing the object function, so will set $c_j = a_0/a_j$.

Calculates the sum of observed rates of change of the objective function (s_0) and the sum of observed rates of change of the jth constraint (s_j), initial are 0, then updates after each parent space is trisected:

$$\begin{cases} s_0 = s_0 + \sum_{\text{child=left}}^{\text{right}} \dfrac{|f(x^{\text{child}}) - f(x^{\text{mid}})|}{\|x^{\text{child}} - x^{\text{mid}}\|} \\ s_j = s_j + \sum_{\text{child=left}}^{\text{right}} \dfrac{|g_j(x^{\text{child}}) - g_j(x^{\text{mid}})|}{\|x^{\text{child}} - x^{\text{mid}}\|} \end{cases}$$

x^{mid}: the midpoint of the parent space

x^{left} and x^{right}: the midpoints of the left and right subspaces after the parent space is trisected, respectively

Then, the average rates of change become $a_0 = s_0/N$ and $a_j = s_j/N$, where N is the

number of sampling points:

$$\frac{a_0}{a_j} = \frac{\frac{s_0}{N}}{\frac{s_j}{N}} = \frac{s_0}{s_j}$$

Therefore,

$$c_j = \frac{s_0}{\max(s_j, 10^{-30})}$$

where the maximum operation is used to avoid division by 0.

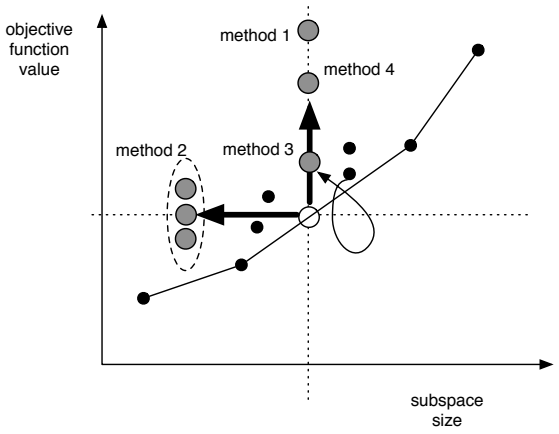


Figure 2.7: Existing methods for handling infeasible design points.

Figure 2.7 compares the four inequality constraint handling methods mentioned above. The solid points are the current feasible design with the envelope profiled as shown. The hollow point in the center is an infeasible design point whose location is the original objective function value f and subspace size d. Method 2 tries to divide an infeasible subspace into smaller subspaces and looks for potentially feasible designs, whereas methods 1, 3, and 4 replace the original objective function value of an infeasible design. With the objective function replacement, method 1 basically removes the infeasible design from the candidate set in the $f - d$ diagram. Method 3 emphasizes the neighborhood and the method 4 tries to balance the objective function value with constraints violations. Method 2 has shown to be ineffective due to the randomness in dividing an infeasible subset [13]. The underlying assumption of method 2 is that there are potentially feasible smaller subsets hidden in a subset that appears to be mid-point infeasible. If this assumption does not hold, method 2 will not work.

Consider the case with multiple infeasible subspaces in the selection of the parent space. On left of Fig.2.8, {A, B, and C} are selected without constraint. In the center of Fig.2.8,

according to the eq.2.4, will select {C}. On the right of Fig.2.8, the DIRECT algorithm selects only {C}. In this case, A has a better function value and larger total violation of constrains. The DIRECT algorithm will not select A because it considers the objective function and total violation of constrains.

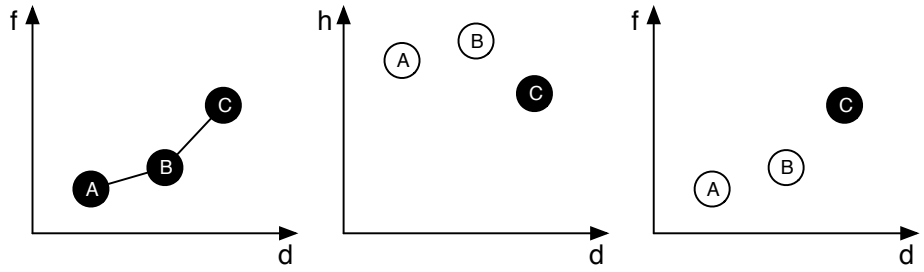


Figure 2.8: Handling multiple infeasible design points

2.6.2 Methods to Handle Equality constraint

Equality constraints represent physical phenomena. DIRECT does not explicitly handle equality constraints. Problems with equalities can often be rewritten as problems with inequality constraints (either by replacing the equality constraint with an inequality constraint that becomes binding in the solution, or by using the equality to eliminate variables).

2.7 Other modifications

Modifications that are mentioned early comes from flowchart. The following lists other modifications about DIRECT.

1. S.E. Cox, R.T. Haftka, C.A. Baker, B. Grossman, W.H. Mason, and L.T. Watson suggested limited the size of the subspaces to improve the efficiency, adding terminal condition: when the longest diagonal of the minimum subspace is 0.01% times less than the longest diagonal of the design space ($\frac{d_{\min}}{d_{\text{initial}}} \leq 0.01\%$) than design space, the DIRECT algorithm is terminated [21].

2. H. Zhu and D.B. Bogy suggested the side of the parent spaces in division direction must be larger than the limits. To avoid the subspaces are too small [12]. Assuming a parent space has two sides and the limit is 0.003. If the two sides are over 0.003, trisects along with the longest side. If one side is over 0.003, trisects along with this side. If the two sides less than 0.003, the parent space does not trisect. The limit is decided according to practical design problems.

3. J.He, L. T. Watson, N. Ramakrishnan, C.A. Shaffer,A. Verstak, J. Jiang, K.Bae, and W.H. Tranter suggested to limit the number of subspaces that have the same longest diagonal (d) [22]. Assuming the limit is five subspaces. When subspaces of the same d is seven, the two subspaces are remove in the f-d diagram.

4. J.M. Gablonsky and C.T. Kelly replaced the longest diagonal of the subspaces with the longest side of the subspaces in measuring the size of subspaces. [20]

The DIRECT algorithm repeats selecting parent spaces, trisects parent spaces, and sampling each center. DIRECT is unable to sample on corners, boundaries, or vertices. Complex problems make the process of selecting parent spaces inefficient and increase the number of sampling points. The objective function value changes rapidly and optimum point is neighbor center but the objective function value in center is very large. The subspace may not be selected be the parent space.

Chapter 3 Surrogate-Assisted DIRECT

In industry, engineering problems with constraints are often complex and have optima on corners, boundaries, or vertices. Engineering problems that are highly nonlinear make the objective function value change rapidly or no mathematical equations. The proposed method, Surrogate-Assisted (S.A.) DIRECT major in engineering problems.

3.1 Algorithm overview

Figure 3.1 shows a flowchart of the proposed method. Problem definition such as DIRECT. In candidate identification, only the f-d concept is used (constraints are neglected) and a filter that filters parent spaces based on relation between the objective function value and constraint function value is added. In subspace optimum, surrogate modeling is used to ensure that the boundary optimum is obtained. We does not sample center directly and sampling uses SQP. In subspace division, a parent space is divided into three subspaces. Subspaces will be divided from parent space after subspace optimum. Measure the distance that are between the sampling point and parent space. Based on the minimum distance, subspace will be divided from parent space. The three subspaces come from parent space will be divided one after another. Because the three subspaces dose not be divided at once, hence they are often the same size. In terminate such DIRECT, user decides number of sampling point. The proposed method, S.A. DIRECT, will be introduced according to the sequence of flowchart in details.

3.2 Candidate identification

In candidate identification, we start from outward and characteristic in f-d plot. Figure 3.2 has two f-d plots; the left side is DIRECT and the right side is S.A. DIRECT. DIRECT trisects the parent space at one once, the f-d plot has neat columns (in left side of Figure 3.2 has three columns, there are three different d). S.A. DIRECT divides the parent space into three subspaces one by one, so the three subspaces often have different size. The f-d plot of S.A. DIRECT does not have neat columns (d is different). In Fig.3.2, the two f-d plots have the

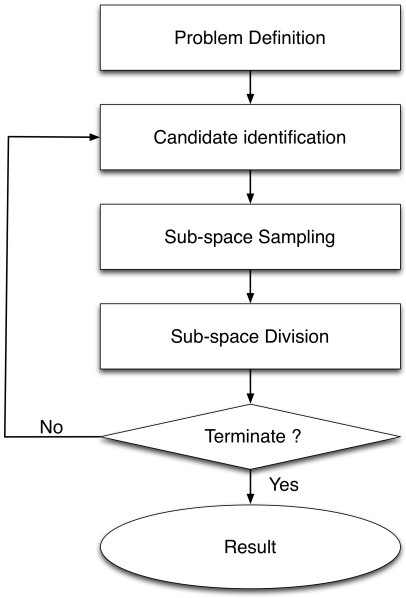


Figure 3.1: S.A. DIRECT flowchart.

same numbers of subspaces. S.A. DIRECT selects more parent spaces than DIRECT. More parent spaces means more sampling points. In order to decrease the number of sampling points, a filtering mechanism is necessary.

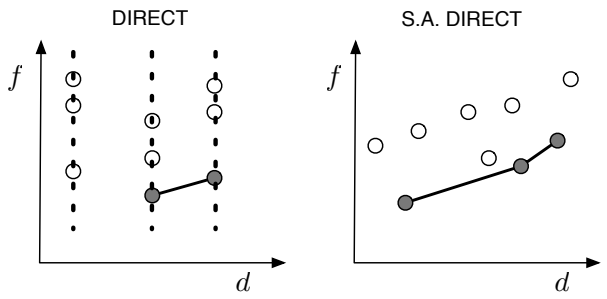


Figure 3.2: f-d plots of DIRECT and S.A. DIRECT.

In the process of candidate identification, S.A. DIRECT selects parent spaces using the $f - d$ filter (neglecting constraints) and then filters currently selected parent spaces according to the relation between constraints and objective function value of parent paces. The proposed $f - g$ filter is described in detail below.

3.3 $f - g$ filter

An f-d plot can be used to explain the $f - d$ filter. Similarly, an f-g plot is used here to explain the $f - g$ filter. Selected parent spaces are filtered or not, according to relation between most influence constraint value and objective function value of each parent space. The most influential constraint is the maximum violation value of the constraint of each subspace:

$$g = \begin{cases} \max(g_j), \text{infeasible} \\ 0, \text{feasible} \end{cases} \tag{3.1}$$

Eq.3.1, g_j is the jth constraint. $g_j = 0$ means that the constraint is feasible in the jth constraint. $\max(g_j)$ means that each subspace selects the maximum violation value in all constraints. $g = 0$ means that all constraints are feasible. According to Eq.3.1, the individual maximum violation value of each selected parent space from the $f - d$ filter is obtained.

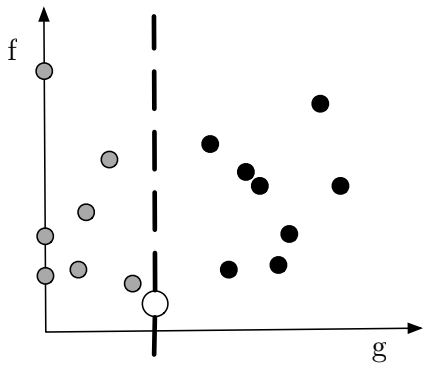


Figure 3.3: $f - g$ filter of S.A DIRECT.

Figure 3.3 show an f-g plot to explain the$f - g$ filter. A dot, (f, g), represents a parent space. The dots come from the $f - d$ filter. If a parent space is feasible in all constraints(i.e., $g_{max} = 0$), the parent space is acceptable and passes the $f - g$ filter. In Fig.3.3, three parent spaces pass the $f - g$ filter. We find the minimum function value(f) from the infeasible dots (parent spaces) to be reference point(the bigger point in Fig.3.3). Then we divide two parts form the reference point. Dots (gray points in Fig.3.3) on the left part and reference dot pass the $f - g$ filter. The other dots (dark points in Fig.3.3) are removed from the f-g plot. Fig.3.3, eight dots, eight parent subspaces will pass the $f - g$ filter. Besides the three feasible parent spaces, S.A. DIRECT also tries to sample and divide in these five infeasible parent spaces further. S.A. DIRECT tries to find the better and feasible results by relaxing the constraints.

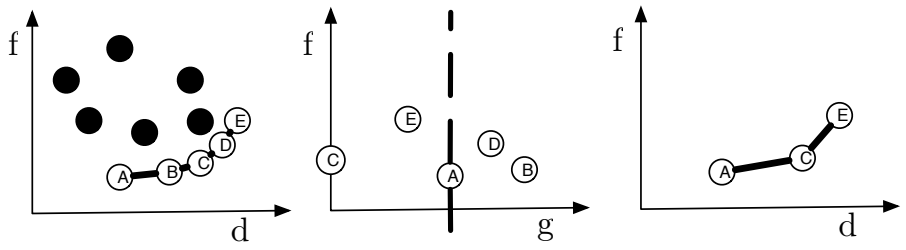


Figure 3.4: Candidate identification of S.A. DIRECT.

Figure 3.4 shows the candidate identification process for S.A. DIRECT. Three steps are used to explain the sequence of candidate identification and how to combine selecting the parent spaces using the $f-d$ filter and filtering selected parent spaces using the $f-g$ filter. In the first step, all subspaces are plotted in f-d plot without constraints (left side of Fig.3.4). Parent spaces $A, B, C, D,$ and E are selected by $f-d$ filter. Then, according to the objective function value and most influence constraint (maximum violation value of constraint using the eq.3.1) of each subspace, the current selected parent spaces $A, B, C, D,$ and E are mapped to the f-g plot (center of Fig.3.4). In second step, the minimum function value A is found and an auxiliary dashed line is plotted from A. Only $A, C,$ and E pass the $f-g$ filter (center of Fig.3.4). In last step, B and D are remove from the f-d plot(the right side of Fig.3.4). In S.A. DIRECT, $A, C,$ and E are selected by the $f-d$ and filtered by the $f-g$ filter.

3.4 Subspace optimization and division

Each parent space is divided into three subspaces, but sampling points are not necessarily in the center. Sampling the center directly ignores other possibilities. To find subspace optimum, a different sampling method is necessary. In order to ensure that the subspace optimum is obtained, surrogate modeling and an improved division method are used. Because the center in a subspace is the most significant and representative, the sampling point is as much as possible at the center of the subspace. The subspace after sampling is separated by subspace optimum from the parent space and the sampling point is in the center of the subspace. The first sampling point comes from the parent space. The parent space divides into first subspace include first sampling point and remained parent space after the first sampling point. The second sampling point comes from the remained parent space. Then the remained parent space divides into the

second subspace include the second sampling point and the third subspace. The last, the third sampling point find from the third subspace. Hence the three subspaces are divided one after another. The subspace optimum and subspace division are described below.

3.4.1 Subspace optimum

Subspace optimum, sampling is a boundary optimum problem. Sampling the center is replaced by SQP. To ensure that the boundary optimum is obtained, Surrogate model is necessary. Besides sampling points and their objective function values are known. The other objective function values are unknown. So the optima also cannot sample. The subspace optimum is that using response surface predicts other objective function values, and then SQP samples the sampling point from response surface. The response surface needs samples and their objective function values. And there are more samples and their objective function values, the prediction is more accuracy. So a response surface is built again in every algorithm iteration. Take for example the two design variables in Fig.3.5, namely X_1 and X_2. On the left side of Fig.3.5, according to all samples (at beginning, there is no sample and giving random samples) and their function values, the response surface, the Kriging model (right side of Fig.3.5, like contour), is built. The region of the parent space is selected from Kriging model and each parent space is sampled three sampling points. Subspace optimum is that by SQP samples in Kriging model from the region of the parent space.

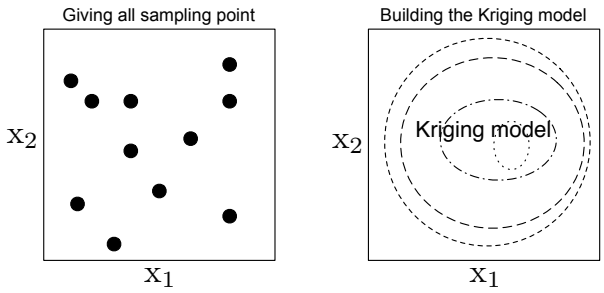


Figure 3.5: Building Kriging model.

The following are used in surrogate modeling:

1. Sequential Quadratic Programming(SQP) [23]: SQP is a local minimum algorithm. It is well studied in nonlinear constraint optimization. It combining the objective function

and constraint function into a Lagrangian function. Giving initial point, then find optima by the gradient. S.A. DIRECT uses fmincon " in MATLAB.

2. Kriging model [24] [25]: The Kriging model interpolates statistics. It has been in a variety of disciplines, including environmental science, hydrogeology, mining, natural resources, remote sensing, and real estate appraisal. The value of an unknown point is evaluated by the average of the distance-weighted values of know points neighbor the unknown point. The variance and affected range of the unknown point are considered. S.A. DIRECT uses the Kriging model, because the Kriging model is a smooth interpolation and the value of know points has no bias.

3.4.2 Subspace division

Two design variables are used as an example to explain how to divide the parent space into three subspaces. They are shown in Fig.3.6.

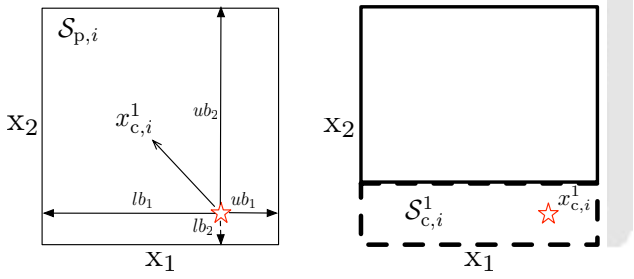


Figure 3.6: First sample and subspace of S.A. DIRECT.

On the left side of Fig.3.6, the subspace optimum uses SQP and the Kriging model to sample the first sampling point from parent space (the red hollow star, $\mathbf{x}^1_{\mathrm{c},i}$). The distance between the first sampling point and the boundary of the parent space is measured (lb_1, lb_2, ub_1, and ub_2 on the left side of Fig.3.6). The minimum distance is found from the direction, the long sides of the parent space. According to the minimum distance and its direction, the first subspace including the first sample is divided from the parent space. On the left side of Fig.3.6, the minimum distance is lb_2 (dashed-line) and its direction is X_2 in this example. The minimum distance from the nearest boundary of the parent space is doubled in the direction of the minimum distance (on the left side of Fig.3.6, this example is the lower boundary in the X_2 direction). Let the first sampling point ($\mathbf{x}^1_{\mathrm{c},i}$) be in the center of the first subspace ($\mathcal{S}^1_{\mathrm{c},i}$) in X_2 direction.

The first subspace including the first sample (the dash-line rectangle in right side of Fig.3.6) is divided from parent space. The remained parent space (no sample, solid-line rectangle on the right side of Fig.3.6) is used to sample the second and third sampling points.

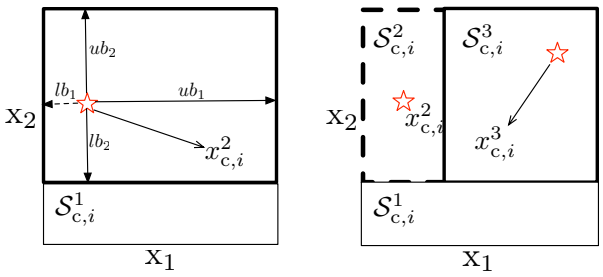


Figure 3.7: Second and third samples and subspaces of S.A. DIRECT.

Figure 3.7 shows the second and third samples. On left side of Fig.3.7, subspace optimum uses SQP and the Kriging model to sample the second point (the red hollow star, $\mathbf{x}^2_{c,i}$). The distance is measured between the second sample and the remained parent space (lb_1, lb_2, ub_1, and ub_2 on the left side of Fig.3.7). The minimum distance is found from the direction, the long sides of the parent space(in this example is X_1). The minimum distance in this example is lb_1 and its direction is X_1. On right side of Fig.3.7, the minimum distance (lb_1) from the nearest boundary of the remained parent space is doubled in the X_1 direction. Let the second sampling point ($\mathbf{x}^2_{c,i}$) be in the center of the second subspace ($\mathcal{S}^2_{c,i}$) in the X_1 direction. The second subspace including the second sample (the dashed-line rectangle in right side of Fig.3.7). The last parent space is the third subspace and uses SQP and the Kriging model to sample the third point (the red hollow star, $\mathbf{x}^3_{c,i}$ on the right side of Fig.3.7). Subspace optimum and subspace division are finished, the parent space is divided into three subspaces, each having one sample.

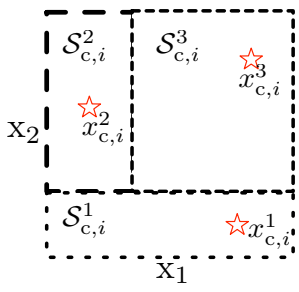


Figure 3.8: Three subspaces of S.A. DIRECT.

Figure 3.8 shows the three subspaces. One parent space produces three subspaces, each having a sampling point (three types of line represent three different and independent subspaces). S.A.

DIRECT does not trisect at one time and does not sample directly the center of a subspace. S.A. DIRECT samples at first and then let the sample be in the center and subspace is divided from parent space. S.A. DIRECT uses the location of sampling points (according to the minimum distance between the sample and the boundary of parent space, and its direction) to decide the subspace. Sampling the first sample from the parent space, the parent space is divided into the first subspace including the first sample and remained parent space. Sampling the second sample from the remained parent space, the remained parent space is divided into the second subspace including the second sample and third subspace. Last, sampling the third sample from the third subspace. The subspace is divided one after another. So the longest diagonal of the three subspaces are often different.

3.4.3 Extension to Multi-dimensional problems

The algorithm is extended from two to N variables as follows:

1. Select parent spaces from all subspaces using the $f-d$ filter and the $f-g$ filter (design space is the parent space at the beginning).
2. Build the Kriging model in the design space using all samples (at the beginning, there is no sampling points, so the random samples are used).
3. Every parent space is used to sample three sampling points and is divided into three subspaces by the subspace optimum.
   (a) Sample the first point in the parent space using the subspace optimum.
   (b) According to the location of the first sample in the parent space, find the minimum distance that divides parent space into two parts. One part includes the first sampling point (the first subspace), and the other part (remained parent space) is used to sample the second and third sampling points and is divided into the second and third subspaces.
   (c) Sample the second point in the remained space using the subspace optimum.
   (d) According to the location of the second sampling point, find the minimum distance, and divide the remained parent into two parts. One includes the second sampling

point (the second subspace), and the other is the third subspace. Sample the third sampling point using the subspace optimum.

4. Update all subspaces.

5. Compare all feasible samples. The best is the result.

6. Calculate the total number of sampling points. If the total is over the terminal condition, S.A. DIRECT is stopped. Otherwise, return to step1.

3.5 Special cases

Subspaces are produced by the minimum distance between sampling points and parent space. However, the parent space cannot be divided in some cases. These cases include a sampling point on the boundary ($\min(lb_i) = 0$ or $\min(ub_i) = 0$), corner ($\min(lb_i) = \min(ub_i) = 0$), or median ($\min(lb_i) = \min(ub_i) \neq 0$) of the parent space. In these cases, S.A. DIRECT cannot divide the parent space using original method. In order to treat these cases, S.A. DIRECT bisects or trisects the parent space at one time. S.A. DIRECT selects bisection or trisection as follow :

- Figure 3.9 shows special case 1. The first sampling point is on the boundary, corner, or median of the parent space. In the Fig.3.10, S. A. DIRECT trisects the parent space. From each of the three equal subspaces, one sampling point is obtained using the subspace optimum.

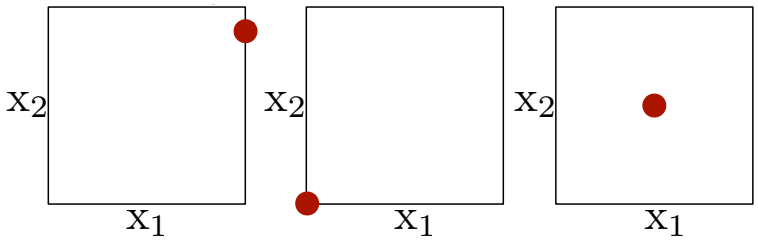


Figure 3.9: Special case 1.

- Figure 3.11 shows the special case 2. The second sample is on the boundary, corner, or median of the parent space. In Fig.3.12, S.A. DIRECT bisects the parent space. From

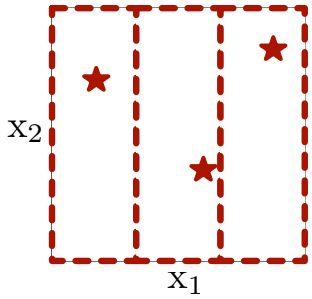


Figure 3.10: Cutting and optimization for special case 1.

each of two equal subspaces, one sampling point is obtained using the subspace optimum.

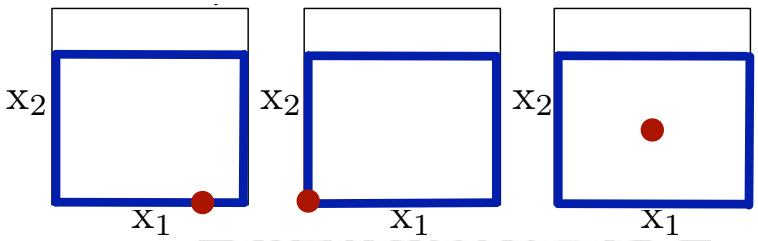


Figure 3.11: Special case 2.

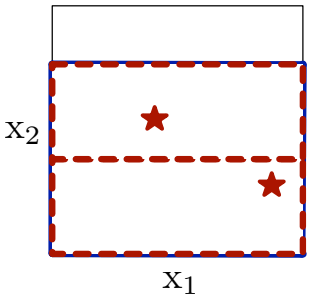
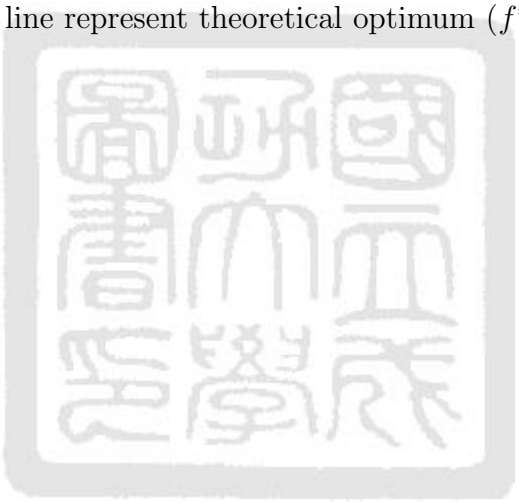


Figure 3.12: Cutting and optimization for special case 2.

- The third sample on the boundary, corner, or median of the parent space is acceptable. The third sampling point represents the third subspace.

Chapter 4 Numerical Comparisons

This chapter presents numerical comparisons of bound constrained problems and inequality constrained problems. Building the Kriging model requires some sampling points. In order to ensure the accuracy of the response surface, the number of initial sampling points is adjusted according to the number of design variables. Practical engineering problems do not allow too many samples. All test problems use eleven initial sampling points. The eleven points are selected at random, because the trend of problems (objection functions) may be uncertain. The DIRECT algorithm (gclSolve) is used in these numerical comparisons. All comparison figure, x axis is function count, y axis is function value, hollow circle line represent S.A. DIRECT, start line represent DIRECT and dash line represent theoretical optimum (f^*).



4.1 Bounds constrained problems

Table 4.1 lists about the properties of the test bound constrained problems. The following present each equation and each comparison plot.

Test function	Abbreviation	Number of dimensions	Number of local minima	Number of global minima
Shekel 5	S5	4	5	1
Shekel 7	S7	4	7	1
Shekel 10	S10	4	10	1
Hartman 3	H3	3	4	1
Hartman 6	H6	6	4	1
Brain RCOS	BR	2	3	3
Goldstein and Price	GP	2	4	1
Six-Hump Camel	C6	2	6	2
Two-Dimensional Shubert	SHU	2	760	18

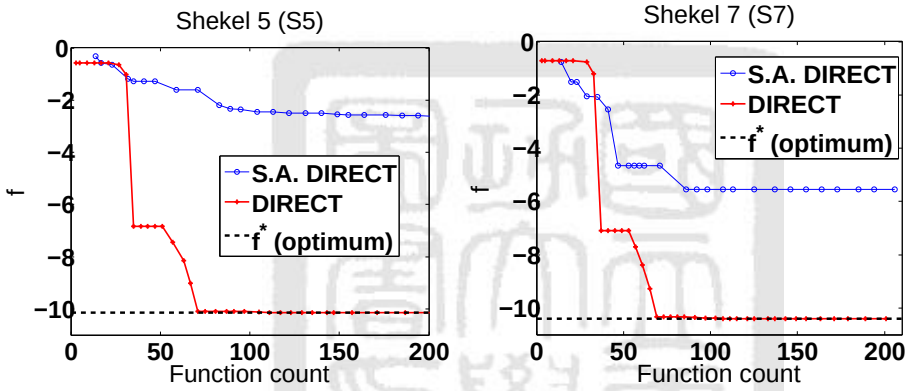
Table 4.1: Properties of boundary constraint problems [14].

Objection function of Shekel 5 $(m = 5)$**, Shekel 7** $(m = 7)$**, and Shekel 10** $(m = 10)$**:**

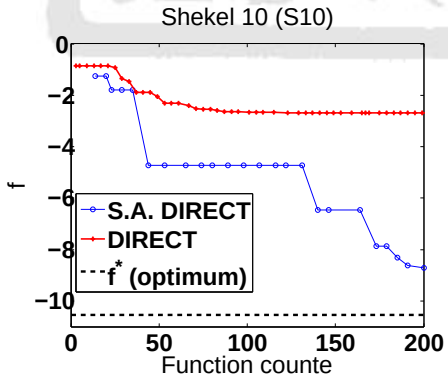
$$S_{4,m}(x_1, x_2, x_3, x_4) = -\sum_{j=1}^{m}[\sum_{i=1}^{4}(x_i - C_{ij})^2 + \beta_j]^{-1} \quad 0 \le x_1, x_2, x_3, x_4 \le 10$$

$$\beta = \frac{1}{10}[1, 2, 2, 4, 4, 6, 3, 7, 5, 5]^T$$

$$C = \begin{bmatrix} 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 5 & 1 & 2 & 3.6 \\ 4 & 1 & 8 & 6 & 3 & 2 & 3 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 3 & 1 & 2 & 3.6 \end{bmatrix}$$



(a) Shekel 5 $(f^* = -10.1532, m = 5)$ (b) Shekel 7 $(f^* = -10.4029, m = 7)$



(c) Shekel 10 $(f^* = -10.5364, m = 10)$

Figure 4.1: Results for Shekel 5,7,10.

In Fig.4.1(a) and (b), the function counts (samples) between 30 and 40, S.A. DIRECT outperforms DIRECT. In Fig.4.1(c), S.A. DIRECT outperforms DIRECT.

Objective function of Hartman 3:

$$f(x) = H_{3,4}(x_1, x_2, x_3) = -\sum_{i=1}^{4} \alpha_i \exp[-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2] \quad 0 \le x_1, x_2, x_3 \le 1$$

$$\alpha = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix} \quad A = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix} \quad P = 10^{-4} \begin{bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{bmatrix}$$

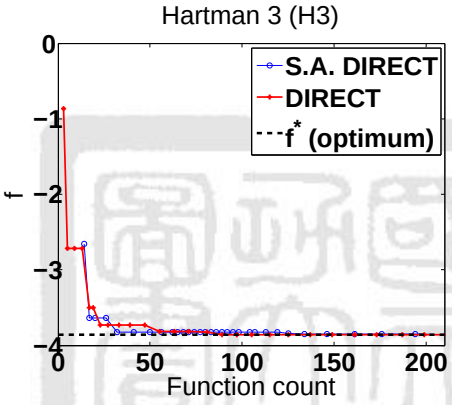


Figure 4.2: Result of Hartman 3 ($f^* = -3.86278$).

In Fig.4.2, the function counts (samples) between 15 and 20, S.A. DIRECT outperforms DIRECT.

Objective function of Hartman 6:

$$f(x) = H_{6,4}(x_1, x_2, x_3, x_4, x_5, x_6) = -\sum_{i=1}^{4} \alpha_i \exp[-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2] \quad 0 \le x_1, x_2, x_3, x_4, x_5, x_6 \le 1$$

$$\alpha = [1, 1.2, 3, 3.2]^T, A = \begin{bmatrix} 10 & 3 & 17 & 3.05 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$$

$$P = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 830 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$$

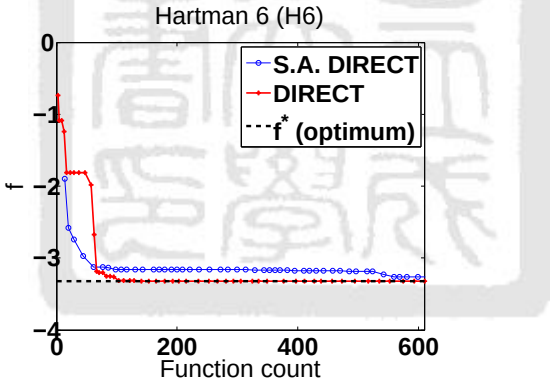


Figure 4.3: Result of Hartman 6 ($f^* = -3.3237$).

In Fig.4.3, the function counts (samples) is less than about 80, S.A. DIRECT outperforms DIRECT.

Objective function of Brain RCOS:

$$f(x_1, x_2) = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e \quad -5 \leq x_1 \leq 10, 0 \leq x_2 \leq 15$$

$$a = 1, b = \frac{5.1}{4\pi^2}, c = \frac{5}{\pi}, d = 6, e = 10, f = \frac{1}{8\pi}$$

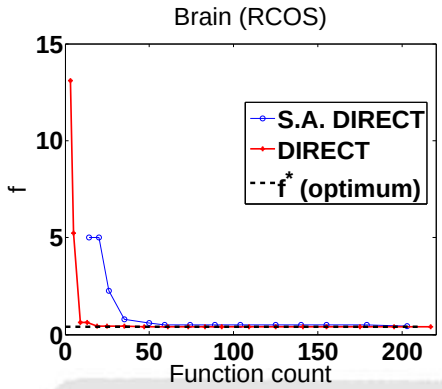


Figure 4.4: Result of RCOS ($f^* = 0.397887$).

In Fig.4.4, DIRECT outperforms S.A. DIRECT.

Objective function of Goldstein and Price:

$$\begin{aligned} f(x_1, x_2) \quad = \quad & (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \\ & (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)) \\ & -2 \leq x_1, x_2 \leq 2, \end{aligned}$$

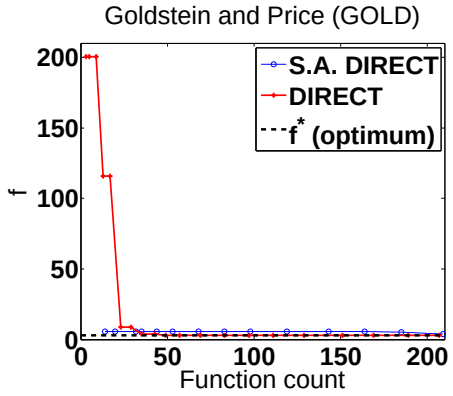


Figure 4.5: Result of GP ($f^* = 3$).

In Fig.4.5, S.A. DIRECT outperforms DIRECT.

Objective function of Six-Hump Camel:

$$f(x_1, x_2) = (4 - 2.1x_1^2 + x_1^{4/3})x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \quad -3 \le x_1 \le 3, -2 \le x_2 \le 2$$

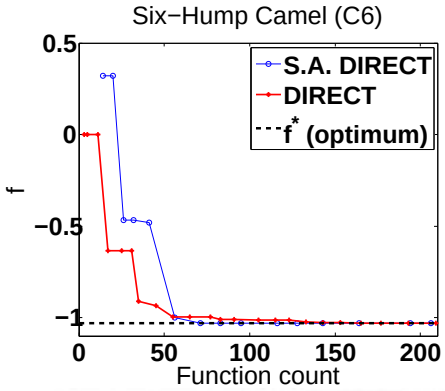


Figure 4.6: Result of C6 ($f^* = -1.0316$).

In Fig.4.6, the function count between 60 and 130 function counts (samples), S.A. DIRECT outperforms DIRECT.

Objective function of Two-Dimensional Shubert:

$$f(x_1, x_2) = \sum_{j=1}^{5} j\cos[(j+1)x_1 + j] \times \sum_{j=1}^{5} j\cos[(j+1)x_2 + j] \quad -10 \le x_1, x_2 \le 10$$

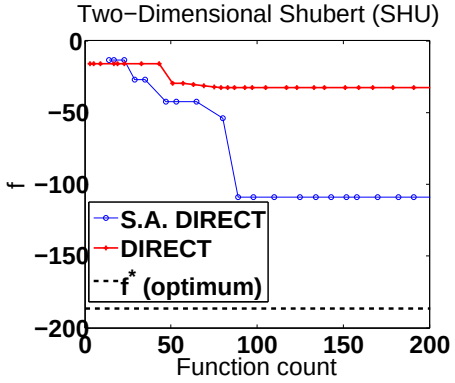


Figure 4.7: Result of SHU ($f^* = -186.73067$).

In Fig.4.7, S.A. DIRECT outperforms DIRECT.

Test function	Number of dimensions	Number of constraint	Function value of global minima
G2	20	2	-0.803619
G4	5	6	-30665.539
G6	2	2	-6961.81388
G7	10	8	24.3062091
G9	7	4	680.6300573
G10	8	6	70493307

Table 4.2: Property of inequality constraint problems [26]

4.2 Inequality constrained problems

Table 4.2 lists the tested inequality constraint problems. The results for each problem are given below.

Problem of G2:

$$\begin{aligned} \max \quad & f(x) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sum_{i=1}^{n} i x_i^2} \right| \\ \text{s.t.} \quad & g_1(x) = -\prod_{i=1}^{n} x_i + 0.75 \leq 0, \\ & g_2(x) = -\sum_{i=1}^{n} x_i - 7.5n \leq 0 \\ & 0 \leq x_i \leq 10, i = 1, 2, \cdots, 20 \end{aligned}$$

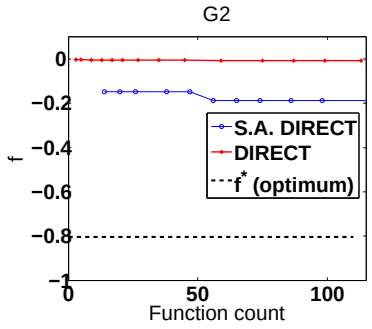


Figure 4.8: Result of G2 ($f^* = -0.803619$).

In Fig.4.8, S.A. DIRECT outperforms DIRECT.

Problem of G4:

$$
\begin{aligned}
\min \quad & f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 4.0792.141 \\
\text{s.t.} \quad & g_1(x) = u(x) - 92 \leq 0, \\
& g_2(x) = -u(x) \leq 0, \\
& g_3(x) = v(x) - 110 \leq 0, \\
& g_4(x) = -v(x) + 90 \leq 0, \\
& g_5(x) = w(x) - 25 \leq 0, \\
& g_6(x) = -w(x) + 20 \leq 0,
\end{aligned}
$$

$$
\begin{aligned}
u(x) = \quad & 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \\
v(x) = \quad & 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \\
w(x) = \quad & 9.300961 + 0.00470263x_5 + 0.0012547x_1x_3 + 0.00195085x_3x_4 \\
& l_i \leq x_i \leq u_i, i = 1, \cdots, 5; l = (78, 33, 27, 27, 27), u = (102, 45, 45, 45, 45)
\end{aligned}
$$

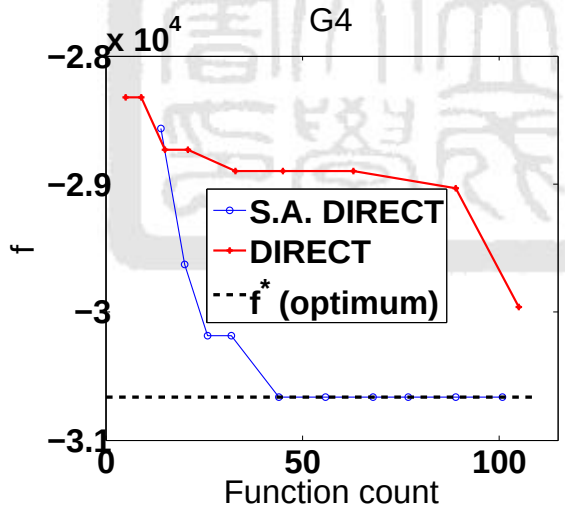


Figure 4.9: Result of G4 ($f^* = -30665.539$).

In Fig.4.9, S.A. DIRECT outperforms DIRECT.

Problem of G6:

$$\begin{aligned} \min \quad & f(x) = (x_1 - 10)^3 + (x_x - 20)^3 \\ \text{s.t.} \quad & g_1(x) = -[(x_1 - 5)^2 + (x_2 - 5)^2] + 100 \leq 0, \\ & g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0 \end{aligned}$$

$$l \leq x_i \leq 100, i = 1, 2; l = (13, 0)$$

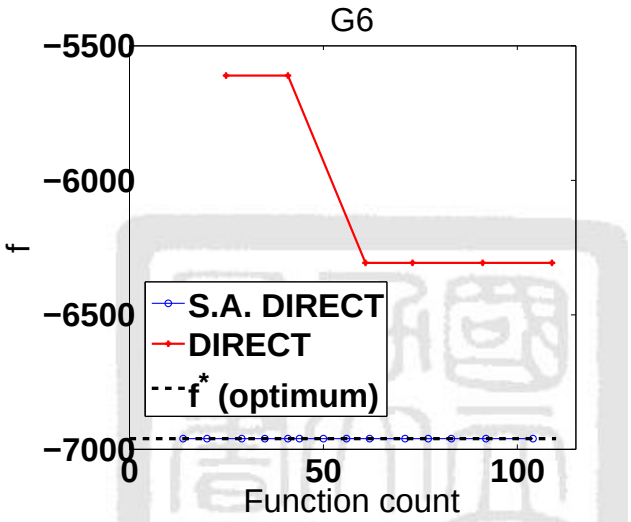


Figure 4.10: Result of G6 ($f^* = -6961.81388$).

In Fig.4.9, S.A. DIRECT outperforms DIRECT.

Problem of G7:

$$
\begin{aligned}
\min \quad & f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\
& +2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \\
\text{s.t.} \quad & g_1(x) = 4x_1 + 5x_2 - 3x_7 + 9x_8 - 105 \leq 0, \\
& g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0, \\
& g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0, \\
& g_4(x) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0, \\
& g_5(x) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0, \\
& g_6(x) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0, \\
& g_7(x) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0, \\
& g_8(x) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0 \\
& -10 \leq x_i \leq 10, i = 1, 2, \cdots, 10
\end{aligned}
$$

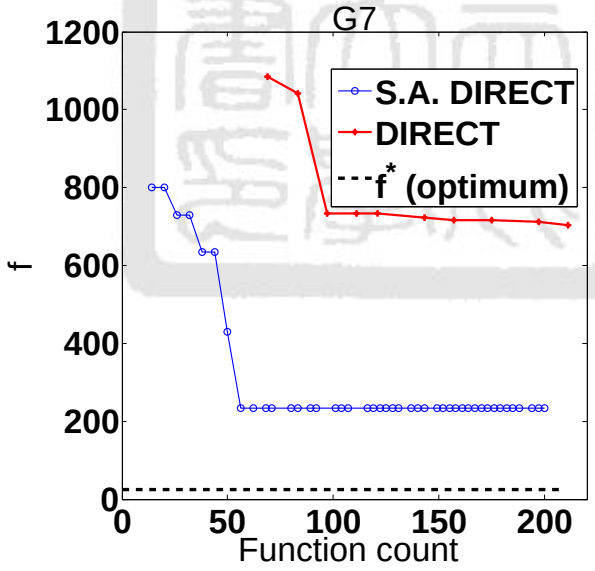


Figure 4.11: Result of G7 ($f^* = 24.3062091$).

In Fig.4.11, S.A. DIRECT outperforms DIRECT.

Problem of G9:

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

$$-10 \leq x_i \leq 10, i = 1, 2, \cdots, 7$$

$$x^* = (2.330499, 1.951372, -0.4775414, 4.365726, ...$$

$$-0.6244870, 1.038131, 1.594227)$$

$$f^* = 680.6300573$$

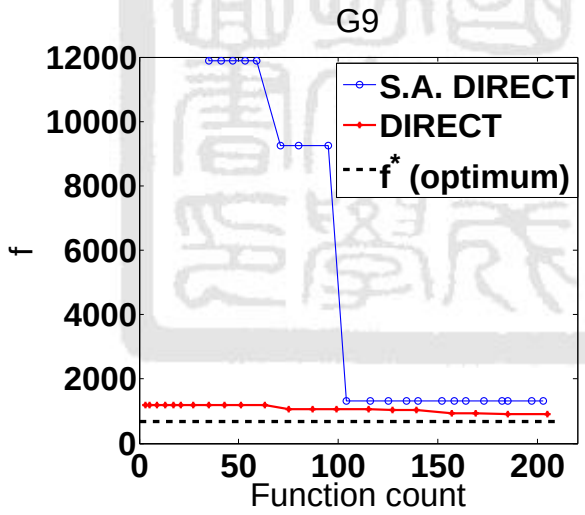


Figure 4.12: Results of G9 ($f^* = 680.6300573$).

In Fig.4.12, DIRECT outperforms S.A. DIRECT.

Problem of G10:

$$
\begin{aligned}
\min \quad & f(x) = x_1 + x_2 + x_3 \\
\text{s.t.} \quad & g_1(x) = -1 + 0.0025(x_4 + x_6) \leq 0, \\
& g_2(x) = -1 + 0.0025(-x_4 + x_5 + x_7) \leq 0 \\
& g_3(x) = -1 + 0.01(-x_5 + x_8) \leq 0 \\
& g_4(x) = 100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \leq 0 \\
& g_5(x) = x_2x_4 - x_2x_7 - 1250x_4 + x_1250x_5 \leq 0 \\
& g_6(x) = x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \leq 0
\end{aligned}
$$

$$l_i \leq x_i \leq u_i, i = 1, 2, \cdots, 8; l = 10 \times (10, 100, 100, 1, 1, 1, 1, 1), u = 1000 \times (10, 10, 10, 1, 1, 1, 1, 1)$$

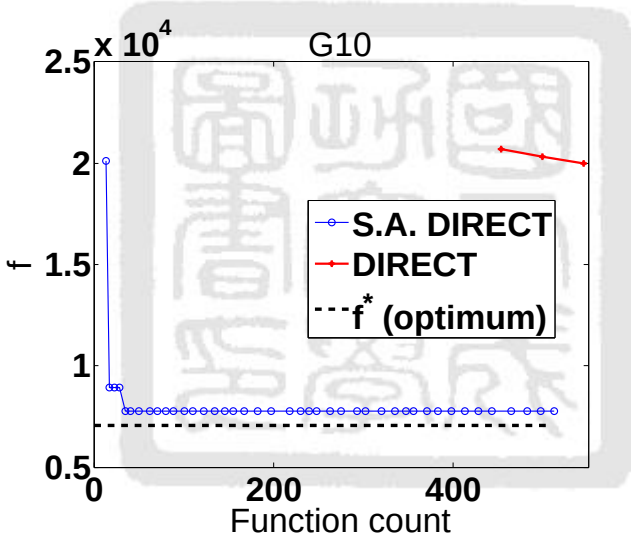


Figure 4.13: Result of G10 ($f^* = 70493307$).

In Fig.4.13, S.A. DIRECT outperforms DIRECT.

4.3 Discussion

Section 4.1 shows that S.A. DIRECT gives performance improvements with few function evaluations. Section 4.2 shows that S.A. DIRECT outperforms DIRECT, besides function G9. S.A. DIRECT does not sample the optimum in some case. The results does not improve further, such as function G9, S.A. DIRECT traps the local. In Fig.4.14, S.A. DIRECT* are the

parent space become to design space per five iterations (loops) and the subspaces back to the beginning. S.A. DIRECT* outperforms S.A. DIRECT. From the numerical comparisons S.A. DIRECT has better outputs in inequality constraint problems.

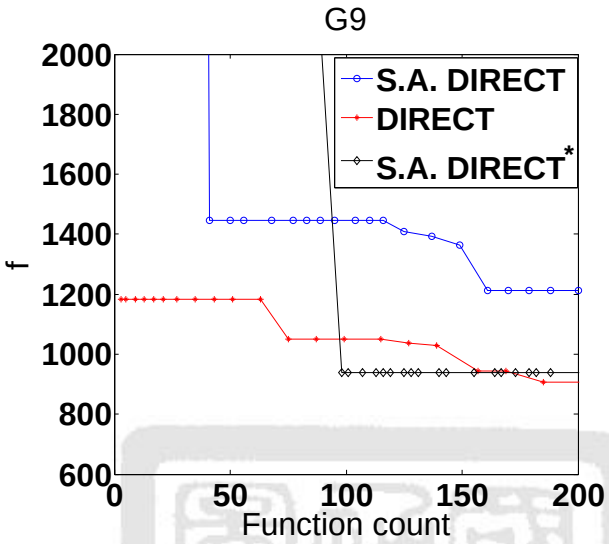


Figure 4.14: Result of G9* ($f^* = 680.6300573$).

Chapter 5 Engineering case study : design of a belt-pulley mechanism

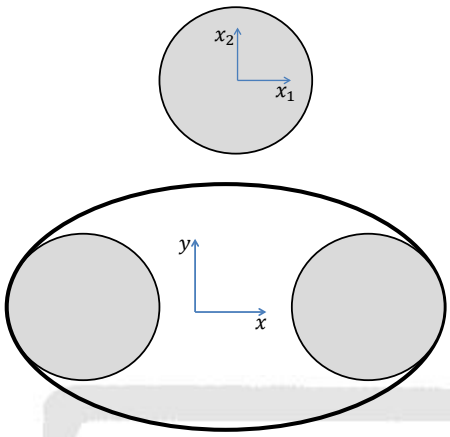


Figure 5.1: Belt-tensioner-pulley system.

Consider a belt-tensioner-pulley system with two pulleys and one steel-cord-reinforced belt with six V-shaped grooves. The radius of both pulleys is 25 cm and the thickness is 15 cm. The original inner and outer radii of belt are 54 and 55.5 cm, respectively. Adding a tensioner (radius: 25 cm; thickness: 15 cm) provides enough tension within the system. Figure 5.1 shows a diagram of the system. The left pulley, right pulley, and tensioner (initial) are centered at $(-29, 0)$, $(57.2, 0)$, and $(14, 100)$, respectively. Due to the geometric constraints Eq.5.1 of the system, locations of the pulleys and the tensioner cannot be arbitrarily determined. The design variables is the tensioner displacement, (x_1, x_2). To design the location of the tensioner has the minimum stress (tension). Objective function is time-consuming, so the terminal condition is set by thirty function counter.

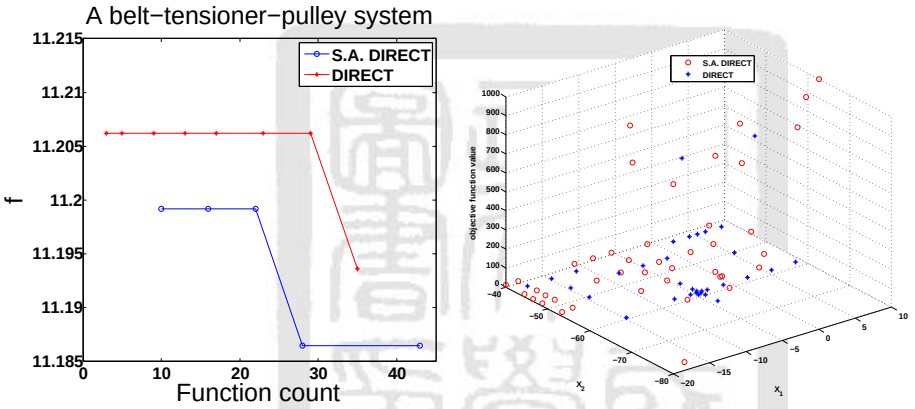
$$
\begin{aligned}
& f = \min\ (belt\ stress) \\
& (-20, -80) \leq x_1, x_2 \leq (10, -40) \\
\text{s.t.}\quad & g_1(x) = 50^2 - [(43 + x_1)^2 + (100 + x_2)^2]^2 < 0 \\
& g_2(x) = 50^2 - [(-43.2 + x_1)^2 + (100 + x_2)^2]^2 < 0
\end{aligned}
\tag{5.1}
$$

Table 5.1 presents seven initial samples (random), the number of samples every iteration, no. of samples, samples data (objective function value (f), the longest diagonal (d), and lower

7 initial samples $(x_1, x_2, f) = (4.44, -58.05, 23.8440), (7.17, -54.43, 19.5466), (7.40, -40.85, 11.1991)$ (-1.03,-40.70,11.3224),(-17.46,78.12,11.2062),(-1.73,-61.70,28.2976),(-7.35,-61.40,29.5115)					
iteration	sample			subspace	filtered no.
	no.	(x_1, x_2)	(f, d)	$(lb), (bu)$	
1(3)	8	(0.10,-74.65)	(1000,16.41)	(-20,-80),(10,-66.67)	8,10
	9	(-20,-53.33)	(20.9277,16.41)	(-20,-66.67),(10,-53.33)	
	10	(-20,-40)	(11.2112,16.41)	(-20,-53.33),(10,40)	
2(6)	11	(-11.50,-46.88)	(12.0209,10.80)	(-20,-53.33),(-3.00,-40)	9,11
	12	(-3.00,-50.13)	(14.9683,7.24)	(-3.00,-5.33),(10,-46.93)	
	13	(-3.00,-46.93)	(12.2114,7.36)	(-3.00,-46.93),(10,-40)	
	14	(5.93,-66.67)	(1000,7.81)	(1.84,-80),(10,-66.67)	
	15	(-20,-80)	(1000,8.62)	(-20,-80),(-9.07,-66.67)	
	16	(-4.22,-68.44)	(1000,8.62)	(-9.07,-80),(1.86,-66.67)	
3(6)	17	(-10.65,-40)	(11.3100,10.15)	(-18.31,-53.33),(-3.00,-40)	17,18
	18	(-20,-46.47)	(11.2032,6.53)	(-20,-52.94),(-18.31,-40)	
	19	(-18.31,-52.94)	(19.2137,0.87)	(-20,-53.33),(-18.31,-52.94)	
	20	(-14.04,-59.24)	(1000,8.33)	(-20,-66.67),(-10,-53.33)	
	21	(-9.62,-54.24)	(19.5599,8.33)	(-10,-66.67),(0,-53.33)	
	22	(10,-62.61)	(1000,8.33)	(10,-66.67),(10,-53.33)	
4(6)	23	(-18.31,-48.63)	(13.34,2.32)	(-20,-52.94),(-18.31,-48.63)	13,15,21,
	24	(-18.31,-44.31)	(11.1865,2.32)	(-20,-48.63),(-18.31,-44.31)	24,26
	25	(-18.31,-40)	(11.2515,2.32)	(-20,-44.31),(-18.31,-40)	
	26	(-5.52,-40)	(11.2013,7.13)	(-8.05,-53.33),(-3.00,-40)	
	27	(-8.05,-46.67)	(11.9315,6.12)	(-18.31,-53.33),(-8.05,-46.67)	
	28	(-8.05,-40)	(11.2203,6.12)	(-18.31,-46.67),(-8.05,-40)	
5(15)	29	(-20,-48.63)	(13.3064,11.11)	(-20,-48.63),(-18.31,-47.19)	
	30	(-18.31,-46.42)	(11.3101,11.11)	(-20,-47.19),(-18.31,-45.75)	
	31	(-20,-44.31)	(11.2994,11.11)	(-20,-45.75),(-18.31,-44.31)	
	32	(-5.53,-53.33)	(18.2324,3.36)	(-8.05,-53.33),(-3.00,-48.89)	
	33	(6.02,-48.89)	(13.8488,3.36)	(-8.05,-48.89),(-3.00,-44.44)	
	34	(-5.24,-43.60)	(11.3181,3.36)	(-8.05,-44.44),(-3.00,-40)	
	35	(1.34,-46.93)	(12.2189,4.09)	(-3.00,-46.93),(1.34,-40)	
	36	(4.67,-46.67)	(11.9889,4.09)	(1.34,-46.93),(5.67,-40)	
	37	(10,-46.45)	(11.7221,4.09)	(5.67,-46.93),(10,-40)	
	38	(0,-55.30)	(20.2198,5.37)	(-10,-57.26),(0,-53.33)	
	39	(-0.26,-57.26)	(22.5149,4.71)	(-0.53,-66.67),(0,-57.26)	
	40	(-0.53,-57.26)	(22.5107,6.67)	(-10,-66.67),(-0.53,-57.26)	
	41	(-10.57,-80)	(1000,5.90)	(-20,-80),(-9.07,-75.56)	
	42	(-11.73,-75.5556)	(1000,5.90)	(-20,-75.56),(-9.07,-71.11)	
	43	(-20,-70.2293)	(1000,5.90)	(-20,-71.11),(-9.07,-66.67)	

Table 5.1: S.A. DIRECT

boundary (lb) and upper boundary (lb) of subspace), and selected parent spaces (filtered no. of samples) at the first four iteration. Design space is decided form (-20,-80) to (10,-40) by seven initial samples and objective function convergence. Beginning, the Kriging model is built with seven initial samples. The design space is the initial parent space, it is divided into three subspaces, (-20,-80) to (10,-66.67), (-20,-66.67) to (10,-53.33), and (-20,-53.33) to (10,40), including samples, (0.1,-74.65), (-20,-53.33), and (-20,-40), respectively. Subspaces no.8 and no.10 are selected parent spaces using $f-d$ filter and $f-g$ filter. At the second iteration, the two parent spaces are divided into three subspaces, respectively.... Similarly, the iteration 2 to 5, all relatively data, samples (no. (x_1, x_2), and (f, d)) and boundary of subspaces, present in table 5.1.



(a) Optimal results. (b) Samples data.

Figure 5.2: The results of engineering problem.

Engineering case study, the objective function is simulated by ANSYS. The analysis is time-consuming and the optimum is unknown, so the function counters (samples) are limited about thirty and the result only presents which one is better. Figure 5.2 (a) shows that S.A. DIRECT outperforms DIRECT. Figure 5.2 (b) shows the all samples with DIRECT (*) and S.A. DIRECT (o). S.A. DIRECT finds the location, $(-8.04799, -46.66667)$, that has 11.1865 MPa. DIRECT finds the location, $(-18.33333, -42.22222)$, that has 11.1954 MPa.

Chapter 6 Conclusions and Future Work

6.1 Conclusions

The goal of this thesis is to provide a systematic algorithm for engineering problems with time-consuming and tight time constraint, such that the design of a belt-tensioner-pulley system can be acceptable. S.A. DIRECT uses trade-off Lipschitz constant, local (f) versus global characteristics (d), to select parent spaces ($f-d$ filter). S.A. DIRECT considers both of local and global searching. When the problems have inequality constraints, S.A. DIRECT uses most influence constraint (the maximum violation value) and objective function values to filter parent spaces ($f-d$ filter). S.A. DIRECT can also in high-dimensional design variables because of both division and sampling. S.A. DIRECT is an effective optimizer in relatively few function counters, especially the problems with inequality constraints and the time-consuming practical problems.

The specific contributions of this thesis are summarized into four main points:

1. **New subspaces division:** This thesis used new subspaces division, the first two subspaces based on the location between the sample and its parent space. Chose the minimum distance and doubled the minimum distance to divide the parent space into two parts. Then the sample was in the center of its subspace along the cutting direction. The remaining parent subspace sampled the third sample.

2. **Separate the objective function and inequality constraints:** DIRECT selected the parent spaces using penalty function or neglected the infeasible subspaces, this thesis selected parent spaces had two steps, the first selected parent spaces using the $f-d$ filter (neglected constraints) and the second filtered current parent space using the $f-g$ filter (considered the objective function and constraint function values of each parent space).

3. $f-g$ **filter:** This thesis compared the relation of the objective function and the most influence constraint (he maximum violation value of inequality constraint). Relaxing the constraints filtered the parent spaces.

4. **Combined the Kriging model and SQP in optimum algorithm:** To deal with the optima on corners, boundaries, or vertices, this thesis sampled form the Kriging model using SQP.

6.2 Future Work

The following research activities deserve much in-depth investigation in the future:

1. Deal with initial sample. In this thesis, initial samples are radon. Getting the initial samples in other method may have improved accuracy in response surface.
2. Take different $f - g$ filter to filter parent spaces or different method in dealing with the constraints. This thesis consider the single constraint in filtering parent spaces.
3. Take different subspaces division. Different dividing may have different size of subspaces and different parent spaces may be selected from all subspaces.
4. Take different response surface. The Kriging model is one of response surfaces, the other surfaces may have different situations.
5. Take different sampling method, this thesis samples using SQP. SQP is a local optimum algorithm. The different sampling method may have different outcomes.
6. Deal with problems with equality constraints, the thesis does not propose it.

References

[1] L.T. Watson and C.A. Baker. A fully distributed parallel global search algorithm. *Engineering Computations*, 18:155–169, 2001.

[2] S. E. Cox, R. T. Haftka, C. A. Baker, B. Grossman, W. H. Mason, and L. T. Watson. A comparison of global optimization methods for the design of a high-speed civil transport. *Journal of Global Optimization*, 21:415–433, 2001.

[3] Z. Mehel-Saodo, S. Bourennane, and C. Fossati. Buried object localization using DIRECT algorithm. In *Proceedings of the 5th IEEE Sensor Array and Multichannel Signal Processing Workshop*, Darmstadt, Germany, 17-20 July 2005.

[4] J.D. Griffin and T.G. Kolda. Asynchronous parallel hybrid optimization combing DIRECT and GSS. *Optimization Methods and Software*, 25:719–817, 2010.

[5] W. Lin, N. Kovvali, and L. Carin. DIRECT algorithm for computation of derivatives of the daubechies basis function. *Applied Mathematics and Computation*, 170:1006–1013, 2005.

[6] J. He, A. Verstak, L. T. Watson, and M. Sosonkina. *Performance Modeling and Analysis of a Massively Parallel DIRECT-Part 1.* Technical report TR-07-01, Department of Computer Science, Virginia Polytechnic Institute and State University,Blacksburg, 2007.

[7] J. He, A. Verstak, L. T. Watson, and M. Sosonkina. Performance modeling and analysis of a massively parallel DIRECT-part 2. *International Journal of High Performance Computing Applications*, 23:29–41, 2009.

[8] Y. Zhang, F. Sun, and H. He. Control strategy optimization for hybrid electric vehicle based on DIRECT algorithm. *IEEE Vehicle Power and Propulsion Conference*, 2008.

[9] Y. Chang, K. Hung, and S. Lee. Human face detection with neural networks and the DIRECT algorithm. *Artificial Life and Robotics*, 12(1):112–115, 2008.

[10] M. C. Bartholomew-Biggs, S. C. Parkhurst, and S. P. Wilson. Using DIRECT to solve an aircraft routing problem. *Computational Optimization and Applications*, 21(3):311–323, 2002.

[11] D.R. Jones. *The DIRECT global optimization algorithm*, volume 1. Kluwer Academic Publishers, Dordrecht, The Netherlands, the encyclopedia of optimization edition, 2001.

[12] H. Zhu and D. B. Bogy. Hard disc drive air bearing design:modified DIRECT algorithm and its application to slider air bearing surface optimization. *Tribology International*, 37(2):193–201, 2004.

[13] J. M. Gablonsky. *Modifications of The DIRECT Algorithm*. PhD thesis, Department of Mathematics in North Carolina State University, 2001.

[14] D.R. Jones, C.D. Perttunen, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Global Optimization*, 79(1):157–181, 1993.

[15] D.E. Finkel and C.T. Keley. An adaptive restart implementation of DIRECT. Technical Report CRSE-TR04-30, Center for Research in Scientific Computation, North Carolina State University, Raleigh,NC, 2004.

[16] D.E. Finkel and C.T. Keley. Additive scaling and the DIRECT algorithm. *Journal of Global Optimization*, 36:597–608, 2006.

[17] O. Liu. Linear scaling and the DIRECT algorithm. *Journal of Global Optimization*, 2012.

[18] N. Goldberg, T. G. Kolda, and A. S. Yoshimura. Concurrent optimization with duet:DIRECT using external trial points. Technical Report TSAND2008-5844, Sandia National Laboratories, Livermore, CA, 2008.

[19] Y. D. Sergeyev and D. E. Kvasov. Global search based on efficient diagonal partitions and a set of lipschitz constant. *Society for Industrial and Applied Mathematics*, 16(3):910–937, 2006.

[20] J.M. Gablonsky and C.T. Kelley. A locally-biased form of the direct algorithm. *Journal of Global Optimization*, 21:27–37, 2001.

[21] S. E. Cox, R. T. Haftka, C. A. Baker, B. Grossman, W. H. Mason, and L. T. Watson. A comparison of global optimization methods for the design of a high-speedy civil transport. *Journal of Global Optimization*, 21:415–433, 2001.

[22] J. He, L. T. Watson, N. Ramakrishnan, C. A. Shaffer, A. Verstak, J. Jiang, K. Bae, and W. H. Tranter. Dynamic data structures for a DIRECT search algorithm. *Computational Optimization and Applications*, 23:5–25, 2002.

[23] J.S. Arora. *Sequential Quadratic Programming: SQP*, volume 1. Elsevier Academic Press, Dordrecht, The Netherlands, introduction to optimization design edition, 2004.

[24] D. Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52:119–139, 1951.

[25] C. Noel. The origins of kriging. *Mathematical Geology*, 22(3):239–252, 1990.

[26] Test function. http://www-optima.amp.i.kyoto-u.as.jp.

Appendix : List of MATLAB Programs

Main code

```
clc, clear all, close all

%%
%%%   S.A. DIRECT setting   %%%
max_eval = 200;    delta = 1e-4; %%%   max_eval: terminal condition
e = 1;    r = 10;   %%%   the side of parent subspaces limit ratio
lb = -10*ones(1,7);   %%%   lower boundary
ub =  10*ones(1,7);   %%%   upper boundary
Bd = abs(ub-lb);
x_normal = rand(11, 7);   %%% random samples in 0~1
n_Dim = size(x_normal, 2);
for a = 1:n_Dim   %%% transport to original design space
    x(:,a) = x_normal(:,a).*Bd(:,a)+lb(1,a);
end
%%
%%%   DIRECT setting   %%%
GLOBAL.MaxEval = max_eval;
PriLev=1;
A = [];
b_L = [];
b_U = [];
I = [];
c_L = -inf*ones(4,1);
c_U = 0*ones(4,1);
%%
%%%   S.A. DIRECT   %%%
tic
%saD(@objective function, @constraint function, lower boundary,...
%     upper boundary, sides_ratio, max_function counter, inital samples);
```

```
opt = saD(@PrG9f, @PrG9c, lb, ub, e, max_eval, x); % with constraint
% opt = saD(@hart3, @nonlcon_11, lb, ub, e, max_eval, x);
% without constraint
toc
%%
%%%   DIRECT   %%%
ticxopt = gcl(@PrG9f, @PrG9c, lb, ub, A, b_L, b_U, c_L, c_U, I,...
    GLOBAL, PriLev);   % with constriant
% xopt = gcl(@hart3, [], lb, ub, [], [], [], [], [], [], GLOBAL, PriLev);
% without constraint
toc
%%
%%%   plot   %%%
a = 680.6300573;% local or global minimum
figure(1)
plot(opt.eval, opt.f_opt,'bo-',xopt.n_point, xopt.f_m,'r*-',...
    [0 max_eval+10], [a a], 'g');hold on;title('function : G7')
legend('S.A. DIRECT','DIRECT','f^*'),xlabel('function count'),ylabel('f')
```

S.A. DIRECT

```
function output = saD(p_f, p_c, lb, ub, e, max_eval, x)
% p_f: objective function, p_c: constraint function,
% lb: lower bournder, ub: upper boundary , e, max_eval, x: initial
   samples
output.x = x;
design_space_lb = lb;
design_space_ub = ub;
n_Dim = size(x,2);
for i = 1:size(x,1)
    output.f(i) = feval(p_f,output.x(i,:));
    %%%   calculates the objective function values of all random samples
    [gg, geq] = feval(p_c,output.x(i,:));
```

```
    %%%  calculates the constraint function values of all random samples
     output.g(i)=max(0,max(gg));
    %%%  constraint g <=0: feasible, ortherwise infeasible
end
fittype = 1;  SCFtype = 1;  %%%  Kriging model parameters
n_fdg = 1;k = 1;f_d = [];fg_i = size(output.x,1);
%%%  n_fdg: number of parent space; f_d: subspace data,
%%%  [d,f_opt,g_opt',children_space_lb,children_space_ub]
output.f_opt= [];output.x_opt= [];output.eval = [];
zero_error = 1e-6;   %%%   constraints tolerance
while fg_i < max_eval
    %%
        for i = 1:n_fdg
            kparam = f_variogram_fit(output.x, output.f',...
                lb, ub, fittype, SCFtype);    %%% kriging parameters
            parent_space_lb = design_space_lb(i,:);
            parent_space_ub = design_space_ub(i,:);
            [ children_space_lb, children_space_ub, x_opt, d ] = ...
                sub_saDD_K(p_c, parent_space_lb, parent_space_ub,...
                kparam, e, n_Dim);
            %%%  one parent space divided into 3 child space
            for n_f_opt=1:size(x_opt,1)
                f_opt(n_f_opt,1) = feval(p_f, x_opt(n_f_opt,:));
            end
            output.x = [output.x;x_opt];
            output.f = [output.f,f_opt'];
            for ii = 1:size(x_opt,1)
                [gg, geq] = feval(p_c,x_opt(ii,:));
                g_opt(ii)=max(0,max(gg));
            end
            output.g = [output.g,g_opt];
            f_d = [f_d;d,f_opt,g_opt',children_space_lb,children_space_ub
    ];
        end
```

```
%% filter 1 f-d plot, noconstrain
        D  = f_d_g_1( f_d );
        D.design = sortrows(unique(D.design,'rows'),[2, 1]);
%% filter 2 f-g plot, constrain
        DD = f_g(D.design);
        DD.design = sortrows(unique(DD.design,'rows'),[2, 1]);
%% next loop search space
        design_space_lb = DD.design(:, 4:3+n_Dim);
        design_space_ub = DD.design(:, end-n_Dim+1:end);
        f_d = []; f_d = [D.other; DD.other];
        n_fdg = size(design_space_ub,1);
        fg_i = size(output.x,1);
%% output data
        feaible_index = find(output.g <= zero_error);
        if ~isempty(feaible_index)
            x_feaible = output.x(feaible_index,:);
            f_feaible = output.f(feaible_index);
            [output.f_opt(end+1), f_opt_index] = min(f_feaible);
            output.x_opt(end+1,:) = x_feaible(f_opt_index,:);
            output.eval(end+1) = fg_i;
            fprintf('\n\n');
            fprintf('\n Iterations this run: %d',k);
            fprintf('Func. Evals this run: %d',fg_i);
            fprintf('f_min:%20.10f',output.f_opt(end));
            fprintf('\n\n');
        else
            [output.f_opt(end+1), f_opt_index] = min(output.f);
            output.x_opt(end+1,:) = output.x(f_opt_index,:);
            output.eval(end+1) = fg_i;
            fprintf('\n\n');
            fprintf('\n Iterations this run: %d', k);
            fprintf('Func. Evals this run: %d',fg_i);
            fprintf('f_min:%20.10f',inf);
            fprintf('\n\n');
```

```
            end
            k = k+1;
end
%%
fprintf('\n\n\n')
fprintf('\n ------------------------------------------------------------ \n');
for i = 1:size(output.f_opt, 2)
fprintf('\n Iterations this run: %d', i);
fprintf('Func. Evals this run: %d',output.eval(i));
fprintf('f_min:%20.10f',output.f_opt(i));
end
fprintf('\n\n');
fprintf(' Result.Solver = ''S.A. DIRECT'' ');
fprintf('\n');
fprintf(' Result.SolverAlgorithm = ''S.A. DIRECT by SoLab of NCKU'' ');
fprintf('\n\n');
end
```

Division and Sample

```
function [ children_space_lb, children_space_ub, x_opt, d ] = sub_saDD_K(
    p_c, parent_space_lb, parent_space_ub, kparam, e, n_Dim)
warning off
f_opt = [];    x_opt = [];    dis_lb = [];    dis_ub = [];
children_space_lb = [];    children_space_ub = [];
zero_error = 1e-5;
d_s_lb(1,:) = parent_space_lb;d_s_ub(1,:) = parent_space_ub;
%d_s_lb,d_s_ub:parent design space 1,2 and 3
e_r = 1/e;
i_k = 6;   %%%   numbers of fmincon
%% fmincon setting
A = [];    B = [];    Aeq = [];    Beq = [];    OPTIONS = optimset('display',
    'off');
```

```
X0 = [];    x_opt_start = [];    f_opt_start = [];
x0x = rand(i_k,n_Dim);
%%
%%%%%%  1st optimal  %%%%%%
x_opt_start = zeros(i_k*2,n_Dim);% otima x using fmincon
f_opt_start = zeros(i_k*2,1);% otima f using fmincon
exitflag = [];
Ff = [];
%% optimum by fmincon
for i = 1:i_k
    X0(i,:) = d_s_lb(end,:)+x0x(i,:).*(d_s_ub(end,:)-d_s_lb(end,:));
    [x_opt_start(i,:) f_opt_start(i) exitflag(i)]...
        = FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
        d_s_lb(end,:), d_s_ub(end,:), p_c, OPTIONS, kparam);
    % f optima is in the kriging mean
    [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)]...
        = FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
        d_s_lb(end,:), d_s_ub(end,:), p_c, OPTIONS, kparam);
    % Kriging model 95% CI
end
%%
%%%  no sample is not over parent space, so fmincon does not consider
%%%  constraints, let sample in parent space
e_f_index = find(exitflag>=1);
if isempty(e_f_index)
    for i = 1:i_k
        X0(i,:) = d_s_lb(end,:)+x0x(i,:).*(d_s_ub(end,:)-d_s_lb(end,:));
        [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = FMINCON(
            @f_predictkrige, X0(i,:), A, B, Aeq, Beq, d_s_lb(end,:),
            d_s_ub(end,:), [], OPTIONS, kparam);    % f optima is in the
            kriging
        [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)] =
           FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq, d_s_lb(end,:),
            d_s_ub(end,:), [], OPTIONS, kparam);
```

```
    end
    e_f_index = find(exitflag>=1);
end
[f_opt(end+1,:), min_index] = min(f_opt_start(e_f_index,:));   x_opt(end
    +1,:) = x_opt_start(e_f_index(min_index),:);
%% measure the distance between the optimum and lower ,upper bound of the
    design space

b = d_s_ub(end,:)-d_s_lb(end,:);
e_index = find((b/max(b))>=e_r);
dis_lb(end+1,:) = ones(1,n_Dim)*inf;   dis_ub(end+1,:) = ones(1,n_Dim)*
    inf;
dis_lb(end,e_index) = abs(x_opt(end,e_index)-d_s_lb(end,e_index));
dis_ub(end,e_index) = abs(d_s_ub(end,e_index)-x_opt(end,e_index));

check_lb = x_opt(end,:)-d_s_lb(end,:);
check_ub = d_s_ub(end,:)-x_opt(end,:);

[L, L_index] = min(dis_lb(end,:));
[U, U_index] = min(dis_ub(end,:));
%%%%   divide the space
%%
 for o = 1
%% special case 1 trisection
    if  L <= zero_error | U <= zero_error %| (L == U & L_index == U_index
        )
        if (L <= zero_error & U <= zero_error) % on the center
            [b_max, b_index] = max(b);   %%% if over 1, choice the first
            cut_index = b_index(1);
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_ub(end+1,:) = d_s_ub(end,:);
            children_space_ub(end, cut_index) =d_s_lb(end, cut_index)...
                +(1/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            children_space_lb(end+1,:) = d_s_lb(end,:);
```

```
        children_space_lb(end, cut_index) = d_s_lb(end, cut_index)
            +...
             (1/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_ub(end+1,:) = d_s_ub(end,:);
        children_space_ub(end, cut_index) = d_s_lb(end, cut_index)...
             +(2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_lb(end, cut_index) = d_s_lb(end, cut_index)...
             +(2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_ub(end+1,:) = d_s_ub(end,:);
    elseif L <= zero_error   % on the  Low boundary
        [b_max, b_index] = max(b);   %%% if over 1, choice the first
        cut_index = b_index(1);
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_ub(end+1,:) = d_s_ub(end,:);
        children_space_ub(end, cut_index) = d_s_lb(end, cut_index)...
             +(1/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_lb(end, cut_index) = d_s_lb(end, cut_index)...
             +(1/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_ub(end+1,:) = d_s_ub(end,:);
        children_space_ub(end, cut_index) = d_s_lb(end, cut_index)...
             +(2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_lb(end, cut_index) = d_s_lb(end, cut_index)...
             +(2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        children_space_ub(end+1,:) = d_s_ub(end,:);
elseif U <= zero_error   % on the Upper boundary
        [b_max, b_index] = max(b);   %%% if over 1, choice the first
        cut_index = b_index(1);
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_ub(end+1,:) = d_s_ub(end,:);
        children_space_ub(end, cut_index) = d_s_lb(end, cut_index)...
             +(1/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
```

```
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_lb(end, cut_index) = d_s_lb(end, cut_index)...
                +(1/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            children_space_ub(end+1,:) = d_s_ub(end,:);
            children_space_ub(end, cut_index) = d_s_lb(end, cut_index)...
                +(2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_lb(end, cut_index) = d_s_lb(end, cut_index)...
                +(2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            children_space_ub(end+1,:) = d_s_ub(end,:);
        end
%% re-optimum form three equal design space
    for j = 1:3
        x_opt_start = zeros(i_k*2,n_Dim);
        f_opt_start = zeros(i_k*2,1);
        exitflag = [];
        Ff = [];
        for i = 1:i_k
         X0(i,:) = children_space_lb(j,:)+...
             x0x(i,:).*(children_space_ub(j,:)-children_space_lb(j,:));
         [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
             FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
             children_space_lb(j,:), children_space_ub(j,:), p_c,...
             OPTIONS, kparam);    % f opti is in the kriging
         [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)] = ...
             FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
             children_space_lb(j,:), children_space_ub(j,:),  p_c,...
             OPTIONS, kparam);
        end
        e_f_index = find(exitflag>=1);
                %%
        if isempty(e_f_index)
            for i = 1:i_k
                X0(i,:) = children_space_lb(j,:)+x0x(i,:)...
```

```
                .*(children_space_ub(j,:)-children_space_lb(j,:));
            [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
                FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
                children_space_lb(j,:), children_space_ub(j,:),
                    [],...
                OPTIONS, kparam);    % f opti is in the kriging
            [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)] =
                ...
                FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
                children_space_lb(j,:), children_space_ub(j,:), [],...
                OPTIONS, kparam);
            end
            e_f_index = find(exitflag>=1);
        end
        [f_opt(j,:), min_index] = min(f_opt_start(e_f_index,:));
        x_opt(j,:) = x_opt_start(e_f_index(min_index),:);
    end
    break;    %%%
%% normal
    else    % normal
        if L < U | (L == U & b(L_index(1)) < b(U_index(1)))
            cut_index = L_index(1);    % if over 1,choice the 1st
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_ub(end+1,:) = d_s_ub(end,:);
            children_space_ub(end, cut_index) = d_s_lb(end, cut_index)+2*
                L;
            d_s_lb(end+1,:) = d_s_lb(end,:);
            d_s_lb(end, cut_index) = d_s_lb(end, cut_index)+2*L;
            d_s_ub(end+1,:) = d_s_ub(end,:);
        elseif L > U | (L == U & b(L_index(1)) > b(U_index(1)))
            cut_index = U_index(1);    % if over 1,choice the 1st
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_lb(end, cut_index) = d_s_ub(end, cut_index)-2*
                U;
```

```
            children_space_ub(end+1,:) = d_s_ub(end,:);
            d_s_lb(end+1,:) = d_s_lb(end,:);
            d_s_ub(end+1,:) = d_s_ub(end,:);
            d_s_ub(end, cut_index) = d_s_ub(end, cut_index)-2*U;
        else % L == U > 0  ,special case
            [b_max, b_index] = max(b); % if over 1,choice the 1st
            cut_index = b_index(1);
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_ub(end+1,:) = d_s_ub(end,:);
            children_space_ub(end, cut_index) = d_s_lb(end, cut_index)
                +...
                (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            d_s_lb(end+1,:) = d_s_lb(end,:);
            d_s_lb(end, cut_index) = d_s_lb(end, cut_index)+...
                (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            d_s_ub(end+1,:) = d_s_ub(end,:);
            d_s_ub(end, cut_index) = d_s_lb(end, cut_index)+...
                (2/3)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
        end
    end
%%
%%%%%%  2nd optimal  %%%%%%
    x_opt_start = zeros(i_k*2,n_Dim);
    f_opt_start = zeros(i_k*2,1);
    Ff = [];
    e_f_index = [];
    for i = 1:i_k
        X0(i,:) = d_s_lb(end,:)+x0x(i,:).*(d_s_ub(end,:)-d_s_lb(end,:));
        [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
            FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
            d_s_lb(end,:), d_s_ub(end,:), p_c, OPTIONS, kparam);
        % f opti is in the kriging
        [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)] =...
            FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
```

```
            d_s_lb(end,:), d_s_ub(end,:), p_c, OPTIONS, kparam);
    end
    %%
    e_f_index = find(exitflag>=1);
    if isempty(e_f_index)
        for i = 1:i_k
            X0(i,:) = d_s_lb(end,:)+x0x(i,:).*(d_s_ub(end,:)-d_s_lb(end
                ,:));
            [x_opt_start(i,:) f_opt_start(i) exitflag(i)] =...
                FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
                d_s_lb(end,:), d_s_ub(end,:), [], OPTIONS, kparam);
            % f opti is in the kriging
            [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)]
                =...
                FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
                d_s_lb(end,:), d_s_ub(end,:), [], OPTIONS, kparam);
        end
        e_f_index = find(exitflag>=1);
    end
    [f_opt(end+1,:), min_index] = min(f_opt_start(e_f_index,:));
    x_opt(end+1,:) = x_opt_start(e_f_index(min_index),:);
%%
    b = d_s_ub(end,:)-d_s_lb(end,:); % measure the distance
    e_index = find((b/max(b))>= e_r);
    dis_lb(end,:) = ones(1,n_Dim)*inf;    dis_ub(end,:) = ones(1,n_Dim)*
        inf;
    dis_lb(end,e_index) = abs(x_opt(end,e_index)-d_s_lb(end,e_index));
    dis_ub(end,e_index) = abs(d_s_ub(end,e_index)-x_opt(end,e_index));
    [L, L_index] = min(dis_lb(end,:));
    [U, U_index] = min(dis_ub(end,:));
%% special case 2 bisection
    if L <= zero_error | U <= zero_error %| (L == U & L_index == U_index)
        if L <= zero_error & U <= zero_error
            [b_max, b_index] = max(b);  % if over,choice the 1st
```

```
    cut_index = b_index(1);
    children_space_lb(end+1,:) = d_s_lb(end,:);
    children_space_ub(end+1,:) = d_s_ub(end,:);
    children_space_ub(end, cut_index) = d_s_lb(end, cut_index)
       +...
        (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
    children_space_lb(end+1,:) = d_s_lb(end,:);
    children_space_lb(end, cut_index) = d_s_lb(end, cut_index)
       +...
        (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
    children_space_ub(end+1,:) = d_s_ub(end,:);
elseif L <= zero_error   % on the Lower boundary
    [b_max, b_index] = max(b);  % if over 1,choice the 1st
    cut_index = b_index(1);
    children_space_lb(end+1,:) = d_s_lb(end,:);
    children_space_ub(end+1,:) = d_s_ub(end,:);
    children_space_ub(end, cut_index) = d_s_lb(end, cut_index)
       +...
        (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
    children_space_lb(end+1,:) = d_s_lb(end,:);
    children_space_lb(end, cut_index) = d_s_lb(end, cut_index)
       +...
        (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
    children_space_ub(end+1,:) = d_s_ub(end,:);
elseif U <= zero_error   % on the Upper boundary
    [b_max, b_index] = max(b);  % if over 1,choice the 1st
    cut_index = b_index(1);
    children_space_lb(end+1,:) = d_s_lb(end,:);
    children_space_ub(end+1,:) = d_s_ub(end,:);
    children_space_ub(end, cut_index) = d_s_lb(end, cut_index)
       +...
        (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
    children_space_lb(end+1,:) = d_s_lb(end,:);
```

```
            children_space_lb(end, cut_index) = d_s_lb(end, cut_index)
                +...
                 (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            children_space_ub(end+1,:) = d_s_ub(end,:);
        end
%% re-optimum from two equal design space
        for j = 2:3
            x_opt_start = zeros(i_k*2,n_Dim);
            f_opt_start = zeros(i_k*2,1);
            exitflag = [];
            Ff = [];
            for i = 1:i_k
             X0(i,:) = children_space_lb(j,:)+x0x(i,:)...
                  .*(children_space_ub(j,:)-children_space_lb(j,:));
             [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
                 FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
                  children_space_lb(j,:), children_space_ub(j,:), p_c, ...
                  OPTIONS, kparam);    % f opti is in the kriging
             [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)]
                =...
                  FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
                  children_space_lb(j,:), children_space_ub(j,:),  p_c,...
                  OPTIONS, kparam);
            end
        %%
            e_f_index = find(exitflag>=1);
            if isempty(e_f_index)
                for i = 1:i_k
                    X0(i,:) = children_space_lb(j,:)+x0x(i,:)....
                        *(children_space_ub(j,:)-children_space_lb(j,:));
                    [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
                      FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq
                        ,...
```

```
                children_space_lb(j,:), children_space_ub(j,:),
                    [],...
                OPTIONS, kparam);    % f opti is in the kriging
        [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)] = ...
            FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
      children_space_lb(j,:), children_space_ub(j,:), [], OPTIONS, kparam
        );
            end
            e_f_index = find(exitflag>=1);
        end
        [f_opt(j,:), min_index] = min(f_opt_start(e_f_index,:));
        x_opt(j,:) = x_opt_start(e_f_index(min_index),:);
    end
    break;    %%%
%% normal
  else   % normal
    if L < U | (L == U & b(L_index(1)) < b(U_index(1)))
        cut_index = L_index(1);   % if over 1,choice the 1st
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_ub(end+1,:) = d_s_ub(end,:);
        children_space_ub(end, cut_index) = d_s_lb(end, cut_index)+2*
            L;
        d_s_lb(end+1,:) = d_s_lb(end,:);
        d_s_lb(end, cut_index) = d_s_lb(end, cut_index)+2*L;
        d_s_ub(end+1,:) = d_s_ub(end,:);
    elseif L > U | (L == U & b(L_index(1)) > b(U_index(1)))
        cut_index = U_index(1);   % if over 1,choice the 1st
        children_space_lb(end+1,:) = d_s_lb(end,:);
        children_space_lb(end, cut_index) = d_s_ub(end, cut_index)-2*
            U;
        children_space_ub(end+1,:) = d_s_ub(end,:);
        d_s_lb(end+1,:) = d_s_lb(end,:);
        d_s_ub(end+1,:) = d_s_ub(end,:);
        d_s_ub(end, cut_index)  = d_s_ub(end, cut_index)-2*U;
```

```
        else % L == U > 0
            [b_max, b_index] = max(b);  % if over 1,choice the 1st
            cut_index = b_index(1);
            children_space_lb(end+1,:) = d_s_lb(end,:);
            children_space_ub(end+1,:) = d_s_ub(end,:);
            children_space_ub(end, cut_index) = d_s_lb(end, cut_index)
                +...
                 (1/2)*(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            d_s_lb(end+1,:) = d_s_lb(end,:);    d_s_lb(end, cut_index)
                =...
                 d_s_lb(end, cut_index)+(1/2)...
                 *(d_s_ub(end, cut_index)-d_s_lb(end, cut_index));
            d_s_ub(end+1,:) = d_s_ub(end,:);
        end
%%    3rd optimal   %%%%%
%%
    x_opt_start = zeros(i_k*2,n_Dim);
    f_opt_start = zeros(i_k*2,1);
    Ff = [];
    e_f_index = [];
    for i = 1:i_k
        X0(i,:) = d_s_lb(end,:)+x0x(i,:).*(d_s_ub(end,:)-d_s_lb(end,:));
        [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
            FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
            d_s_lb(end,:), d_s_ub(end,:), p_c, OPTIONS, kparam);
        % f opti is in the kriging
        [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)] = ...
            FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
            d_s_lb(end,:), d_s_ub(end,:), p_c, OPTIONS, kparam);
    end
    %%
    e_f_index = find(exitflag>=1);
    if isempty(e_f_index)
        for i = 1:i_k
```

```
            X0(i,:) = d_s_lb(end,:)+x0x(i,:).*(d_s_ub(end,:)-d_s_lb(end
                ,:));
            [x_opt_start(i,:) f_opt_start(i) exitflag(i)] = ...
                FMINCON(@f_predictkrige, X0(i,:), A, B, Aeq, Beq,...
                d_s_lb(end,:), d_s_ub(end,:), [], OPTIONS, kparam);
            % f opti is in the kriging
            [x_opt_start(i+i_k,:) f_opt_start(i+i_k) exitflag(i+i_k)]
                =...
                FMINCON(@f_kriging_95, X0(i,:), A, B, Aeq, Beq,...
                d_s_lb(end,:), d_s_ub(end,:), [], OPTIONS, kparam);
        end
        e_f_index = find(exitflag>=1);
    end
    [f_opt(end+1,:), min_index] = min(f_opt_start(e_f_index,:));
    x_opt(end+1,:) = x_opt_start(e_f_index(min_index),:);
    end
    children_space_lb(end+1,:) = d_s_lb(end,:); %the third subspace
    children_space_ub(end+1,:) = d_s_ub(end,:);
end
    d = 0.5*((sum((children_space_ub-children_space_lb).^2, 2)).^0.5);
    % the longest diagonal line
end
```

f-d filter

0

```
function [ D ] = f_d_g_1( f_d )
%         1    2    3       4, 5              6, 7
%         x    y
% f_d = [d, f_opt, g, children_space_lb, children_space_ub ];
f_d_design = []; f_d_no = []; f_d_ok = [];
epsilon = 1e-3;   %  1e-3~1e-7
f_d_ok = f_d;
```

```
[d_max, d_index] = max(f_d_ok(:, 1));
[f_min, f_index] = min(f_d_ok(:, 2));
d1 = f_d_ok(f_index, 1);f1 = f_min;
f_d_design = [f_d_design; f_d_ok(f_index, :)];
df_f = f_d_ok(d_index, 2);df_d = f_d_ok(d_index, 1);
while size(f_d_ok, 1) > 1
    m = (df_f-f1)./(df_d-d1);
    slop = m*(f_d_ok(:, 1)-d1)-(f_d_ok(:, 2)-f1);
    ok_index = find( slop > 0 );
    m1 = (f_d_ok(ok_index, 2)-f1*(1-epsilon))./(f_d_ok(ok_index, 1)-d1);
    epsilon = 0;
    [m_min, m_index] = min(m1);
    d1 = f_d_ok(ok_index(m_index), 1);
    f1 = f_d_ok(ok_index(m_index), 2);
    total_index = 1:size(f_d_ok, 1);
    no_index = setdiff(total_index', [f_index; ok_index]);
    f_d_design = [f_d_design; f_d_ok(ok_index(m_index), :)];
    f_d_no = [f_d_no; f_d_ok(no_index, :)];
    f_d_ok = f_d_ok( ok_index, :);
end
    if ((df_f-f_d_design(end, 2))./(df_d-f_d_design(end, 1))) >= m
        f_d_design(end+1, :) = f_d(d_index, :);
    end
f_d_other = [];
delete_index = [];
for i = 1:size(f_d_design, 1)
    kk = size(f_d_no, 1);
    for j = 1:kk
        if f_d_design(i, :) == f_d_no(j, :)
            delete_index(end+1, :) = j;
        end
    end
end
delete_index = unique(delete_index);
```

```
f_d_no(delete_index, :) = [];
D.design = f_d_design;
D.other = f_d_no;
end
```

f-g filter

```
function output = f_g( f_d )
%         1    2    3       4, 5              6, 7
%              y    x
% f_d = [d, f_opt, g, children_space_lb, children_space_ub ];

g_tolerance = 1e-8;

    if all( f_d(:, 3) <= g_tolerance)
        output.design = f_d;
        output.other = [];
    else
        g_index = find(f_d(:, 3) <= g_tolerance);
        %  max{0, g} <= 0, design space is feasible
        g_index_1 = find(f_d(:, 3) > g_tolerance);
        [f_d_min, g_index_2] = min(f_d(g_index_1, 2));   % find min
        g_index_3 = find((f_d(g_index_1, 2) <= f_d_min) |...
            (f_d(g_index_1, 3) <= f_d(g_index_1(g_index_2), 3)));%kbn
        output.design = [f_d(g_index, :); f_d(g_index_1(g_index_3), :)];
        total_index = 1:size(f_d,1);
        design_index = [g_index', g_index_1(g_index_3)'];
        other_index = setdiff(total_index', design_index);
        output.design = f_d(design_index, :);
        output.other = f_d(other_index, :);
    end
end
```

Kriging model

```
function kparam = f_crossvariogram_fit (x, y, lb, ub, fittype, SCFtype)

% function kparam = f_variogram_fit(x, y, lb, ub, fittype, SCFtype)
%
% This function file takes in a data set and creates the kriging
    metamodels
% for each response.  The resulting information is stored in a structure
% array named kparam.
%
% It uses the theoretical variogram to fit the experimental variogram not
% the likelihood method.
%
% ---------
% INPUTS:
% ---------
%
% x - [n X d] matrix of sample point input values
% y - [n X m] matrix of sample point output values
% lb - optional [1 x d] vector of lower bounds on the design variables
%       (default=min(x))
% ub - optional [1 x d] vector of upper bounds on the design variables
%       (default=max(x))
% fittype -
% SCFtype -
%
% ----------
% OUTPUTS:
% ----------
% kparam - the structure array containing the following kriging
    parameters:
%    inputs - [n x d] matrix of sample point input values
%    outputs - [n x m] matrix of sample point output values
```

```
%   theta - the [m x d] matrix containing the theta values for prediction
%   p - the [m x d] matrix of p values (for the general exponential SCF)
%   nugget - the [m x 1] vector for the nugget effect
%   lb - the [m x d] matrix of lower bounds on the design variables
%   ub - the [m x d] matrix of upper bounds on the design variables
%   beta - the [(?) x m] matrix of global model terms
%   sigma2 - the [1 x m] vector of variance terms used for prediction
%   K - the [n+? x n+? x m] array of expanded correlation matrices
%   mse - the least square error between the. and exp. variogram
%   fittype - how many parameter to be fiting
%   SCFtype - SCF type selected from f_SCF for fitting kriging model
%
% NCKU, System Optimization Lab.,
% Coded by Y.-C. Huang
% last updated 2009/08/26

% ------------------------
% some property varilble
% ------------------------
n = size(x,1); % the number of input sampled points
m = size(y,2); % the number of output dimensions

%% model fitting
% ----------------------------------------
% fit variogram for each output dimension
% ----------------------------------------
mse = [ones(m)]; sigma_kl = [ones(m)]; theta_kl = [ones(m)]; p = [ones(m)
    ]; nugget = [ones(m)];
for meti = 1:m
    for metj = meti:m
    % ------------------------
    % set the fitting parameter
    % ------------------------
```

```
% ----------------------------------------
% check exp_vario_rt has NaN or not
% ----------------------------------------
clear exp_vario_hst exp_vario_rt
if(meti==metj)
    [exp_data, exp_vario_hst, exp_vario_rt] = f_variogram_exp(x,y(:,
        meti),lb,ub);
else
    ym=[y(:,meti),y(:,metj)];
    [exp_data, exp_vario_hst, exp_vario_rt] = f_crossvariogram_exp(x,
        ym,lb,ub);
end
iter = 1;
for i = 1:size(exp_vario_rt,1)
    if ~isnan(exp_vario_rt(i,1))
        exp_vario_hs{meti,metj}(iter,1) = exp_vario_hst(i,1);
        exp_vario_r{meti,metj}(iter,1) = exp_vario_rt(i,1);
        iter = iter + 1;
    end
end

% ----------------------------------------
% beginning fitting kriging with UMDIRECT
% ----------------------------------------
if nargin <= 4
    fittype = 1; % select how many parameter to be fiting
    SCFtype = 1; % SCF type for fitting
end

switch fittype
    case 1
        % ------------
        % sigma2, theta
```

```
            % ---------------
            if min(exp_vario_r{meti,metj})>0
            vlb0{meti,metj} = [0, 0.01];
            vub0{meti,metj} = [max(exp_vario_r{meti,metj})*1.5, 15];
            else
            vlb0{meti,metj} = [min(exp_vario_r{meti,metj})*1.5, 0.01];
            vub0{meti,metj} = [0, 15];
            end

        case 2
            % ------------------
            % sigma2, theta, p
            % ------------------
            vlb0{meti,metj} = [0.01, 0.01, 0.01];
            vub0{meti,metj} = [max(exp_vario_r{meti,metj})*2, 1e3, 1.99];

        case 3
            % -------------------------
            % sigma2, theta, p, nugget
            % -------------------------
            vlb0{meti,metj} = [0.01, 0.01, 1.5, 0.01];
            vub0{meti,metj} = [max(exp_vario_r{meti,metj})*2, 1e3, 1.99,
                0.99];

        otherwise
    end
    end
end
q=1;
    for meti = 1:m
        for metj = meti:m
        vlb(1,q:q+1)=[vlb0{meti,metj}];
        vub(1,q:q+1)=[vub0{meti,metj}];
        q=q+2;
```

```
    end
end

%
    ------------------------------------------------------------------------

fileInfo.fName = 'f_all_variogram_cross_mse';
fileInfo.fParams = {exp_vario_hs, exp_vario_r, SCFtype};
fileInfo.gName = 'permissible_model';
UMDoptions.display = 0;
UMDoptions.maxIter = 200;
UMDoptions.maxfCount = 2e2*size(vlb,2);
UMDoptions.termType = 3;
UMDoptions.saveFile = 'noSave';
UMDoptions.localSearch = 0;
outstruct = UMDIRECT(fileInfo, vlb, vub, UMDoptions);

mse = [outstruct.fBest];
q=0;
for meti=1:m
    for metj=meti:m
     sigma_kl(meti,metj) = [outstruct.xBest(1,1+q)];
     sigma_kl(metj,meti)=sigma_kl(meti,metj);
     theta_kl(meti,metj) = [outstruct.xBest(1,2+q)];
     theta_kl(metj,meti)=theta_kl(meti,metj);
      switch fittype
          case 1
             p(meti,metj) = [1.99];
             p(metj,meti)=p(meti,metj);
             nugget(meti,metj) = [ 1e-12];
             nugget(metj,meti)=nugget(meti,metj);
          case 2
             p(meti,metj) = [outstruct.xBest(1,3)];
             p(metj,meti)=p(meti,metj);
```

```
                nugget(meti,metj) = [1e-12];
                nugget(metj,meti)=nugget(meti,metj);
            case 3
                p(meti,metj) = [outstruct.xBest(1,3)];
                p(metj,meti)=p(meti,metj);
                nugget(meti,metj) = [outstruct.xBest(1,4)];
                nugget(metj,meti)=nugget(meti,metj);
            otherwise
        end
        q=q+2;
    end
end

%e% %    qi=1;
% %    for meti = 1:m
% %        for metj = 1:m
% %            subplot(m,m,qi);plot(exp_vario_hs{meti,metj},exp_vario_r{
  meti,metj},'--o')
% %            hold on
% %            hst=[0:0.05:max(exp_vario_hs{meti,metj})];
% %            rt=sigma_kl(meti,metj).*(1-exp(-theta_kl(meti,metj).*hst)
  );
% %            subplot(m,m,qi);
% %            hold on
% %            plot(hst,rt)
% %            qi=qi+1;
% %        end
% %    end

%% kriging K-matrix
% ------------------------------
% rescale input data to [0,1]
% ------------------------------
xs = (x-ones(n,1)*lb)./(ones(n,1)*(ub-lb));
```

```
K=cell(m);
%xs=x;
for meti=1:m
    for metj = meti:m

        if (meti==metj)
        xi=xs( find(      y(:,meti)~=inf        ),:   );
        xj=xi;
        else
        xi=xs(find(      y(:,meti)~=inf        ),: );
        xj=xs(find(      y(:,metj)~=inf        ),: );
        end

        n=size(xi(:,1));
        s=size(xj(:,1));

        R = ones(n,s); % correlation matrix, R
         for i = 1:n
              for j = 1:s
                   R(i,j) = ...
                       f_SCF(norm(xi(i,:)-xj(j,:)),theta_kl(meti,metj),p(
                           meti,metj),SCFtype);
              end
         end
          R = (1-nugget(meti,metj))*R + nugget(meti,metj)*eye(n,s);
          K{meti,metj} = sigma_kl(meti,metj)*R; % expanded correlation
              matrix
          if meti~=metj
           K{metj,meti}=K{meti,metj}';
          end
    end
end
sum_n=0;
```

```
for i=1:m
    nn=size(find(y(:,i)~=inf),1);
    sum_n=sum_n+nn;
end
R=ones(sum_n);
F = zeros(sum_n,m);
 m1=0;
for meti=1:m
    n1=0;
    for metj = 1:m
            R(m1+1:m1+size(K{meti,metj},1),n1+1:n1+size(K{meti,metj},2))=
                K{meti,metj};
        if meti==metj
            F(m1+1:m1+size(K{meti,metj},1),metj)=ones(size(K{meti,metj
                },1),1);
        end
%         if meti~=metj
%             R(n1+1:n1+size(K{meti,metj},2),m1+1:m1+size(K{meti,metj},1))
                =K{meti,metj}';
%         end
            n1=n1+size(K{meti,metj},2);
    end
    m1=m1+size(K{meti,metj},1);
end
G=[zeros(m)];
CK=[R F;F' G];%ock martix


kparam.inputs = x;
kparam.outputs = y;
kparam.theta_kl = theta_kl;
kparam.p = p;
kparam.nugget = nugget;
kparam.lb = lb;
```

```
kparam.ub = ub;
%kparam.beta = beta;
kparam.sigma_kl = sigma_kl;
kparam.K = K;
kparam.CK = CK;
kparam.mse = mse;
kparam.fittype = fittype;
kparam.SCFtype = SCFtype;
kparam.sum_n = sum_n;

function [ynew, yvar, lambda] = f_predictkrige(xnew, kparam)

% function [ynew, yvar, lambda] = f_predictkrige(xnew, kparam)
%
% This function file uses the kparam values from fitkrige.m to predict
    the value
% of the response at several locations using a kriging model.
%
% --------
% INPUTS:
% --------
% xnew - [nn x d] matrix containing the input values to predict
%
% kparam - the structure array containing the following kriging
    parameters:
%   inputs - [n x d] matrix of sample point input values
%   outputs - [n x m] matrix of sample point output values
%   theta - the [m x d] matrix containing the theta values for prediction
%   p - the [m x d] matrix of p values (for the general exponential SCF)
%   nugget - the [m x 1] vector for the nugget effect
%   lb - the [m x d] matrix of lower bounds on the design variables
%   ub - the [m x d] matrix of upper bounds on the design variables
%   beta - the [(?) x m] matrix of global model terms
%   sigma2 - the [1 x m] vector of variance terms used for prediction
```

```
%   K - the [n+? x n+? x m] array of expanded correlation matrices
%   mse - the least square error between the. and exp. variogram
%   fittype - how many parameter to be fiting
%   SCFtype - SCF type selected from f_SCF for fitting kriging model
%
% ---------
% OUTPUTS:
% ---------
% ynew - the [nn x m] matrix of predicted responses where the different
%        responses are separated by columns.
%
% yvar - the [nn x m] matrix of variances in the predicted responses.
%
%
% NCKU, System Optimization Lab.,
% Coded by Y.-C. Huang
% last updated 2009/08/26

% ------------------------------------
% pull parameters out of structure array
% ------------------------------------
x = kparam.inputs;
y = kparam.outputs;
theta = kparam.theta;
p = kparam.p;
nugget = kparam.nugget;
lb = kparam.lb;
ub = kparam.ub;
beta = kparam.beta;
sigma2 = kparam.sigma2;
K = kparam.K;
fittype = kparam.fittype;
SCFtype = kparam.SCFtype;
```

```
% ---------------------------------
% determine the size of data set
% ---------------------------------
[n,d] = size(x);
[ny,m] = size(y);

% ---------------------------------
% compute some stuff needed later
% ---------------------------------
o = ones(n,1);
a = o*lb; b=o*ub;
xs = (x-a)./(b-a); % rescale sample data to [0,1]
nn = size(xnew,1);
xnews = (xnew-ones(nn,1)*lb)./ ...
    (ones(nn,1)*ub-ones(nn,1)*lb); % rescale xnew to [0,1] also

% ---------------------------
% initial ynew, yvar, lambda
% ---------------------------
ynew = ones(nn,m);
if nargout > 1, yvar = ones(nn,m); end
if nargout > 2, lambda = ones(nn,n,m); end

% ---------------------------------------------------------------
% begin looping through all response models and untried samples
% ---------------------------------------------------------------
for met = 1:m % number of response models
    Kinv = inv(K(:,:,met)); % kriging weights K = [[R,F]; [F',0]]
    xnewsmat = shiftdim(xnews',-1); % nn,n,1 -> 1,n,nn
    hsmat = sqrt(sum((repmat(xs,[1,1,nn])-repmat(xnewsmat,[n,1])).^2,2));
    rxmat_tmp = (1-nugget(met))*f_SCF(hsmat,theta(met,:),p(met,:),SCFtype
        );
    if nn == 1, rxmat = rxmat_tmp;
    else rxmat = shiftdim(rxmat_tmp,2)'; end % n,1,nn -> n,nn,1
```

```
    lamKT = [rxmat', ones(size(rxmat,2),1)]*Kinv;
    ynew(:,met) = lamKT(:,1:n)*y(:,met); % mean estimate
    if (nargout > 1)
        temp = 1 - sum(lamKT.*[rxmat; ones(1,size(rxmat,2))]',2);
        yvar(:,met) = max(eps, sigma2(met)*temp); % variance estimate
        if (nargout > 2), lambda = lamKT; end
    end
end

function r = f_SCF(hs, theta, p, SCFtype)
% Spatial Correlation Coefficient (w/o sigma2)

theta = repmat(theta,[size(hs,1),1,size(hs,3)]);
p = repmat(p,[size(hs,1),1,size(hs,3)]);

switch SCFtype
    case 1
        % ------------------------------------------------------
        % General exponential model (Gaussian model when p = 2)
        % ------------------------------------------------------
        r(:,1,:) = exp(-theta.*hs);

    case 2
        % ---------------------
        % Matern linear model
        % ---------------------
        r(:,1,:) = (1+theta.*hs).*exp(-theta.*hs);

    case 3
        % ---------------------
        % Matern cubic model
        % ---------------------
        r(:,1,:) = (1+theta.*hs+(theta.^2).*(hs.^2)).*exp(-(theta.*hs));
```

```
    otherwise
end

function mse = f_variogram_cross_mse(x, exp_vario_hs, exp_vario_r,
    SCFtype)
% x is [sigma2, theta, p, nugget], not input

sigma2 = x(1,1);
theta = x(1,2);
if size(x,2) > 2, p = x(1,3); else p = 1.99; end
if size(x,2) > 3, nugget = x(1,4); else nugget = 1e-12; end

sigma2mat = repmat(sigma2,[size(exp_vario_r,1),1]);
SCFmat = f_SCF(exp_vario_hs(:,1), theta, p, SCFtype);
the_vario_r(:,1) = sigma2mat.*(1-(1-nugget)*SCFmat);

se(:,1) = (exp_vario_r - the_vario_r).^2;
mse = mean(se);

function [exp_data, exp_vario_hs, exp_vario_r] = f_variogram_exp(x, y, lb
    , ub)
% caution: only deal with one dimension in y

%-----------------------------------
% rescale to input data to [0,1]
%-----------------------------------
[n,d] = size(x);
xs = (x-ones(n,1)*lb)./(ones(n,1)*(ub-lb));



%-------------------------
% variogram data produce
%-------------------------
data = [];
for i = 1:n-1
```

```
    shs = 0; % square hs
    for j = 1:d, shs = shs + (xs(i,j)-xs(i+1:size(xs,1),j)).^2; end
    % hs, sample 1 number, sample 2 number, r(rh)
    data = [data; [sqrt(shs), ones(size(xs,1)-i,1)*i, ((i+1):size(xs,1))'
        , ...
                    0.5*(y(i,1) - y(i+1:size(xs,1),1)).^2] ];
end
cmin = min(data(:,1));
cmax = 0.8*max(data(:,1));
div_reg = 10;

% ---------
% set lag
% ---------
for i = 1:div_reg+1
    region_index(i,1) = cmin + (i-1)*(cmax - cmin)/div_reg;
end

% ----------
% lag zone
% ----------
region_zone(1,1:div_reg) = region_index(1:div_reg,1);
region_zone(2,1:div_reg) = region_index(2:div_reg+1,1);

region_value = zeros(div_reg,2);
for j = 1:div_reg
    mask = double(data(:,1) >= region_zone(1,j)) & (data(:,1) <=
        region_zone(2,j));
    region_value(j,1) = sum(mask.*data(:,4)); % sum of r(h)
    region_value(j,2) = sum(mask); % sum of sample number
end

exp_data = data;
exp_vario_r(1:div_reg,1) = (region_value(:,1)./region_value(:,2));
```

```
exp_vario_hs(1:div_reg,1) = 0.5*sum(region_zone,1);
% % figure
% % plot(exp_data(:,1),exp_data(:,4),'.')
%figure
%plot(exp_vario_hs,exp_vario_r,'o')

function kparam = f_variogram_fit(x, y, lb, ub, fittype, SCFtype)

% function kparam = f_variogram_fit(x, y, lb, ub, fittype, SCFtype)
%
% This function file takes in a data set and creates the kriging
    metamodels
% for each response.  The resulting information is stored in a structure
% array named kparam.
%
% It uses the theoretical variogram to fit the experimental variogram not
% the likelihood method.
%
% ---------
% INPUTS:
% ---------
%
% x - [n X d] matrix of sample point input values
% y - [n X m] matrix of sample point output values
% lb - optional [1 x d] vector of lower bounds on the design variables
%       (default=min(x))
% ub - optional [1 x d] vector of upper bounds on the design variables
%       (default=max(x))
% fittype -
% SCFtype -
%
% ----------
% OUTPUTS:
% ----------
```

```
% kparam - the structure array containing the following kriging
    parameters:
%    inputs - [n x d] matrix of sample point input values
%    outputs - [n x m] matrix of sample point output values
%    theta - the [m x d] matrix containing the theta values for prediction
%    p - the [m x d] matrix of p values (for the general exponential SCF)
%    nugget - the [m x 1] vector for the nugget effect
%    lb - the [m x d] matrix of lower bounds on the design variables
%    ub - the [m x d] matrix of upper bounds on the design variables
%    beta - the [(?) x m] matrix of global model terms
%    sigma2 - the [1 x m] vector of variance terms used for prediction
%    K - the [n+? x n+? x m] array of expanded correlation matrices
%    mse - the least square error between the. and exp. variogram
%    fittype - how many parameter to be fiting
%    SCFtype - SCF type selected from f_SCF for fitting kriging model
%
% NCKU, System Optimization Lab.,
% Coded by Y.-C. Huang
% last updated 2009/08/26

% ------------------------
% some property varilble
% ------------------------
n = size(x,1); % the number of input sampled points
m = size(y,2); % the number of output dimensions

%% model fitting
% ----------------------------------------
% fit variogram for each output dimension
% ----------------------------------------
for met = 1:m
    % ------------------------
    % set the fitting parameter
    % ------------------------
```

```
if met == 1, mse = []; sigma2 = []; theta = []; p = []; nugget = [];
    end

% ------------------------------------
% check exp_vario_rt has NaN or not
% ------------------------------------
clear exp_vario_hst exp_vario_rt exp_vario_hs exp_vario_r
[exp_data, exp_vario_hst, exp_vario_rt] = f_variogram_exp(x,y(:,met),
    lb,ub);
iter = 1;
for i = 1:size(exp_vario_rt,1)
    if ~isnan(exp_vario_rt(i,1))
        exp_vario_hs(iter,1) = exp_vario_hst(i,1);
        exp_vario_r(iter,1) = exp_vario_rt(i,1);
        iter = iter + 1;
    end
end

% ------------------------------------------
% beginning fitting kriging with UMDIRECT
% ------------------------------------------
if nargin <= 4
    fittype = 1; % select how many parameter to be fiting
    SCFtype = 1; % SCF type for fitting
end

switch fittype
    case 1
        % ---------------
        % sigma2, theta
        % ---------------
        vlb = [0, 0.01];
        vub = [max(exp_vario_r)*2, 20];
```

```
    case 2
        % ------------------
        % sigma2, theta, p
        % ------------------
        vlb = [0.01, 0.01, 0.01];
        vub = [max(exp_vario_r)*2, 1e3, 1.99];

    case 3
        % -------------------------
        % sigma2, theta, p, nugget
        % -------------------------
        vlb = [0.01, 0.01, 1.5, 0.01];
        vub = [max(exp_vario_r)*2, 1e3, 1.99, 0.99];

    otherwise
end

%
    ------------------------------------------------------------------------

fileInfo.fName = 'f_variogram_mse';
fileInfo.fParams = {exp_vario_hs, exp_vario_r, SCFtype};
UMDoptions.display = 0;
UMDoptions.maxIter = 100;
UMDoptions.maxfCount = 2e2*size(vlb,2);
UMDoptions.termType = 3;
UMDoptions.saveFile = 'noSave';
UMDoptions.localSearch = 0;
outstruct = UMDIRECT(fileInfo, vlb, vub, UMDoptions);

mse = [mse; outstruct.fBest];
sigma2 = [sigma2; outstruct.xBest(1,1)];
theta = [theta; outstruct.xBest(1,2)];
```

```
%%      figure
%%      hq=[0:0.001:1];
%%      rq=sigma2(met).*(1-exp(-theta(met).*hq));
%%      plot(hq,rq)

    switch fittype
        case 1
            p = [p; 1.99];
            nugget = [nugget; 1e-12];
        case 2
            p = [p; outstruct.xBest(1,3)];
            nugget = [nugget; 1e-12];
        case 3
            p = [p; outstruct.xBest(1,3)];
            nugget = [nugget; outstruct.xBest(1,4)];
        otherwise
    end
end

%% kriging K-matrix
% ------------------------------
% rescale input data to [0,1]
% ------------------------------
xs = (x-ones(n,1)*lb)./(ones(n,1)*(ub-lb));

for met = 1:m
    R = ones(n); % correlation matrix, R
    for i = 1:n-1
        for j = i+1:n
            R(i,j) = ...
                f_SCF(norm(xs(i,:)-xs(j,:)),theta(met,:),p(met,:),SCFtype
                    );
            R(j,i) = R(i,j);
        end
```

```
    end
    R = (1-nugget(met))*R + nugget(met)*eye(n);
    F = ones(n,1); % functional form f(x)
    K(:,:,met) = [[R,F]; [F',0]]; % expanded correlation matrix
    beta(:,met) = ((F'/R)*y(:,met))/((F'/R)*F); % y = fx*beta + z
end

kparam.inputs = x;
kparam.outputs = y;
kparam.theta = theta;
kparam.p = p;
kparam.nugget = nugget;
kparam.lb = lb;
kparam.ub = ub;
kparam.beta = beta;
kparam.sigma2 = sigma2;
kparam.K = K;
kparam.mse = mse;
kparam.fittype = fittype;
kparam.SCFtype = SCFtype;

function mse = f_variogram_mse(x, exp_vario_hs, exp_vario_r, SCFtype)
% x is [sigma2, theta, p, nugget], not input

sigma2 = x(1,1);
theta = x(1,2);
if size(x,2) > 2, p = x(1,3); else p = 1.99; end
if size(x,2) > 3, nugget = x(1,4); else nugget = 1e-12; end

sigma2mat = repmat(sigma2,[size(exp_vario_r,1),1]);
SCFmat = f_SCF(exp_vario_hs(:,1), theta, p, SCFtype);
the_vario_r(:,1) = sigma2mat.*(1-(1-nugget)*SCFmat);

se(:,1) = (exp_vario_r - the_vario_r).^2;
```

```
mse = mean(se);

function results = UMDIRECT(fileInfo,lb,ub,options,restartFile)
% results=UMDIRECT(fileInfo,lb,ub,options,restartFile)
%
% This is the University of Michigan's implementation of the DIRECT
% algorithm.
% It solves problems of the form:
%
%     min   f(x)
%   s.to:  g(x) <= 0
%          lb <= x <= ub
%
% It is based upon work from the following papers:
%
% 1) D.R. Jones, C.D. Perttunen and B.E. Stuckman (1993).
% "Lipschitzian Optimization Without the Lipschitz Constant",
% Journal of Optimization Theory and Application, 79(1): 157-181.
%
% 2) D.R. Jones (2001). "DIRECT", Entry in "Encyclopedia of Optimization
    ",
% Kluwer Academic Publishers, 1:431-440.
%
%
%INPUTS:
%--------
%
% fileInfo - structure array containing following file names:
%   fileInfo.fName           filename for objective function
%   fileInfo.gName           filename for constraint functions
%   fileInfo.fParams         row cell array of additional inputs for
    fName
%   fileInfo.gParams         row cell array of additional inputs for
    gName
```

```
%
% lb - [1 x nVar] vector of lower bounds on the design variables
%
% ub - [1 x nVar] vector of upper bounds on the design variables
%
% options - structure array of DIRECT specific options:
%     options.maxfCount     maximum number of function calls allowed (
    default = 200*nVar)
%     options.maxIter       maximum number of iterations allowed (default =
    50)
%     options.conTol        [1 x nCon] vector of maximum allowable
    constraint violation (default = 1e-3)
%                           (if scalar, use same tolerance on all
    constraints)
%     options.display       scalar value for amount of output sent to screen
    (default = 0)
%     options.saveFile      save the results structure at each iteration to
    'results_fName' (default = 0)
%     options.lgBalance     local / global balance parameter (default = 1e
    -4)
%     options.localSearch   allow for local optimization (0=no, 1=use SQP,
    2=use local-DIRECT)
%     options.termType      choose the termination criterion:
%                           1 = fCount limit exceeded
%                           2 = iter limit
    exceeded
%                           3 = either fCount
    OR iter limit exceeded
%                           4 = both fCount
    AND iter limit exceeded
%                           5 = no
    improvement in last 100 evaluations
%                           6 = no
    improvement in last 10 iterations
```

```
%
% restartFile - name of file to load in when restarting DIRECT
%               (either the name of file or the structure array itself
%                                                    are acceptable)
%
%
%OUTPUTS:
%---------
%
% results - structure array of results containing the following fields:
%   results.xBest        [1 x nVar] vector for best design point
%   results.fBest        scalar best feasible function value at xBest
%   results.gBest        [1 x nCon] vector of constraint values at xBest
%   results.fCount       total number of function evaluations
%   results.iter         number of iterations performed
%   results.lb           [1 x nVar] vector of lower
    bounds on design variables
%   results.ub           [1 x nVar] vector of upper
    bounds on design variables
%   results.fileInfo     structure array containing the fileInfo pased to
    UMDIRECT
%   results.options      structure array containing the options pased to
    UMDIRECT
%   results.fChangeRate  scalar rate of change in f (used for calculating
    constraint weights)
%   results.gChangeRate  [1 x nCon] vector of rate changes for each
    constraint
%   results.conWeight    [1 x nCon] vector of constraint weights (for
    auxiliary function)
%
%   results.hist - structure array containing history of search:
%          hist.fCount   column vector w/ function call number when beat
    previous fBest
```

```
%            hist.iter     column vector w/ iteration number when beat
    previous fBest
%
%   results.rect - structure array of rectangle properties:
%            rect.dist                  [fCount x 1] vector of vertex
    distances
%            rect.nDivAll       [1 x nVar] vector w/ number of divisions
    along each dimensions for all rects.
%            rect.nDiv          [fCount x nVar] matrix w/ num. of
    divisions along each dimension for each rect.
%            rect.x             [fCount x nVar] matrix of design points
%            rect.f             [fCount x 1] vector of objective function
    values
%            rect.g             [fCount x nCon] matrix of constraint values
%            rect.aux           column vector of Auxiliary Function values
    (constrained version only)
%            rect.auxCon        column vector of constraint violation
    portion of auxiliary function
%
%
%EXAMPLE:
%---------
%Here's an example of how one might start a run of DIRECT from scratch.
%
% fileInfo.fName='obj';          %name of file to run objective function
% fileInfo.gName='cons';         %name of file to run constraints
% fileInfo.fParams=[1e4 10];     %extra parameters that must be sent to '
    obj.m'
%
% lb=[0 5 0];                              %lower bounds on x
% ub=[10 25 50];                           %upper bounds on x
%
% options.maxfCount=500;         %limit on # of function calls
```

```
% options.conTol=[1e-6 1e-2];   %constraint tolerance on the 2
    constraints
% options.display=1;             %show progress every iteration
% options.saveFile='myfile';     %name of file to save to every iteration
%
% myresults=DIRECT(fileInfo,lb,ub,options);      %start DIRECT
%
%
%
%If the user then wanted to continue with the same problem,
%       they could restart DIRECT with the following.
%
% newoptions.maxfCount=1000;  %set a higher limit on the fCount
%
% mynewresults=DIRECT([],[],[],newoptions,'myfile');
%                  OR
% mynewresults=DIRECT([],[],[],newoptions,myresults);
%
%
%
% copyright 2002
% Ryan A. Fellini, Michael J. Sasena, John W. Whitehead
% University of Michigan
% last modified 7/12/02
%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Step 0 - Initialize DIRECT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<5,
    restartFile=[];
```

```
    if nargin<4
        options=[];
        if nargin<3
            error('Not enough inputs for DIRECT!')
        end
    end
end

%fill in any missing fileInfo arguments
if ~isempty(fileInfo)
    %allow for fileInfo to be empty in case of restart
    if ischar(fileInfo), fileInfo.fName=fileInfo; end
    if ~isfield(fileInfo,'gName'), fileInfo.gName=[]; end
    if ~isfield(fileInfo,'fParams'), fileInfo.fParams=[]; end
    if ~isfield(fileInfo,'gParams'), fileInfo.gParams=[]; end
    if ~isfield(fileInfo,'nArgOut'), fileInfo.nArgOut=[]; end
end

%check if lb and ub are properly assigned
if ~(isempty(lb) | isempty(ub))
    %allow for them to be empty in case of restart
    lb=lb(:)'; ub=ub(:)'; %force row vector
    %check for consistency
    nVar=length(lb);
    if length(ub)~=nVar
        %chech vector sizing
        error('lb and ub must be the same size!')
    end
end

%track which options settings to override if restart is used.
%Also tracks which options were user-specified.
newOptions=options;
if isempty(options), newOptionsNames={};
```

```
else, newOptionsNames=fieldnames(options);
end

%fill in any missing options arguments
if isempty(options), options.maxfCount=200*nVar; end %need to initialize
    it
if ~isstruct(options), error('Options must be a structure array'), end
if ~isfield(options,'maxfCount'), options.maxfCount=200*nVar; end
if ~isfield(options,'maxIter'), options.maxIter=50; end
if ~isfield(options,'conTol'), options.conTol=1e-10; end
if ~isfield(options,'fTol'), options.fTol=1e-10; end
if ~isfield(options,'xTol'), options.xTol=0.001; end
if ~isfield(options,'display'), options.display=1; end
if ~isfield(options,'saveFile'), options.saveFile='noSave'; end
if ~isfield(options,'lgBalance'), options.lgBalance=1e-4; end
if ~isfield(options,'localSearch'), options.localSearch=0; end
if ~isfield(options,'termType'), options.termType=3; end
if ~isfield(options,'termParams'), options.termParams=[]; end

%adjust termination type according to what user specified
if any(strcmp(newOptionsNames,'maxfCount')) & ~any(strcmp(newOptionsNames
    ,'maxIter'))
    options.termType=1;
elseif ~any(strcmp(newOptionsNames,'maxfCount')) & any(strcmp(
    newOptionsNames,'maxfCount'))
    options.termType=2;
end

%fix any problems with file names in fileInfo or saveFile
if ~isempty(fileInfo)
    if strcmp(fileInfo.fName(end-1:end),'.m')
        fileInfo.fName=fileInfo.fName(1:end-2);
    end
    if ~isempty(fileInfo.gName)
```

```
        if strcmp(fileInfo.gName(end-1:end),'.m')
            fileInfo.gName=fileInfo.gName(1:end-2);
        end
    end
end
if ~isempty(options.saveFile)
    if strcmp(options.saveFile(end-3:end),'.mat')
        options.saveFile=options.saveFile(1:end-4);
    end
end

%initialize some variables
stopFlag=0;              %stop when stopFlag = 1
feasFlag=0;              %have we found a feasible point yet?
firstLocal=1;    %allow for an initial local search after 50 fn calls

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Internal Nomenclature
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nVar                   number of design variables
% nCon                   number of constraint functions
% iter                   number of iterations
% fCount                 number of function evaluations
% stopFlag               stop DIRECT and exit main "do_while" loop
% feasFlag               has a feasible point been found
% nRect                  number of rectangles
% currentSet                          set of rectangles selected for
    division
% fStar                  fmin - lgBalance     (used to calculate dist)
% j                      j = mod(sum(nDiv),nVar)  (used to calculate dist)
% k                      (sum(nDiv) - j)/nVar     (used to calculate dist)
% parent                 currently selected rectangle for trisection,
%                                     resulting in two "children"
```

```
% conViol                sum of constraint violations for a given
    rectangle
% oldfCount              temporary to store number of function calls
    before
%                                          current iteration
% childDist              Euclidean distance of child from parent
% divDimen               dimension along which to divide a rectangle


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Step 1 - Evaluate center of design space (or load prior results)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ~isempty(restartFile)
   %pick up from where DIRECT left off
    if ischar(restartFile)
       %reload prior DIRECT run results from file
       load(restartFile)
    elseif isstruct(restartFile)
       %the structure array was loaded directly
       results=restartFile;
    else
       %otherwise, the user screwed up
       error('restartFile must either be a structure array or a valid
           filename')
    end

   %pull out info from results structure array
    fileInfo=results.fileInfo;
    lb=results.lb;
    ub=results.ub;
    options=results.options;
    xBest=results.xBest;
    fBest=results.fBest;
```

```
    rect=results.rect;
    history=results.history;
    fCount=results.fCount;
    iter=results.iter+1;

    %determine some parameters from results
    nRect=size(rect.x,1);
    nVar=size(rect.x,2);
    nCon=size(rect.g,2);
    gBest=[];
    newBest=0;
    if ~isempty(history.fCount)
       %if history file isn't empty, a feasible pt has been found
      feasFlag=1;
   end
   if isempty(lb), lb=results.lb; end
   if isempty(ub), ub=results.ub; end

   %pull out more info from results for constrained problems
   if nCon>0
      gBest=results.gBest;
      fChangeRate=results.fChangeRate;
      gChangeRate=results.gChangeRate;
   end

   %override any options that were reset in defining the problem
   for i=1:length(newOptionsNames)
      options=setfield(options,newOptionsNames{i},...
         eval(['newOptions.',newOptionsNames{i}]));
   end

else
   %no resultsFile, so start from scratch...
```

```
   %first point is the design center point
   rect.x = lb + 0.5*(ub-lb);
   nRect = 1;

   %check out fName file to see if there are multiple outputs
   if isempty(fileInfo.nArgOut)
      fileInfo.nArgOut = nargout(fileInfo.fName);
      if fileInfo.nArgOut < 1
         error('Error executing function-calling file! Check file and re-
            run.');
      end
   end

   %evaluate first point
   [rect.f, rect.g] = evalFiles(fileInfo,rect.x);

   %count number of constraint functions
   nCon=length(rect.g);

   %check for consistency in conTol assignment
   if length(options.conTol)~=nCon
      if length(options.conTol)==1
         options.conTol=options.conTol*ones(1,nCon);
      else
         if nCon~=0
            error('length of conTol must agree with number of constraints
               ')
         end
      end
   end
   %force row orientation of conTol vector
   options.conTol=options.conTol(:)';

   %check if center point is feasible
```

```
if nCon>0
    if sum(rect.g>options.conTol)==0
        feasFlag=1;
    end
else
    %unconstrained, thus feasible
    feasFlag=1;
end

%track info on rectangle division
rect.dist=0.5*sqrt(nVar);
rect.nDiv=zeros(1,nVar);
rect.nDivAll=zeros(1,nVar);

%initialize some other variables
iter = 1;
fCount = 1;
fChangeRate = 0;
gChangeRate = zeros(1,nCon);

%start keeping track of history of best points
if feasFlag
    xBest=rect.x;
    fBest=rect.f;
    gBest=rect.g;
    history.fCount=1;
    history.iter=1;
else
    xBest=[];
    fBest=Inf; %give a value to show on display Step 5
    gBest=Inf; %give a value to show on display Step 5
    history.fCount=[];
    history.iter=[];
end
```

```
end

%initialize results structure array (to force display in a certain order)
results.xBest=[];
results.fBest=[];
if nCon>0, results.gBest=[]; end
results.fCount=[];
results.iter=[];
results.history=[];
results.fileInfo=fileInfo;
results.lb=lb;
results.ub=ub;
results.options=options;
results.rect=[];
if nCon>0
    results.fChangeRate=[];
    results.gChangeRate=[];
    results.conWeight=[];
end

%start sending updates to screen if requested to
if options.display>0
    disp('Iter     Total     Evals         fBest       max(g)')
    disp('         Evals    this iter')
    disp('-------------------------------------------------------------')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Begin main iteration loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%will terminate if EITHER fCount or iteration limit is reached
while stopFlag==0
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Step 2 - Select set of candidate rectangles for division
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if fCount == 1
    currentSet = [1]; % select only existing rect. for first iteration
else
    if nCon == 0
       %unconstrained problem
        currentSet = rectSelection(rect,fBest-options.lgBalance);
    else
       %constrained problem
        currentSet = constrSelection(rect,feasFlag,fBest-options.
            lgBalance);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Step 3 - Split candidates & evaluate centers of new rectangles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%track fCount at beginning of iteration and reset newBest
oldfCount = fCount;
newBest = 0;

for i=1:size(currentSet,1)

   %use this to break loop if searching for 1st feasible pt
    oldFeasFlag=feasFlag;

   % name "parent" rectangle index as current rectangle for trisection
    parent = currentSet(i);
   % find all longest sides of rectangle
    divDimen = find(rect.nDiv(parent,:)==min(rect.nDiv(parent,:)));
```

```
      if length(divDimen)>1
         % tied, so break by total number of divisions in each dim'n
          divDimen=divDimen(find(rect.nDivAll(divDimen)==min(rect.nDivAll(
             divDimen))));
          if length(divDimen)>1
              % still tied, so just use lowest dimension
              divDimen=min(divDimen);
          end
      end

      % split parent rectangle into two children
      delta = 3^(-(rect.nDiv(parent,divDimen)+1)); %fraction of range
      delta = delta*(ub(divDimen)-lb(divDimen)); %scale to true
         range

      xChild=ones(2,1)*rect.x(parent,:); % create children
      xChild(:,divDimen)=xChild(:,divDimen)+[1;-1]*delta; % offset
         children by delta

      % update rect.nDiv and rect.nDivAll
      rect.nDiv(parent, divDimen) = rect.nDiv(parent, divDimen) + 1;
      rect.nDivAll(divDimen) = rect.nDivAll(divDimen) + 1;

      % calculate new center-vertex distance (rect.dist) for parent
         rectangle
      j = mod(sum(rect.nDiv(parent,:)),nVar);
      k = (sum(rect.nDiv(parent,:)) - j)/nVar;
      rect.dist(parent) = 3^(-k)/2*(7/9+nVar-j)^0.5;

      for j=1:2

          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          % evaluate new rectangle center points
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
          %call function files
          [fChild,gChild]=evalFiles(fileInfo,xChild(j,:));

          %record new evaluations, "child" rectangle inherits rect.dist,
              rect.nDiv from "parent"
          rect.x=[rect.x; xChild(j,:)];
          rect.f=[rect.f; fChild];
          rect.g=[rect.g; gChild(:)'];
                rect.dist=[rect.dist ; rect.dist(parent)];
                rect.nDiv=[rect.nDiv ; rect.nDiv(parent,:)];

                % update rate of change variables here, if constrained
                if nCon > 0

                    if 0
                       %originally used this in updates below, but ChildDist ==
                           delta above
                       %
                       % find Euclidean distance of child from parent (midpoint
                           to midpoint)
                        childDist=0;
                        for k=1:nVar
                           childDist=childDist+(xChild(j,k)-rect.x(parent,k))^2;
                        end
                        childDist=sqrt(childDist);
                    end

                    % roc_obj update
                    childChangeRate = abs(fChild-rect.f(parent))/delta;
                    fChangeRate=fChangeRate+childChangeRate;

                    % roc update
                    for k=1:nCon
```

```
                childChangeRate = abs(gChild(k)-rect.g(parent,k))/delta;
                gChangeRate(k) = gChangeRate(k)+childChangeRate;
            end
        end

        %update fCount and check if there's a new best point
        fCount=fCount+1;
        if fChild<fBest
            if nCon==0 | sum(gChild>options.conTol)==0
                %we'll account for multiple global optima at the end
                history.fCount=[history.fCount ; fCount];
                history.iter=[history.iter ; iter];
                fBest = fChild;
                gBest = gChild(:)';
                xBest = xChild(j,:);
                newBest = 1;
                feasFlag = 1;
            end
        end

    end %exit loop for evaluating children of current parent

    if oldFeasFlag==0 & feasFlag==1
        break %stop this iteration if just looking for 1st feasible
            point
    end

end %exit loop for calling CurrentSet

fStar=fBest-options.lgBalance;

% update auxiliary function values if constrained version
if nCon > 0
```

```
      % update constraint weights
      for i=1:nCon
          conWeight(i)=fChangeRate/max(gChangeRate(i),10e-10);
      end

      % first, update parent rectangles
      for i=1:(fCount-oldfCount)/2

          % name "parent" rectangle index as current rectangle to update
          parent = currentSet(i);

          % sum constraint violations
          conViol=0;
          for j=1:nCon
              conViol=conViol+conWeight(j)*max([rect.g(parent,j) 0]);
          end
          rect.auxCon(parent) = conViol/rect.dist(parent);
          rect.aux(parent) = (max(rect.f(parent)-fStar,0)/rect.dist(parent
          ))+rect.auxCon(parent);
      end

      % next, update children, where there are 2 times as many children
          as parents
      for i=1:(fCount-oldfCount)

          % name "child" rectangle index as current rectangle to update
          child = oldfCount+i;

          % sum constraint violations
          conViol=0;
          for j=1:nCon
              conViol=conViol+conWeight(j)*max([rect.g(child,j) 0]);
          end
          rect.auxCon(child) = conViol/rect.dist(child);
```

```
          rect.aux(child) = (max(rect.f(child)-fStar,0)/rect.dist(child))+
              rect.auxCon(child);
      end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%  Step 4 - Check for termination of DIRECT
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  if options.termType==1
      %stop if fCount limit reached
      if fCount>=options.maxfCount
          stopFlag=1;
      end

  elseif options.termType==2
      %stop if iter limit reached
      if iter>=options.maxIter
          stopFlag=1;
      end

  elseif options.termType==3
      %stop if EITHER fCount or iter limit reached
      if (fCount>=options.maxfCount) | (iter>=options.maxIter)
          stopFlag=1;
      end

  elseif options.termType==4
      %stop when BOTH fCount and iter limit reached
      if (fCount>=options.maxfCount) & (iter>=options.maxIter)
          stopFlag=1;
      end
```

```
   elseif options.termType==5
      %stop if little improvement has been shown in last 100 fevals
      if ~isfield(options,'termParams'), options.termParams=[100 0.001]; 
   end
      if length(options.termParams)<2, options.termParams(2)=0.001; end
      %start checking once a feasible point is found
      if ~isempty(history.fCount)
         %start checking once taken 100 fn calls after first feasible pt
         if fCount>=options.termParams(1)+history.fCount(1)
            %best point 100 fn calls ago
            oldf=rect.f(history.fCount(max(find( ...
               history.fCount<=(fCount-options.termParams(1))))));
            %latest best point
            %keyboard
            newf=rect.f(history.fCount(end));
            %make sure we've made 0.1% relative improvement in last 100
   fn calls
               if newf>eps
                  if (oldf-newf)/newf < options.termParams(2)
                     stopFlag=1;
                  end
               else
                  if (oldf-newf) < options.termParams(2)
                     stopFlag=1;
                  end
               end
            end
         end

   elseif options.termType==6
      %stop if no improvement has been shown in last 10 iters
      if (iter-history.iter(end))>10
         stopFlag=1;
      end
```

```
end

%never exceed maximum allowable fCount
if fCount>=options.maxfCount
    stopFlag=1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Step 5 - Update results to screen and file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if options.display > 0
    disp(sprintf('%3.0i    (%4.0i)      %3.0i          %7.4f       %7.4f',
        ...
       iter,fCount,(fCount-oldfCount),fBest,max(gBest)))
end

% the below saves at every iteration if save file specified, or will
% save at end of run to default file name.
if (~strcmpi(options.saveFile,'noSave') | stopFlag)
   %collect results structure array
    results.xBest=xBest;
    results.fBest=fBest;
    results.gBest=[];
    results.rect=rect;
    results.history=history;
    results.fCount=fCount;
    results.iter=iter;
    if nCon > 0
        results.gBest=gBest;
        results.fChangeRate=fChangeRate;
        results.gChangeRate=gChangeRate;
```

```
        results.conWeight=conWeight;
    end
    %save results to .mat file if requested
    if ~strcmpi(options.saveFile,'noSave')
        save(options.saveFile,'results');
    end
end

% update iteration counter
iter=iter+1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Step 6 - Launch a local search if necessary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if options.localSearch & feasFlag & fCount>50 & (newBest|firstLocal)

    %set up SQP options using Matlab's optimset
    sqpOptions=optimset('Display','off','DiffMaxChange',1e-2*min(ub-lb)
    ,...
        'DiffMinChange',1e-5*min(ub-lb),'LargeScale','off','TypicalX',
    xBest(1,:),...
        'MaxFunEvals',nVar*100,'MaxIter',100);

    warning off
    if nCon==0
        %launch unconstrained local search from xBest
        if ~isempty(fileInfo.fParams)
            [xsqp,fsqp,sqpFlag,sqpOutput] = ...
                fminsearch(fileInfo.fName,xBest(1,:),sqpOptions,fileInfo.
    fParams{:});
        else
            [xsqp,fsqp,sqpFlag,sqpOutput] = ...
```

```
                fminsearch(fileInfo.fName,xBest(1,:),sqpOptions);
            end
        else
            %launch constrained local search from xBest
            if ~isempty(fileInfo.fParams)
                [xsqp,fsqp,sqpFlag,sqpOutput]=fmincon(fileInfo.fName,xBest
                    (1,:),...
                    [],[],[],[],lb,ub,[],sqpOptions,fileInfo.fParams{:});
            else
                [xsqp,fsqp,sqpFlag,sqpOutput]=fmincon(fileInfo.fName,xBest
                    (1,:),...
                    [],[],[],[],lb,ub,[],sqpOptions);
            end
        end
        warning backtrace

        %update fCount and iter
        fCount = fCount + sqpOutput.funcCount;
        iter = iter + 1;

        %check if there's a new best point
        if sqpFlag & fsqp<=fBest
            %history.fCount=[history.fCount ; fCount];
            %history.iter=[history.iter; iter-1];
            fBest = fsqp;
            if fsqp==fBest
                xBest=[xsqp; xBest];
            else
                xBest = xsqp;
            end
            feasFlag = 1;
        end

        if options.display > 0
```

```
        disp(sprintf('%3.0iL   (%4.0i)      %3.0i          %7.4f       %7.4
            f', ...
            iter-1,fCount,(sqpOutput.funcCount),fBest,max(gBest)))
    end

    %update fmin and other variables
    firstLocal = 0;

    %REMAINING WORK FOR LOCAL SEARCH TO BE READY:
    %  how to report results every ten fn calls?
    %  how to "count" fn calls for sake of history array?
    %  how to store pts that were run considering rect already exists?
    %        do we use a redundant structure array to store every
        function call?
    %        do we store just local search pts and when local search
        started?
    %  fmincon and fminsearch can still go into infinite loops thanks
        to Matlab
    %

  end %end local search if branch

end %end Master Loop of UMDIRECT

%use the post-processing code "findlocals" to distinguish all
% local optima within a given fTol and xTol of each other
if 0
    [xLocal] = findlocals(rect.x,rect.f,rect.g,options.xTol, ...
        options.fTol,options.conTol,lb,ub);
else
    xLocal=[];
end
results.xLocal=xLocal;
```

```
%send warning if couldn't find a feasible point
if ~feasFlag
   %disp('Warning!  No feasible point was found!')
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  QUESTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% RECENT CHANGES:
%        - added options.fTol and options.xTol to identify unique global
    optima
%
% QUESTIONS:
%    - how are linear vs. nonlinear constraints handled differently?
%           maybe use linearity for computing exact rate of change info
%
% TO DO LIST:
%        - add options.termParams
%        - fix local search (Step 6)
%        - enable DIRECT-local search possibilities


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%        SUBROUTINE FOR EVALUATING FUNCTIONS
    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [fout,gout]=evalFiles(fileInfo,x)

if fileInfo.nArgOut==1
```

```
␣␣␣%run␣objective␣function␣file
␣␣␣if␣~isempty(fileInfo.fParams)
␣␣␣␣␣␣%include␣fParams
␣␣␣␣␣␣fout=feval(fileInfo.fName,x,fileInfo.fParams{:});
␣␣␣else
␣␣␣␣␣␣fout=feval(fileInfo.fName,x);
␣␣␣end
␣␣␣%run␣constraint␣file
␣␣␣if␣~isempty(fileInfo.gName)
␣␣␣␣␣␣if␣~isempty(fileInfo.gParams)
␣␣␣␣␣␣␣␣␣%include␣gParams
␣␣␣␣␣␣␣␣␣gout=feval(fileInfo.gName,x,fileInfo.gParams{:});
␣␣␣␣␣␣else
␣␣␣␣␣␣␣␣␣gout=feval(fileInfo.gName,x);
␣␣␣␣␣␣end
␣␣␣else
␣␣␣␣␣␣gout=[];
␣␣␣end
else
␣␣␣%two␣outputs␣from␣fName␣file
␣␣␣if␣~isempty(fileInfo.fParams)
␣␣␣␣␣␣%include␣fParams
␣␣␣␣␣␣[fout,gout]=feval(fileInfo.fName,x,fileInfo.fParams{:});
␣␣␣else
␣␣␣␣␣␣[fout,gout]=feval(fileInfo.fName,x);
␣␣␣end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%␣SUBROUTINE␣FOR␣UNCONSTRAINED␣RECTANGLE␣SELECTION␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ␣%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [currentSet] = rectSelection(rect,fStar)
% last updated 1/26/02, MJS



% determine unique set of distances
distList = unique(rect.dist);

% determine best f value for any given distance
fBest=zeros(size(distList)); %initialize fBest to reduce overhead
for i = 1:length(distList)
    fBest(i) = min(rect.f(find(rect.dist == distList(i))));
end

% determine the lowest f value for each distance
setLowest = []; % initialize set of lowest values
multiplesSet = []; % initialize set to hold multiples
for i = 1:length(distList)
    j = find(rect.dist == distList(i) & rect.f == fBest(i));
    if length(j) > 1
        multiplesSet = [multiplesSet j'];
    end
    setLowest = [setLowest j(1)];
end
setLowest = [0 setLowest];

% compute the convex hull for the remaining rectangles
hull = grahamshull([0 distList'],[fStar fBest']);
currentSet = setLowest(hull);
currentSet = currentSet(2:end);
currentSet = union(currentSet,multiplesSet);
currentSet = currentSet(:);       %make column vector
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% grahamshull function %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function h = grahamshull(x,y)

% function h = grahamshull(x,y)
%
% grahamshull returns all points on the convex hull, even redundant ones.
% The vector x must be in ascending order.
%
% Based on the algorithm GRAHAMSHULL in:
% "Computational Geometry: An Introduction", Franco P. Preparata and
% Michael Ian Shamos, Springer-Verlag, New York, pp. 108-9, 1985.
%

x = x(:);
y = y(:);
m = length(x);
if length(y) ~= m
    error('Input dimensions must agree for convex hull')
end

% Index vector for points in convex hull
h = [1:m]';
if m < 3
   return;
end

v = 1;
flag = 0;


while (next(v,m) ~= 1) | (flag == 0)
```

```
    if next(v,m) == m
        flag = 1;
    end
    a = v;
    b = next(v,m);
    c = next(next(v,m),m);
    tol = 1e-6;
    if det([x(a) y(a) 1; x(b) y(b) 1; x(c) y(c) 1]) >= -tol
        v = next(v,m);
    else
        j = next(v,m);
        x = [x(1:j-1); x(j+1:m)];
        y = [y(1:j-1); y(j+1:m)];
        h = [h(1:j-1); h(j+1:m)];
        m = m - 1;
        v = pred(v,m);
    end
end

function i = next(v,m);
if v == m
    i = 1;
else
    i = v + 1;
end

function i = pred(v,m);
if v == 1
    i = m;
else
    i = v - 1;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SUBROUTINE FOR CONSTRAINED RECTANGLE SELECTION                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [currentSet] = constrSelection(rect, feasFlag, fStar)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%  constrSelection.m - a function to select rectangles for the constrained DIRECT algorithm
%
%  Last updated: 2/6/02 by J. Whitehead
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INTERNALLY USED VARIABLES:
% ----------------------------
%
% nVar                  number of design variables
% nCon                  number of constraint functions
% fCount                number of function calls made
% currentSet            set of rectangles selected for division
% feasFlag              has a feasible point been found
% fStar                 fmin - lgBalance    (used to calculate dist)
% j                     j = mod(nTri,nVar)  (used to calculate dist)
% k                     (nTri - j)/nVar     (used to calculate dist)
% stopFlag              stop DIRECT and exit main "do while" loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nRect = size(rect.x,1);
```

```
tieTol = abs(min([((1e-6)*fStar) 1e-6])); % relative tolerance to
    determine intersection ties
currentSet = []; % initialize set of selected
    rectangles
domin = zeros(1, nRect); % initialize rectangle dominance
    check variable
ties = []; % initialize ties vector

% if feasible, select rectangles by lower envelope to create 'currentSet'
if (feasFlag == 1)

   % select rectangle(s) with lowest rect.aux value as first rectangle
   % sorting takes too long. Find min value and all ties.
   minVal = min(rect.aux);
   for i=1:nRect
      if (rect.aux(i) <= (minVal + tieTol))
         currentSet = [currentSet; i];
         ties = [ties i];
      end
   end

   breakFlag = 0;
   maxIntersect = fStar;

   while breakFlag == 0

      % 1. if necessary, do tie-breaking to select which rectangle to
          follow, put that index in currRect
      if (length(ties) > 1)
         % select first rectangle in 'ties' vector
         incumbent = ties(1);
         ties(1) = [];

         for i=1:length(ties)
```

```
            % check if right side intersect of incumbent
            if ( maxIntersect > rect.f(incumbent) )

                % check if also right side intersect of other rect, 
                    otherwise follow incumbent
                if ( maxIntersect > rect.f(ties(i)) )
                    % check if other rectangle should be followed, instead 
                        of incumbent

                    % if other is further left, follow it
                    if ( rect.f(ties(i)) < rect.f(incumbent) )
                        incumbent = ties(i);

                    % special case, if both intersect on left and have 
                        same f-value
                    elseif ( rect.f(ties(i)) == rect.f(incumbent) )
                        if ( rect.dist(ties(i)) > rect.dist(incumbent) )
                            incumbent = ties(i);
                        end
                    end
                end

            else % is left side intersect of incumbent

                % if on right side of other rect, follow it
                if ( maxIntersect > rect.f(ties(i)) )
                    incumbent = ties(i);

                else % is on left side of other rect, so check slopes

                    if ( rect.dist(ties(i)) > rect.dist(incumbent) )
                        incumbent = ties(i);
                    end
                end
```

```
            end
        end
    else
        incumbent = ties(1);   % if none tied, pick only element in '
            ties' vector
    end
    %keyboard
    ties = [];
    currRect = incumbent;

    % 2. once selected one rectangle to follow, do dominance check to
        weed out others
    for i = 1:nRect

        % if already dominated, ignore
        if ( (domin(i) ~= 1) & (i ~= currRect) )

            if (rect.dist(currRect) >= rect.dist(i))
                if ( (rect.auxCon(currRect) <= rect.auxCon(i)) & (rect.f(
                    currRect) < rect.f(i)) )
                    domin(i) = 1; %rectangle i is dominated

                elseif ( (max([(rect.f(currRect) - rect.f(i)) 0]) / rect.
                    dist(currRect)) ...
                        < (rect.auxCon(i) - rect.auxCon(currRect)) )
                    domin(i) = 1; % rectangle 'i' is dominated, keep track
                        of it
                end
            end
        end
    end
```

```
      % 3. now check where all other rectangles intersect with sloped (
          left) side of current rectangle
      breakFlag = 1;
      intersect = [];
      for i = 1:nRect
          % if dominated, ignore
          if (domin(i) ~= 1) & (i ~= currRect)
              % calculate the f-star value for right and left side
                  intersection
              rSideInter = rect.f(currRect) +(rect.auxCon(currRect) - rect.
                  auxCon(i))*rect.dist(currRect);

              % for left side intersect, make sure not to get divide by
                  zero error!
              if (rect.dist(currRect) ~= rect.dist(i))
                  lSideInter = (rect.dist(currRect)*rect.dist(i)*(rect.
                      auxCon(i)-rect.auxCon(currRect))+rect.f(i)*rect.dist(currRect) - ...
                      rect.f(currRect)*rect.dist(i))/(rect.dist(currRect) -
                      rect.dist(i));
              else
                  lSideInter = -inf;
              end

              % check for false intersections
              if (rSideInter < rect.f(i)) | (rSideInter > rect.f(currRect))
                  | (rSideInter >= (maxIntersect-tieTol))
                  rSideInter = -inf;
              end
              if (lSideInter > rect.f(currRect)) | (lSideInter > rect.f(i))
                  | (lSideInter >= (maxIntersect-tieTol))
                  lSideInter = -inf;
              end
```

```
            % select closest intersection point for that rectangle, i.e.
                the maximum one
            intersect(i) = max(rSideInter, lSideInter);

            % if intersection is a 'true' intersection, make sure break
                flag is false
            if (intersect(i) > -inf)
                breakFlag = 0;
            end
        else
            intersect(i) = -inf;        % if dominated or current
                rectangle, set intersection to -inf
        end
    end
    if breakFlag == 0

        % 4. select intersection point "closest" to current point, find
            all ties, go to step 1.
        maxIntersect = max(intersect);
        for i = 1:nRect
            % ignore current rectangle
            if (i ~= currRect)

                % select as part of lower envelope all rectangles within
                    the tie tolerance
                if ( intersect(i) >= (maxIntersect - tieTol) )
                    currentSet = [currentSet; i];

                    % select only those rectangles which have exact same
                        intersect point for following along lower envelope
                    if ( intersect(i) == maxIntersect )
                        ties = [ties i];
                    end
                end
```

```
            end
         end
      end

   end % 'while breakFlag'


   % if not feasible, select the rectangle that minimizes the rate of
   % change necessary to bring the constraint violations down to zero
else
   % sorting takes too long. Find min value and all ties.
   minRect = min(rect.auxCon);
   for i=1:nRect
      if (rect.auxCon(i) <= (minRect + tieTol))
         currentSet = [currentSet; i];
      end
   end
end

currentSet = unique(currentSet);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot envelope of rectangles (for debugging)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ploton=0;
s = {'m-'; 'c-'; 'r-'; 'g-'; 'b-'; 'm:' ; 'c:'; 'r:'; 'g:' ; 'b:'};
if ploton==1
   clf
   hold on
   for i=1:nRect
      x=[(rect.f(i)-10) rect.f(i) (rect.f(i)+10)];
      y(1) = (10/rect.dist(i))+rect.auxCon(i);
      y(2) = rect.auxCon(i);
      y(3) = rect.auxCon(i);
```

```
      plot(x,y,s{(1+mod(i,9))})
      keyboard
   end
   hold off
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  SUBROUTINE FOR LOCATING ALL LOCAL OPTIMA FROM A GIVEN DATA SET AND TOL
    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [xLocal,sortID]=findlocals(x,f,g,xTol,fTol,gTol,lb,ub)

if nargin<8
   ub=[];
   if nargin<7
      lb=[];
      if nargin<6
         gTol=[];
         if nargin<5
            fTol=[];
            if nargin<4
               xTol=[];
               if nargin<3
                  g=[];
                  if nargin<2
                     error('Error!  Findlocals requires at least two
    arguments!')
                  end
               end
            end
         end
      end
```

```
    end
end

if isempty(xTol), xTol=0.01; end
if isempty(fTol), fTol=0.01; end
if isempty(gTol), gTol=1e-6; end
if isempty(lb), lb=min(x); end
if isempty(ub), ub=max(x); end

%find all feasible points
if isempty(g)
    feasID=[1:length(f)];
else
    feasID = find(all(g<=gTol,2));
end

%find best feasible point
fBest = min(f(feasID));

%find all feasible points within fTol of fBest
candidates = feasID(find(f(feasID)<=fBest+fTol*abs(fBest)));
[fSort,sortID] = sort(f(candidates));

%make ID number of original sample, not of candidates
sortID = candidates(sortID);

%find which of these candidates are xTol away from the nearest best
    candidate
%  by sweeping through their objective function values and checking
    distances
xLocal=x(sortID(1),:);

for i=2:length(candidates)
    %update current point to check
```

```
␣␣␣xCurrent=x(sortID(i),:);
␣␣␣xBetter=x(sortID(1:i-1),:);

␣␣␣%find␣distances␣to␣nearest␣known␣better␣solution
␣␣␣clear␣dist
␣␣␣for␣ii=1:size(xBetter,1)
␣␣␣␣␣␣dist(ii)=norm((xCurrent-xBetter(ii,:))./[ub-lb]);
␣␣␣end

␣␣␣%if␣it's the best point within xTol, call it a local optimum
   % also accounts for situations where two points have identical f
   if min(dist)>xTol | f(sortID(i))==f(sortID(i-1))
      xLocal = [xLocal ; xCurrent];
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Objective function example

```
function y = PrG9f(x,p)
% Matlab Code by A. Hedar (Nov. 23, 2005).
if size(x,2)~=7, x=x';␣end
if␣size(x,2)~=7,␣error('not a 7-D input vector!'),␣end
y␣=␣(x(:,1)-10).^2+5*(x(:,2)-12).^2+x(:,3).^4+3*(x(:,4)-11).^2+...
␣␣␣␣10*x(:,5).^6+7*x(:,6).^2+x(:,7).^4-4*x(:,6).*x(:,7)-10*x(:,6)-8*x
    (:,7);
end

function [ y ] = f_kriging_95( x, kparam )
[ynew, yvar, lambda] = f_predictkrige(x, kparam);
y = ynew -  abs(yvar)*1.96;% Kring max bias
% y = ynew +  abs(yvar)*1.96;
end
```

Constraints function example

```
function [y yeq] = PrG9c(x,p)
% Matlab Code by A. Hedar (Nov. 23, 2005).
% Constraints
v1 = 2*x(1)^2;
v2 = x(2)^2;
y(1) = v1+3*v2^2+x(3)+4*x(4)^2+5*x(5)-127;
y(2) = 7*x(1)+3*x(2)+10*x(3)^2+x(4)-x(5)-282;
y(3) = 23*x(1)+v2+6*x(6)^2-8*x(7)-196;
y(4) = 2*v1+v2-3*x(1)*x(2)+2*x(3)^2+5*x(6)-11*x(7);
% Variable lower bounds
%for j=1:7; y(j+4) = -x(j)-10; end
% Variable upper bounds
%for j=1:7; y(j+11) = x(j)-10; end
%
    ************************************************************************

y=y';
yeq = [];
end
```

No constraints function example

```
function [c,ceq] = nonlcon_11(x, p)
c = 0;%nonconstraint, so all constraint are zeros
ceq=[];
```

Personal Communication

姓名：王東泰

Name: Dung-Tai Wang

生日：1979年12月13日

Birthday : 1979/12/13

出生地：台灣台南

Birthplace: Tainan, Taiwan

學歷：

Education :
國立成功大學機械工程學系 肄業(2009,9∼2011,9)
Department of Mechanical Engineering at National Cheng-Kung University Bachelor
國立成功大學機械工程研究所 碩士班(2011,9∼2013,7)
Department of Mechanical Engineering at National Cheng-Kung University Master

通訊處：台南市東區崇善路841號5樓之一

Address : No.541, 5F-1, Chongshan Rd., East District, Tainan County , Taiwan (R.O.C.)

電話：0932770792

Telephone : 0932770792

E-mail：wangdungtai@gmail.com