Constrained Global Optimization with Radial Basis Functions

Jan-Erik Käck

Department of Mathematics and Physics
Mälardalen University
P.O. Box 883
SE-721 23 Västerås, Sweden

Research Report MdH-IMa-2004

September 10, 2004

Abstract

Response surface methods show promising results for global optimization of costly non convex objective functions, i.e. the problem of finding the global minimum when there are several local minima and each function value takes considerable CPU time to compute. Such problems often arise in industrial and financial applications, where a function value could be a result of a time-consuming computer simulation or optimization. Derivatives are most often hard to obtain. The problem is here extended with linear and nonlinear constraints, and the nonlinear constraints can be costly or not. A new algorithm that handles the constraints, based on radial basis functions (RBF), and that preserves the convergence proof of the original RBF algorithm is presented. The algorithm takes advantage of the optimization algorithms in the Tomlab optimization environment (www.tomlab.biz). Numerical results are presented for standard test problems.

Contents

1 Introduction

The research in global optimization is becoming more and more important. One of the main reasons for this is that the use of computer models and simulations are becoming more common in, for instance, the industry. It is often advantageous to use a computer algorithm to modify these models instead of doing it by hand. Since the computer simulations often are time consuming it is vital that the modifying algorithm doesn't run the simulation an unnecessary amount of times. There are several good methods available to handle this problem. The problem is that several models also have constraints, which might be time consuming to evaluate as well. The user is in these cases almost always forced to construct a penalized model (i.e. objective function), where the violation of the constraint is added to the objective function thereby enforcing a penalty. This is the case, especially when using response surface methods (for instance RBF and Kriging methods) to evaluate the simulated model. This approach is far from the best. This paper introduces a new and much simpler way to handle the constraints by *cyclic infeasibility tolerances*. The method explained in this paper is intended for the RBF algorithm explained in Section 3, but can easily be modified to work in almost any response surface setting.

2 The Problem

The formulation of a global constrained optimization problem is given in equation 1, where both linear and non-linear constraints are present. This paper introduces a new framework on how to deal with both costly and non-costly constraints when solving the problem with a radial basis function interpolation solver (denoted RBF solver). The problem, which is defined as

$$\begin{array}{llccc} min & f(\mathbf{x}) & & & \\ s.t & & & & \\ & \mathbf{b}_L & \leq & A\mathbf{x} & \leq & \mathbf{b}_U \\ & \mathbf{c}_L & \leq & \mathbf{c}(\mathbf{x}) & \leq & \mathbf{c}_U \\ & & & \mathbf{x} \in \Omega & & \end{array} \tag{1}$$

has a costly and black-box objective function $f(\mathbf{x})$, i.e. it takes much CPU-time to compute it and no information but the function value for each point $\mathbf{x}$ is available. A is the coefficient matrix of the linear constraints. $\mathbf{b}_L$ and $\mathbf{b}_U$ are bounding vectors for the linear constraints. Ω is the set of values that $\mathbf{x}$ is allowed to take. Normally this is the bounding box $\Omega = \{\mathbf{x} | \mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U\}$. The non-linear constraint functions $\mathbf{c}(\mathbf{x})$ can be either costly or non-costly and black-box or not. By identifying the costly functions as $\mathbf{c}_c(\mathbf{x})$ and the non costly as $\mathbf{c}_{nc}(\mathbf{x})$ so that $\mathbf{c}^T(\mathbf{x}) = [\mathbf{c}_c^T(\mathbf{x}), \mathbf{c}_{nc}^T(\mathbf{x})]$ it is possible to reformulate the problem as

$$\begin{array}{llccc} min & f(\mathbf{x}) & & & \\ s.t & & & & \\ & \mathbf{c}_{c_L} & \leq & \mathbf{c}_c(\mathbf{x}) & \leq & \mathbf{c}_{c_U} \\ & & & \mathbf{x} \in D_0 & & \end{array} \tag{2}$$

where D_0 is the set restricted by all simple (e.g. non costly) constraints. This means that $D_0 = \{\mathbf{x} | \mathbf{b}_L \leq A\mathbf{x} \leq \mathbf{b}_U, \mathbf{c}_{nc_L} \leq \mathbf{c}_{nc}(\mathbf{x}) \leq \mathbf{c}_{nc_U}\} \bigcap \Omega$. For simplicity the costly non-linear constraints will be referred to as $\mathbf{c}(\mathbf{x})$ and the full set of non linear constraints as $\mathbf{c}_f(\mathbf{x})$. The simple constraints will be dealt with in the next section, which gives an overview of the basic RBF algorithm. Section 4 and 5 will then treat the costly non-linear constraints.

3 Basic RBF Algorithm

The RBF algorithm used in this paper, and presented in this section, was first presented by Gutmann [3], and implemented by Björkman, Holmström [2]. The overview given in this section is mostly taken from the Björkman, Holmström paper [2], with the major modification being that the points $\mathbf{x}$ are allowed to reside in D_0 instead of Ω.

Suppose that the objective function f has been evaluated at n different points $\mathbf{x}_1, ..., \mathbf{x}_n \in D_0$ (where D_0 is the set bounded by simple constraints), with $F_i = f(\mathbf{x}_i)$, $i = 1, ..., n$. Consider the question of deciding the next point $\mathbf{x}_{n+1}$ where to evaluate the objective function. The idea of the RBF algorithm is to compute a radial basis function (RBF) interpolant, s_n, to f at the points $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n$ and then determine the new point $\mathbf{x}_{n+1}$, where the objective function f should be evaluated by minimizing a utility function $g_n(\mathbf{y})$ which takes less CPU-time to compute than the original objective function f. The radial basis function interpolant s_n is on the form

$$s_n(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \phi\left(\|\mathbf{x} - \mathbf{x}_i\|_2\right) + \mathbf{b}^T\mathbf{x} + a, \tag{3}$$

with $\lambda_1, ..., \lambda_n \in \mathbb{R}$, $\mathbf{b} \in \mathbb{R}^d$, $a \in \mathbb{R}$ and ϕ is either cubic with $\phi(r) = r^3$ (denoted $rbfType = 2$) or the thin plate spline $\phi(r) = r^2 \log r$ (denoted $rbfType = 1$).

Now consider the system of linear equations

$$\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}, \tag{4}$$

where Φ is the $n \times n$ matrix with $\Phi_{ij} = \phi\left(\|\mathbf{x}_i - \mathbf{x}_j\|_2\right)$ and

$$P = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ . & . \\ . & . \\ \mathbf{x}_n^T & 1 \end{pmatrix}, \lambda = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ . \\ . \\ \lambda_n \end{pmatrix}, c = \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ b_d \\ a \end{pmatrix}, F = \begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ . \\ . \\ f(\mathbf{x}_n) \end{pmatrix}. \tag{5}$$

The matrix

$$\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \tag{6}$$

is nonsingular if the rank of P is $d + 1$. So if the points $\mathbf{x}_i$ are not situated on one line, the linear system of equations (4) has a unique solution. Thus a unique radial basis function interpolant to f at the points $\mathbf{x}_1, ..., \mathbf{x}_n$ exists. Furthermore, it is the "smoothest" (see Powell [7]), function s from a linear space that satisfies the interpolation conditions

$$s(\mathbf{x}_i) = f(\mathbf{x}_i), \quad i = 1, ..., n. \tag{7}$$

The new point $\mathbf{x}_{n+1}$ is calculated such that it is the value of $\mathbf{y}$ that minimizes the utility function $g_n(\mathbf{y})$. $g_n(\mathbf{y})$ is defined as

$$g_n(\mathbf{y}) = \mu_n(\mathbf{y})\left[s_n(\mathbf{y}) - f_n^*\right]^2, \quad \mathbf{y} \in D_0 \setminus \{\mathbf{x}_1, ..., \mathbf{x}_n\}, \tag{8}$$

where f_n^* is a target value and $\mu_n(\mathbf{y})$ is the coefficient corresponding to $\mathbf{y}$ of the Lagrangian function L that satisfies $L(\mathbf{x}_i) = 0$, $i = 1, ..., n$ and $L(\mathbf{y}) = 1$. The coefficient $\mu_n(\mathbf{y})$ is computed by first extending the Φ matrix to

$$\Phi_y = \begin{pmatrix} \Phi & \phi_y \\ \phi_y^T & 0 \end{pmatrix}, \tag{9}$$

where $(\phi_y)_i = \phi(\|\mathbf{y} - \mathbf{x}_i\|_2)$, $i = 1, ..., n$. Then the P matrix is extended to

$$P_y = \begin{pmatrix} P \\ \mathbf{y}^T \quad 1 \end{pmatrix}, \tag{10}$$

and the system of linear equations

$$\begin{pmatrix} \Phi_y & P_y \\ P_y^T & 0 \end{pmatrix} \mathbf{v} = \begin{pmatrix} 0_n \\ 1 \\ 0_{d+1} \end{pmatrix}, \tag{11}$$

is solved for $\mathbf{v}$. (The notation 0_n and 0_{d+1} for column vectors with all entries equal to zero and with dimension n and $(d+1)$ respectively). $\mu_n(\mathbf{y})$ is set to the $(n+1)$:th component in the solution vector v which means $\mu_n(\mathbf{y}) = v_{n+1}$. The computation of $\mu_n(\mathbf{y})$ must be performed for many different $\mathbf{y}$ when minimizing $g_n(\mathbf{y})$ so it does not make sense to solve the full system each time. Instead, it is possible to factorize the interpolation matrix and then update the factorization for each $\mathbf{y}$. For the value of f_n^* it should hold that

$$f_n^* \in \left(-\infty, \min_{\mathbf{y}\in D_0} s_n(\mathbf{y})\right]. \tag{12}$$

The case $f_n^* = \min_{\mathbf{y}\in D_0} s_n(\mathbf{y})$ is only admissible if $\min_{\mathbf{y}\in D_0} s_n(\mathbf{y}) < s_n(\mathbf{x}_i)$, $i = 1, ..., n$. There are two special cases for the choice of f_n^*. In the case when $f_n^* = \min_{\mathbf{y}\in D_0} s_n(\mathbf{y})$, then minimizing (8) will be equivalent to

$$\min_{\mathbf{y}\in D_0} s_n(\mathbf{y}). \tag{13}$$

In the case when $f_n^* = -\infty$ then minimizing (8) will be equivalent to

$$\min_{\mathbf{y}\in D_0\setminus\{\mathbf{x}_1,...,\mathbf{x}_n\}} \mu_n(\mathbf{y}). \tag{14}$$

In [4], Gutmann describes two different strategies for the choice of f_n^*. They are here described with the modification of handling non costly constraints.

The first strategy (denoted $idea = 1$) is to perform a cycle of length $N + 1$ and choose f_n^* as

$$f_n^* = \min_{\mathbf{y}\in D_0} s_n(\mathbf{y}) - \left[\frac{(N - (n - n_{init})) \mod (N+1)}{N}\right]^2 \left(\max_i f(\mathbf{x}_i) - \min_{\mathbf{y}\in D_0} s_n(\mathbf{y})\right), \tag{15}$$

where n_{init} is the number of initial points. Here, $N = 5$ is fixed and $\max_i f(\mathbf{x}_i)$ is not taken over all points. In the first stage of the cycle, it is taken over all points, but in each of the subsequent steps the $n - n_{max}$ points with largest function value are removed (not considered) when taking the max. So the quantity $\max_i f(\mathbf{x}_i)$ is decreasing until the cycle is over and then all points are considered again and the cycle starts from the beginning. So if $(n - n_{init}) \mod (N + 1) = 0$, $n_{max} = n$, otherwise

$$n_{max} = \max\{2, n_{max} - \text{floor}((n - n_{init})/N)\}. \tag{16}$$

The second strategy (denoted $idea = 2$) is to consider f_n^* as the optimal value of

$$\begin{array}{ll} \min & f^*(\mathbf{y}) \\ \text{s.t.} & \mu_n(\mathbf{y})\left[s_n(\mathbf{y}) - f^*(\mathbf{y})\right]^2 \le \alpha_n^2 \\ & \mathbf{y} \in D_0, \end{array} \tag{17}$$

and then perform a cycle of length $N + 1$ on the choice of α_n. Here, $N = 3$ is fixed and

$$\begin{aligned} \alpha_n &= \tfrac{1}{2}\left(\max_i f(\mathbf{x}_i) - \min_{\mathbf{y}\in D_0} s_n(\mathbf{y})\right), \quad n = n_0, n_0+1 \\ \alpha_{n_0+2} &= \min\left\{1, \tfrac{1}{2}\left(\max_i f(\mathbf{x}_i) - \min_{\mathbf{y}\in D_0} s_n(\mathbf{y})\right)\right\} \\ \alpha_{n_0+3} &= 0, \end{aligned} \tag{18}$$

where n_0 is set to n at the beginning of each cycle. For this second strategy, $\max\limits_i f(\mathbf{x}_i)$ is taken over all points in all parts of the cycle.

When there are large differences between function values, the interpolator has a tendency to oscillate strongly. To handle this problem, large function values can in each iteration be replaced by the median of all computed function values (denoted $REPLACE = 1$).

The function $h_n(\mathbf{x})$ defined by

$$\begin{cases} \frac{1}{g_n(\mathbf{x})}, & \mathbf{x} \notin \{\mathbf{x}_1, ..., \mathbf{x}_n\} \\ 0, & x \in \{\mathbf{x}_1, ..., \mathbf{x}_n\} \end{cases}, \tag{19}$$

is differentiable everywhere so instead of minimizing $g_n(\mathbf{y})$ it is better to minimize $-h_n(\mathbf{y})$. The step of the cycle is denoted $modN$ and can take a value from $\mathbb{N} \bigcap [0, N]$.

3.1 Some Implementation Details

The subproblem

$$\min_{\mathbf{y}\in D_0} \quad s_n(\mathbf{y}) \tag{20}$$

is itself a problem which could have more than one local minima. Eq. (20) is solved by taking the interpolation point with the least function value i.e. $\arg\min f(\mathbf{x}_i)$ $i = 1, ..., n$, as start point and then perform a local search. In many cases this leads to the minimum of s_n. Of course, there is no guarantee that it does. Analytical expressions for the derivatives of s_n is used in the sub-optimizer. To minimize $g_n(\mathbf{y})$ for the first strategy, or $f^*(\mathbf{y})$ for the second strategy, a global sub-optimizer is needed. In this paper the Matlab routine *glcFast*, which implements the DIRECT algorithm [6], is used. This algorithm can easily handle the cheap constraints imposed by using D_0 instead of Ω as the space over which the utility functions are minimized. The sub-optimization is run for 1000 evaluations and chooses $\mathbf{x}_{n+1}$ as the best point found. When $(n - n_{init}) \mod (N+1) = N$ (when a purely local search is performed) and the minimizer of s_n is not too close to any of the interpolation points, the global sub-optimizer is not run to minimize $g_n(\mathbf{y})$ or $f^*(\mathbf{y})$. Instead, the minimizer of (20) is selected as the new point $\mathbf{x}_{n+1}$.

3.2 Starting Points

It is not clear how the initial points should be selected. The RBF algorithm in Tomlab implements several different ideas. These are.

1. **Corners**. Select the corners of the box as starting points. This gives 2^d starting points.
2. **Dace Init**. Latin hypercube space-filling design.
3. **Random percentage**. Randomly distribute the points in the box, but never let them get to close.
4. **Gutmann strategy**. Initial points are $\mathbf{x}_L$ and d points $\mathbf{x}_L + (x_{Ui} - x_{Li}) * \mathbf{e}_i, i = 1, ..., d$

This paper uses the second strategy to generate starting points. The reason for this is that it is not random (same starting points are generated every time), and the points are distributed evenly over the space.

4 Penalty Approach to Costly Constraints

The most straightforward (from an initial point of view) way of extending the RBF algorithm to handle costly constraints is to transform the problem into an unconstrained penalized form. This was done by Bjrkman, Holmstrm in [1] and is also discussed by Sasena in [8].

In order to form a penalized objective function the inequalities in Equation 2 are rewritten into

$$\begin{array}{rcl} \mathbf{c}(\mathbf{x}) - \mathbf{c}_U & \leq & 0, \\ \mathbf{c}_L - \mathbf{c}(\mathbf{x}) & \leq & 0. \end{array} \tag{21}$$

It is then possible to form $\hat{\mathbf{c}}(\mathbf{x})$ as

$$\hat{\mathbf{c}}(\mathbf{x}) \quad = \quad \left(\begin{array}{c} \mathbf{c}(\mathbf{x}) - \mathbf{c}_U \\ \mathbf{c}_L - \mathbf{c}(\mathbf{x}) \end{array} \right), \tag{22}$$

which is a function from $\mathbb{R}^d$ to $\mathbb{R}^{2m}$. $\hat{\mathbf{c}}(\mathbf{x})$ is said to be the *infeasibility measure* for each constraint in a point $\mathbf{x}$. A point $\mathbf{x}$ is then feasible if $\hat{c}_i(\mathbf{x}) \leq 0 \ \forall i \in [1, 2m]$. The point $\mathbf{x}$ is said to be infeasible if $\exists i | \hat{c}_i(\mathbf{x}) > 0, i \in [1, 2m]$. These conclusions make it possible to form the *sum of constraint violations* function $C_s(\mathbf{x})$ as

$$C_s(\mathbf{x}) \quad = \quad \textstyle\sum_{i=1}^{2m} \max(0, \hat{c}_i(\mathbf{x})). \tag{23}$$

It is also possible to define the *least feasible constraint* $C_a(\mathbf{x})$,

$$C_a(\mathbf{x}) \quad = \quad \max(\hat{c}_i(\mathbf{x})). \tag{24}$$

$C_s(\mathbf{x})$ is larger than zero if the point is infeasible and equal to zero if the point is feasible. $C_a(\mathbf{x})$ is larger than zero if the point is infeasible and less than or equal to zero if the point is feasible. C_s and C_a are compared in figure 1 for test problem 18 (see Appendix A for details).

Keep in mind that the discussion here is only in respect to the costly non linear constraints. For the point to actually be feasible it is required that it not only fulfills these requirements, but also that it is located within D_0.

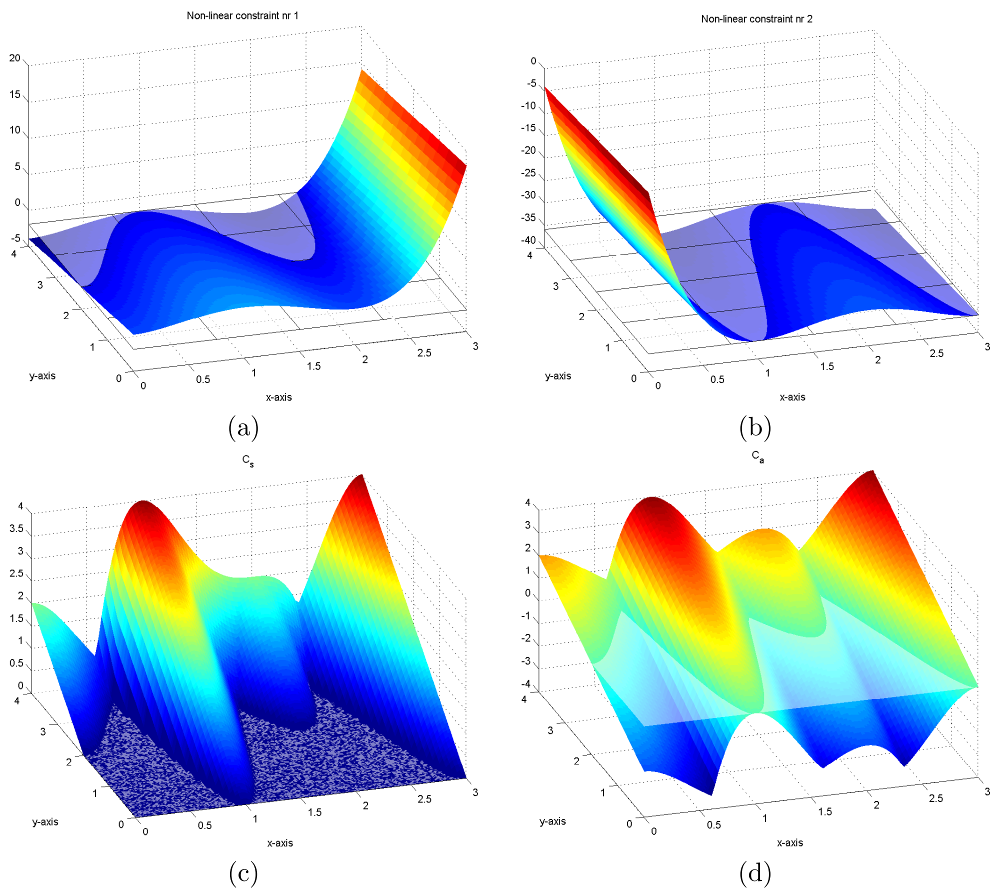


Figure 1: (a) and (b) shows the non-linear constraints with their lower bounds as flat semi-transparent surfaces. (c) shows C_s for these constraints and (d) shows C_a.

4.1 Smoothness

No matter how the penalty function is constructed some conditions must be fulfilled. Since the idea of the response surface methods is that the optimization (in each iteration) should be done on some sort of interpolation surface instead of on the real objective function, the penalty should be added to the interpolation surface. It is therefore necessary to construct this penalty in such a way that the interpolation does not fail or is forced into unnecessary heavy oscillation. To avoid these problems it is necessary that the following rule is followed when constructing the penalized function values: *A penalized function value added to the response surface must be constructed in such a way that it gives the smoothest possible surface, at the same time as it imposes a penalty on the surface.* This is in a sense self-explanatory, but it is still good to keep it in mind.

4.2 The Constraints Violation Function

The function $C_s(\mathbf{x})$ from equation 23 is understandably of great importance, and therefore in need of some special attention.

There is no guarantee that $C_s(\mathbf{x})$ is scaled in the same order as the objective function. This can cause difficulties if, for instance, the infeasible points get such a high penalized value that the quote $\frac{|f_{Pen}(\mathbf{x}_{feasible})|}{|f_{Pen}(\mathbf{x}_{infeasible})|} << 1$ for all possible choices of $\mathbf{x}_{feasible}$ and $\mathbf{x}_{infeasible}$. Not even if each individual constraint is scaled in the same order as the objective function is it possible to say that $C_s(\mathbf{x})$ is scaled correctly. This is because $C_s(\mathbf{x})$ is the sum of all constraint violations, and can therefore, in worst case, take a maximum value of

$$\max_{\mathbf{x}\in\mathbf{X}}(C_s(\mathbf{x})) = \sum_{i=1}^{2m} \max(0, \hat{c}_i(\mathbf{x})) \le 2m * \max(f(\mathbf{x})), \tag{25}$$

if $\max_{i,\mathbf{x}}(\hat{c}_i(\mathbf{x})) = \max_{x} f(\mathbf{x})$. If the set of all evaluated points X, is divided into two sets $X = S_f \bigcup S_i$, where $S_f = \{\mathbf{x}|C_s(\mathbf{x}) \le 0\}$ and $S_i = \{\mathbf{x}|C_s(\mathbf{x}) > 0\}$ it is possible to solve this problem by first formulating the *normalized relative constraint violation function*

$$nC_s(\mathbf{x}) = \frac{C_s(\mathbf{x}) - \min_{\mathbf{y}\in S_i} C_s(\mathbf{y})}{\max_{\mathbf{y}\in S_i} C_s(\mathbf{y}) - \min_{\mathbf{y}\in S_i} C_s(\mathbf{y})} \in [0, 1]. \tag{26}$$

Observe that $\min_{\mathbf{x}\in S_i} C_s(\mathbf{x}) > 0$. This means that the most feasible of the infeasible points has a nC_s value equal to 0. This can be useful, but can be avoided by formulating

$$nC_{s+}(\mathbf{x}) = \frac{C_s(\mathbf{x})}{\max_{\mathbf{y}\in S_i}(C_s(\mathbf{y}))} \in (0, 1], \tag{27}$$

the *normalized constraint violation function*. The next step is to scale the violation function correctly (i.e. in the same order as the objective function). This is done by first computing

$$dF = \max_{\mathbf{x}\in\mathbf{X}}(f(\mathbf{x})) - \min_{\mathbf{x}\in\mathbf{X}}(f(\mathbf{x})) \tag{28}$$

and then multiplying $nC_s(\mathbf{x})$ and $nC_{s+}(\mathbf{x})$ respectively by dF

$$\begin{aligned} dC_s(\mathbf{x}) &= nC_s(\mathbf{x}) * dF \in [0, dF], \\ dC_{s+}(\mathbf{x}) &= nC_{s+}(\mathbf{x}) * dF \in (0, dF]. \end{aligned} \tag{29}$$

By doing these steps a scaled *relative* constraint violation function is obtained in $dC_s(\mathbf{x})$ and a scaled constraint violation function in $dC_{s+}(\mathbf{x})$.

The second attribute of the constraint violation function that needs careful attention is the discontinuity of the gradient at the feasibility border. In order to test how well RBF handles a non smooth surface a test was performed on the objective function

$$f(x) = \begin{cases} \frac{x}{0.2} & 0 \le x \le 0.2 \\ -10x + 3 & 0.2 < x \le 0.25 \\ 5x - 0.75 & 0.25 < x \le 0.3 \\ 1.25x + 0.375 & 0.3 < x \le 0.5 \\ \frac{-10x}{3} + \frac{8}{3} & 0.5 < x \le 0.8 \\ 10x - 8 & 0.8 < x \le 0.85 \\ 0.5 & 0.85 < x \le 1, \end{cases} \tag{30}$$

which can be seen in figure 2. Figure 3 and 4 shows what RBF does when it is allowed to freely chose where to evaluate the objective function. It is clear that RBF can handle this type of non-smooth functions, but it requires a large amount of function evaluations in order to accurately interpolate the non-smooth points. It is therefore likely that any penalty method will produce poor results (high number of objective function evaluations). This was also the conclusion of the work done in

[8] by Sasena. Some tests has still been performed as a part of this work, these are discussed in the following section.

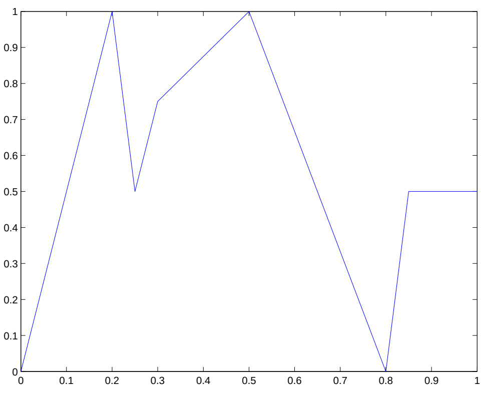
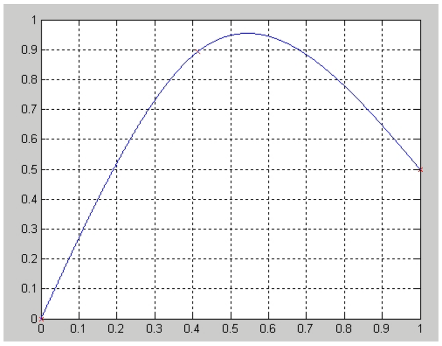
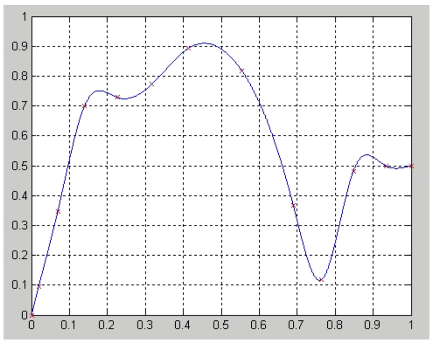
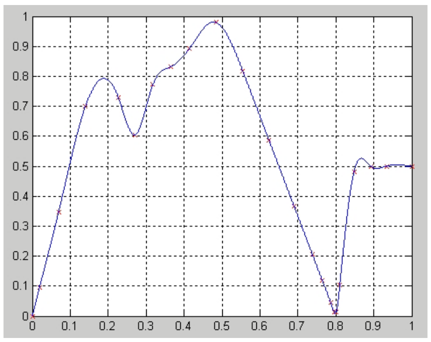
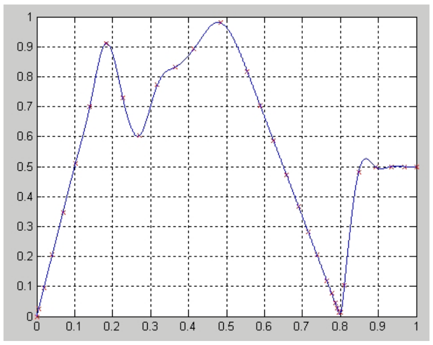


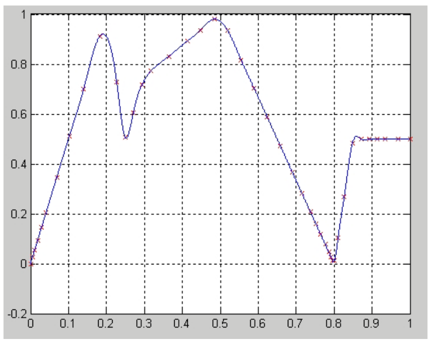
Figure 2: Non-smooth test surface.


(a)


(b)


(c)


(d)


(e)

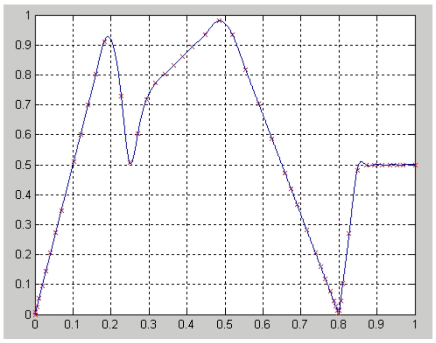

(f)

Figure 3: RBF interpolations of test surface. (a): Initial interpolation (three points evaluated). (b)-(f): Interpolation surfaces with an increase of ten evaluated points per picture.

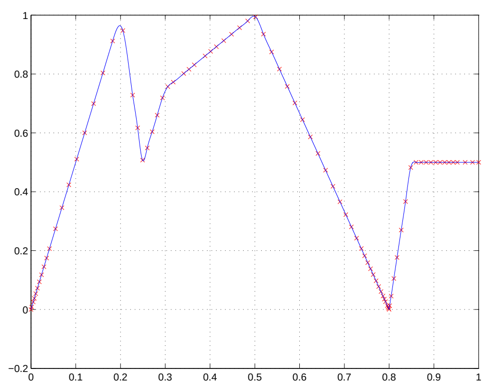


Figure 4: "Final" RBF interpolation of fig. 2 (with 90 interpolation points chosen by RBF).

4.3 Weighting strategies

The number of possible ways to create a weighted penalty is enormous. It is therefore possible to write large amount of papers on this subject alone. Therefore no comprehensive study on this subject is done here. Instead a general weigh is discussed.

4.3.1 Standard way of adding penalty

The general penalized form

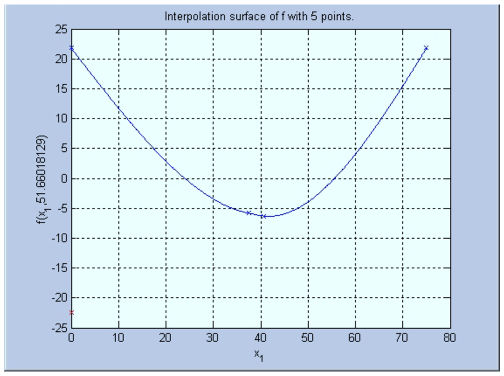
$$f(\mathbf{x}) + \sum_{i=1}^{m} \omega_i * \max(0, c_i(\mathbf{x})), \tag{31}$$

can be rewritten in the form introduced in Section 4.2 to

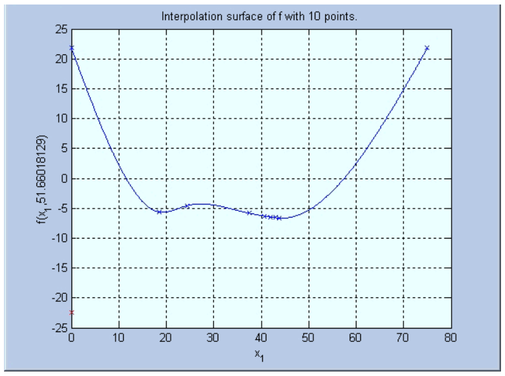
$$f(\mathbf{x}) + \omega * dC_s(\mathbf{x}). \tag{32}$$

The later form implements the individual weights of eq. (31) automatically by scaling the constraints against each other (see eq. 29). The parameter ω is, in the most simple form, set to 1.

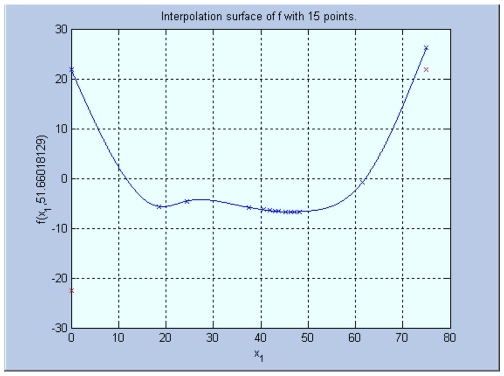
Recall Equation (26) on which $dC_s(\mathbf{x})$ is based. This equation states that the most feasible, of the infeasible points will have a $dC_s(\mathbf{x})$ value equal to 0. This means that no penalty is given to the evaluated point(s) which violates the constraints the least. This increases the chances of getting a smooth surface near the boundaries, but it might also cause a large amount of infeasible points being evaluated. Examples on how an interpolation surface can look, using Eq. (32), is given in figure 5 for a one dimensional test problem with optimum located near 13.55, and where the RBF algorithm in each step chooses where to evaluate the next point. The line is the interpolation surface, the crosses on that line is the penalized function values, and the crosses not on the line is the actual objective function values. In order to make the interpolation surface smoother the large objective function values can be replaced with the median (see Section 3). The corresponding images of the interpolation surface are shown in figure 6. Even in the later figures it is evident that the steep hill of the interpolation surface, caused by the penalty, just next to the optimum is intractable. It is not clear how to force the algorithm to go just as far up this hill as necessary.



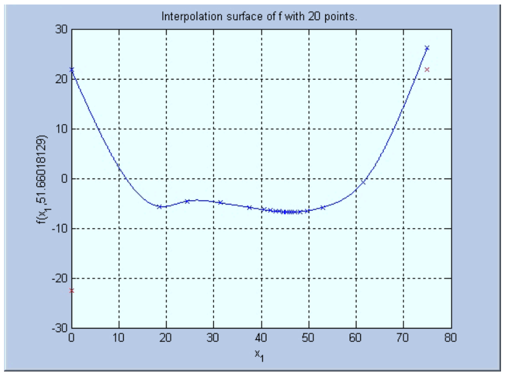
(a)



(b)

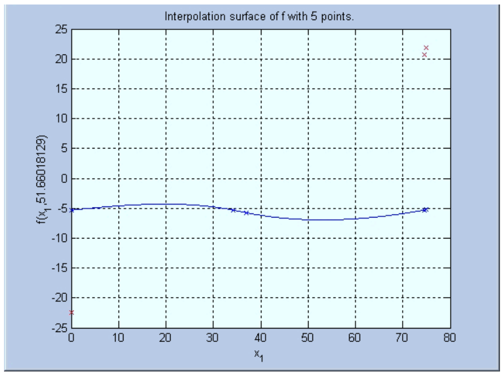


(c)

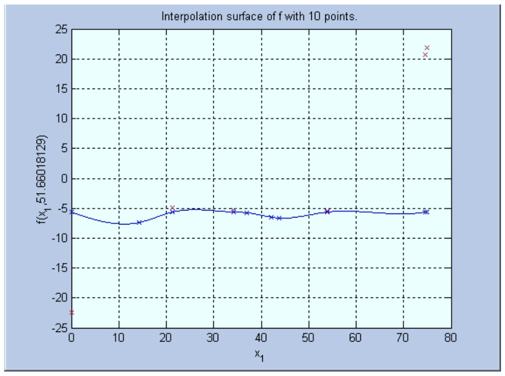


(d)

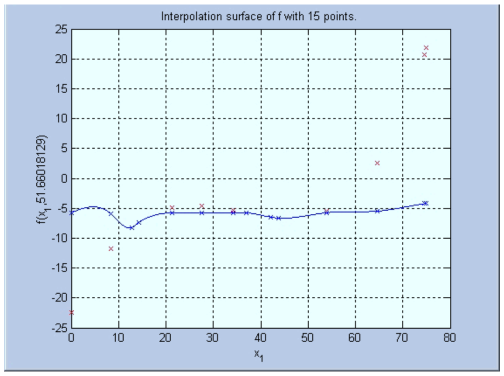
Figure 5: Interpolated one dimensional surface with five interpolated points (a), ten interpolated points (b), 15 interpolated points (c) and 20 interpolated points (d).



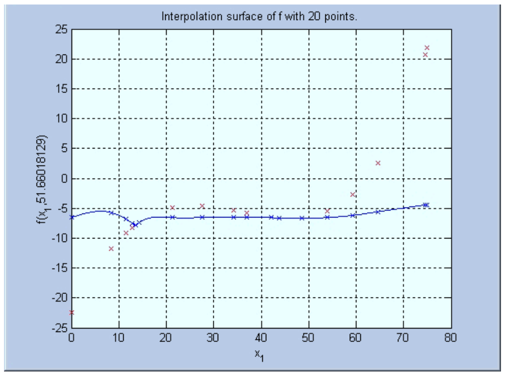
(a)



(b)



(c)



(d)

Figure 6: Interpolated one dimensional surface with five interpolated points (a), ten interpolated points (b), 15 interpolated points (c) and 20 interpolated points (d), with $REPLACE = 1$

4.3.2 Weighted infeasibility distance

In order to make the area near the border more tractable it is possible to take into consideration the distance from an infeasible point to the closest feasible point. By weighting this distance with the constraint violation it is possible to reduce the penalty near the border. Although if this is made in an inappropriate way, it is very likely that an unnecessary high amount of infeasible points will be evaluated. The Weighted penalty can be written

$$w_i = \alpha d_{k_i} + \beta dC_s(\mathbf{x}_i), \tag{33}$$

where

$$\begin{aligned} k_i &= \arg\min_{\mathbf{x}_j \in S_f} (||\mathbf{x}_i - \mathbf{x}_j||_2), \mathbf{x} \in S_i, \\ d_{k_i} &= \frac{||\mathbf{x}_i - \mathbf{x}_{k_i}||_2 - \min\limits_{\mathbf{x}_l \in S_i} \min\limits_{\mathbf{x}_m \in S_f} (||\mathbf{x}_l - \mathbf{x}_m||_2)}{\max\limits_{\mathbf{x}_l \in S_i} \min\limits_{\mathbf{x}_m \in S_f} (||\mathbf{x}_l - \mathbf{x}_m||_2) - \min\limits_{\mathbf{x}_l \in S_i} \min\limits_{\mathbf{x}_m \in S_f} (||\mathbf{x}_l - \mathbf{x}_m||_2)}, \end{aligned} \tag{34}$$

and α and β are weights that is used to compute the penalized function value. The problem is then how to select these two parameters, which is left out of this work.

4.4 Convergence

Since the Problem that is being discussed in this paper is of black box type it is not possible to give any proof that the algorithm has converged in finite time. Therefore the convergence discussed here is rather of the kind that when infinitely many points has been evaluated, the entire surface is known, and therefore the optimum has been found. The penalty approach is guaranteed to converge in this sense, if the response surface method used has a convergence proof and that the penalized function fulfills the requirements of this convergence proof. The proof of this is simple and therefore left out. For the RBF algorithm used here a convergence proof exists for a general objective function, and therefore also for the penalized objective function.

4.5 Conclusions on Penalty Methods

No promising results has been obtained with the penalty methods discussed in this paper. It has been showed that the penalized interpolation surface obtains an intractable behavior. No results for different values on the weighting parameters has been presented here but several strategies has been tested, but the results has been very problem dependent. The most significant advantage of the penalty methods is that an unconstrained problem can be solved instead of the original constrained. The conclusion is to not use penalty methods. In consequence of this the rest of this paper will discuss different ways of interpolating the constraints. No results will be presented for the penalty methods in the Result section.

5 Response Surface Modelled Constraints

Since the penalty method has been discarded some other strategy has to be developed, and as discussed earlier in Section 2 it is possible to forward the constraints to the global and local sub-problem optimizers. Doing this will cause a substantial amount of evaluations of the constraints, and is therefore not possible if the constraints are costly. Instead a constraint interpolation function, based on one or several RBF surfaces, can be used in the sub-optimizers as constraint function.

5.1 Single-Surface Approach

In Section 4 two different constraint violation functions were presented (C_a and C_s). It is evident that $C_s(\mathbf{x})$ from Equation (23) is a bad function to create an interpolation of the constraints from since it is flat (with a value of zero) for all feasible points, and it is very likely that the interpolation will differ from zero due to oscillation. It is also non-smooth on the feasibility border, which makes the border hard to interpolate correctly within a reasonable time. $C_a(\mathbf{x})$, from Equation (24), on the other hand gives the least feasible constraint value in every point, thus making it possible to interpolate a *least feasible constraint function*. This function is never flat, and it is continuous $\forall \mathbf{x} \in \Omega$, given that the constraints are non-flat and always continuous (The interpolation might still become flat, because of poorly chosen interpolation points). The problem with $C_a(\mathbf{x})$ is that its gradient is non-continuous in the points where the least feasible constraint is changed. These non-smooth points will be denoted *corners* from here on. An example of such a corner is shown in figure 7.

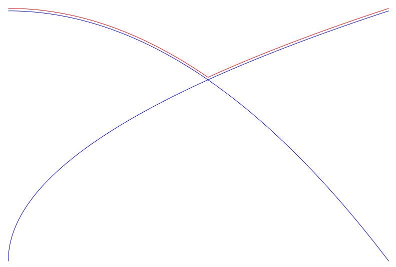


Figure 7: Example of a corner. The double line is how the new constraint surface ($C_a(x)$) would look.

RBF can handle this type of non-smooth surfaces, as already stated in Section 4.2, but it requires a lot of evaluations of the constraints in order to get the interpolation perfect, or at least good enough. This does not have to cause any trouble, as long as the interpolation function does not hide feasible points due to bad interpolation. A special case, where this happens is the Equality Constraint problems, where all feasible points, and thus also the optimum, is situated on the corners. An example of this is given in figure 8. And as long as the feasible point(s) has not been identified, the constraint interpolation is very likely to over estimate the constraint values, and it is therefore very likely that the interpolated surface will not come down to zero at the right point in a surveyable time.

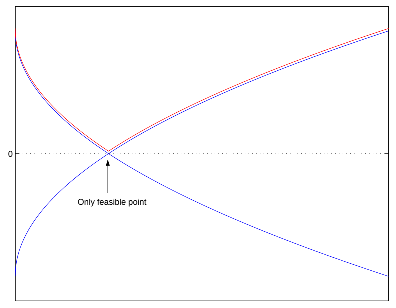


Figure 8: Example of a corner for a equality constraint problem. The double line is how the new constraint surface would look.

5.1.1 Tolerance cycling

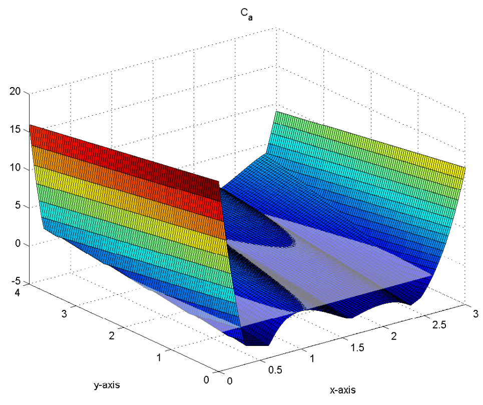
Since it is not possible to know how the constraints behave it is necessary that all points are evaluated to preserve convergence as discussed in section 4.4. This can be done by performing some iterations unconstrained. Using the single surface approach discussed here, based on C_a, this can be achieved by setting the allowed constraint violation parameter $cTol$ of the sub optimizers to different levels in different iterations, thereby obtaining a cycling of tolerance. Several such cycling strategies can be used. 3 different strategies are discussed in this paper for the algorithm presented in Section 6. The first strategy ($cTolStrat == 0$) is to set the tolerance equal to 10^{-3} for all iterations without any cycling. It goes without saying that if this strategy is used convergence properties, as they are explained above and in Section 4.4, can not hold. The second strategy ($cTolStrat == 1$) is to set the tolerance equal to $\max C_a$ for all iterations without any cycling. The local searches are still performed with tolerance equal to 10^{-3}. The third strategy ($cTolStrat == 2$) is to set the tolerance equal to

$$cTol \quad = \quad \begin{cases} \infty & \text{if } modN = 0 \\ (1 - \frac{modN}{N}) \max(C_a) + \frac{modN}{N} 10^{-3} & otherwise \end{cases} \tag{35}$$

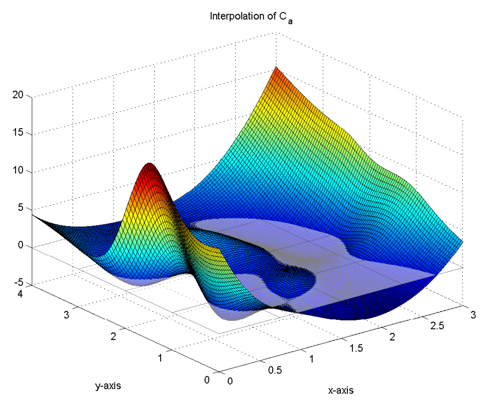
which cycles the tolerance accordingly to the chosen cycle strategy in the RBF algorithm and with the pure global search of the RBF algorithm fully unconstrained and the pure local search is performed with the assumption that the constraint interpolation is perfect.

5.2 Multi-Surface Approach

Instead of having a single interpolation function to represent all the constraints, it could be advantageous to interpolate each constraint separately. This requires of course more computer power than other "weighted" approaches. The advantage is that the multi-surface approach does not have any corners (unless any of the original constraints has corners), which is the main drawback of the single-surface approach. It is therefore more likely to be able to solve, for instance, equality constrained problems. A comparison between the single-surface and the multi-surface approach is given in figure 9 and 10 for Problem 18 (see Appendix A for details on the problem) with added upper bounds on the constraints $\mathbf{c}_U = [5, -20]^T$.



(a)



(b)

Figure 9: (a) Plot of C_a. (b) Interpolation surface of C_a with 21 interpolation points.

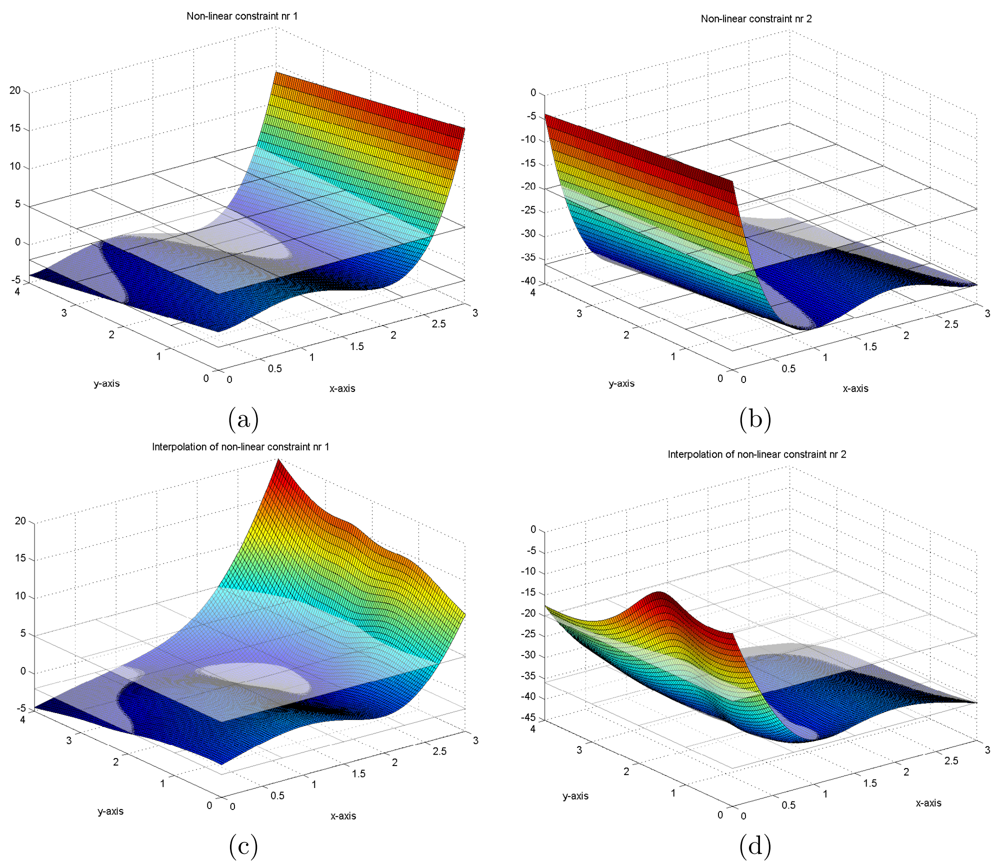


Figure 10: (a) Plot of non-linear constraint 1. (b) Plot of non-linear constraint 2. (c) Interpolation surface of non-linear constraint 1 with 21 interpolation points. (d) Interpolation surface of non-linear constraint 2 with 21 interpolation points.

By comparing the figures it is evident that the multi surface approach gives a better resemblance with the original surfaces than the single surface approach gives of C_a.

Just as in the case of the single-surface approach some constraint tolerance shifting has to be applied in order to maintain convergence properties. Such a shifting has to be applied to every constraint function, and both bounds of each constraint function. So that

$$\begin{array}{rcl} \tilde{\mathbf{c}}_L & = & \mathbf{c}_L - \mathbf{c}_{L_{Tol}}, \\ \tilde{\mathbf{c}}_U & = & \mathbf{c}_U + \mathbf{c}_{U_{Tol}} \end{array} \tag{36}$$

becomes the new bounds, and are used by the sub-optimizers. Here

$$\mathbf{c}_{L_{Tol}} \in \mathbb{R}^m_+, \mathbf{c}_{U_{Tol}} \in \mathbb{R}^m_+. \tag{37}$$

5.2.1 Tolerance Cycling

In order to keep global convergence properties and increase speed of feasible local convergence, the vectors $\mathbf{c}_{\mathbf{L_{Tol}}}$ and $\mathbf{c}_{\mathbf{U_{Tol}}}$ has to be changed between iterations so that sub optimization is performed both constrained and unconstrained. These changes is cycled so that every N:th iteration is performed unconstrained. The tolerance is then decreased so that the $2kN-1$ iteration is performed fully constrained, for some N and $k = 1, 2, 3, \ldots$. The cycling is performed in conjunction with the already built in global/local cycling mentioned in Section 3 of RBF. The bounds can thus be written

$$\begin{aligned}
\tilde{\mathbf{c}}_L &= \begin{cases} -\infty & \text{if } modN = 0 \\ \mathbf{c}_L - (1 - \frac{modN}{N}) * \mathbf{M}_L & \text{otherwise} \end{cases}, \\
\tilde{\mathbf{c}}_U &= \begin{cases} \infty & \text{if } modN = 0 \\ \mathbf{c}_U + (1 - \frac{modN}{N}) * \mathbf{M}_U & \text{otherwise} \end{cases}.
\end{aligned} \tag{38}$$

Here $\mathbf{M}_L$ and $\mathbf{M}_U$ are taken to be

$$\mathbf{M}_L = \begin{pmatrix} \max CvL_1 \\ \max CvL_2 \\ . \\ . \\ . \\ \max CvL_m \end{pmatrix}, \quad \mathbf{M}_U = \begin{pmatrix} \max CvU_1 \\ \max CvU_2 \\ . \\ . \\ . \\ \max CvU_m \end{pmatrix} \tag{39}$$

where CvL and CvU are

$$\begin{aligned}
CvL &= \begin{pmatrix} \max(0, c_{L_1} - c_1(\mathbf{x}_1)) & \max(0, c_{L_1} - c_1(\mathbf{x}_2)) & \cdots & \max(0, c_{L_1} - c_1(\mathbf{x}_n)) \\ \max(0, c_{L_2} - c_2(\mathbf{x}_1)) & \max(0, c_{L_2} - c_2(\mathbf{x}_2)) & \cdots & \max(0, c_{L_2} - c_2(\mathbf{x}_n)) \\ \vdots & \vdots & \ddots & \\ \max(0, c_{L_m} - c_m(\mathbf{x}_1)) & \max(0, c_{L_m} - c_m(\mathbf{x}_2)) & \cdots & \max(0, c_{L_m} - c_m(\mathbf{x}_n)) \end{pmatrix}, \\
CvU &= \begin{pmatrix} \max(0, c_1(\mathbf{x}_1) - c_{U_1}) & \max(0, c_1(\mathbf{x}_2) - c_{U_1}) & \cdots & \max(0, c_1(\mathbf{x}_n) - c_{U_1}) \\ \max(0, c_2(\mathbf{x}_1) - c_{U_2}) & \max(0, c_2(\mathbf{x}_2) - c_{U_2}) & \cdots & \max(0, c_2(\mathbf{x}_n) - c_{U_2}) \\ \vdots & \vdots & \ddots & \\ \max(0, c_m(\mathbf{x}_1) - c_{U_m}) & \max(0, c_m(\mathbf{x}_2) - c_{U_m}) & \cdots & \max(0, c_m(\mathbf{x}_n) - c_{U_m}) \end{pmatrix}.
\end{aligned} \tag{40}$$

vectors with the violations of lower and upper bound j, respectively, for every evaluated point.

5.3 Convergence

The discussion on convergence given in Section 4.4 no longer holds since the constraints "cuts" away parts of the problem space. This is not a problem in itself, since infeasible parts are to be cut away. The problem lies in the fact that a bad interpolation of the constraints very well might cut away feasible parts, and even the optimum, thus preventing the algorithm from converging. In order to restore guarantee of convergence some iterations has to be performed without applying any interpolated constraints. For the RBF algorithm this means that the space may become dense, thereby maintaining the original convergence proof. (The problem becomes unconstrained, for which convergence proof exists.) So by applying the tolerance cycling discussed in Section 5.2.1 convergence can be guaranteed.

6 The Algorithm

The algorithms presented here, and used to develop the work presented in this paper, is based on the RBF algorithm presented by Gutmann [4] and implemented in Matlab by Björkman and

H

$Continue$	=	1 if main loop continues, 0 if algorithm finished.
$Idea$	=	Explained in Section 3.
$Feasible$	=	1 if feasible solution found, 0 if not.

Table 1: Algorithmic flags

Holmström [2]. A brief discussion about the main ideas in the RBF algorithm is given in Section 3. In this section the complete constrained algorithm is presented in Pseudo-Matlab code.

Three different algorithms are presented. The first one demonstrates how RBF is modified to handle non costly constraints. The second algorithm uses the single surface approach, and the third one uses the multi surface approach. The split up has been done for readability and the actual algorithm incorporates all three versions in the same code.

A number of flags are used in the algorithm, these are explained in table 1

Algorithm 1 No costly constraints

```
Create initial Points $X = (\mathbf{x}_1, \mathbf{x}_2, ...)$
Compute $F = (f(\mathbf{x}_1)f(\mathbf{x}_2)...)$
Compute the sum of the violation of the linear constraints $(fPen)$
Add the sum of the violation of the non-linear constraints to $fPen$
Create the interpolation surface out of $F$
Initiate the global and local sub-solvers to use the original linear constraints as linear constraints.
Add the original non-linear constraints to the global and local sub-solvers as non-linear constraints.
$Continue = 1$
while $Continue == 1$
  Solve $\min S_n(\mathbf{y})$
  if $Idea == 1$
    Compute $f_n^*$
    Solve $\min g_n(\mathbf{y})$
  else if $Idea == 2$
    Compute $\alpha_n$
    Solve $\min f^*(\mathbf{y})$
  end
  Compute the objective function at the new point
  Update $S_n$
  Compute the sum of the constraint violations for the linear constraints $(fPen)$
  Add the sum of the constraint violations of the non-linear constraints to $fPen$
  if $fPen < violTol$
    $Feasible = 1$
    Update $fMin$ if necessary
  end
  if $Feasible == 1$
    check convergence
    if convergence
      $Continue = 0$
    end
  end
  check maximum Iteration and maximum evaluations of objective function
  if maximum reached
    $Continue = 0$
  end
end
```

Algorithm 2 Single Constraint Interpolation Surface

```
Create initial Points $X = (\mathbf{x}_1, \mathbf{x}_2, ...)$
Compute $F = (f(\mathbf{x}_1) f(\mathbf{x}_2)...)$
Compute the sum of the violation of the linear constraints ($fPen$)
Compute $C_a = [C_a(\mathbf{x}_1), C_a(\mathbf{x}_2)...]$
Create interpolation surface out of $C_a$ ($S_{n_C}$)
Create the interpolation surface out of $F$
Initiate the global and local sub-solvers to use the original linear constraints as linear constraints.
Add $S_{n_C}$ to global and local sub-solvers as non-linear constraint
Set $Feasible = 0$
$Continue = 1$
while $Continue == 1$
  Set $cTol$ in global solver accordingly to $cTolStrat$
  Solve $\min S_n(\mathbf{y})$
  if $Idea == 1$
    Compute $f_n^*$
    Solve $\min g_n(\mathbf{y})$
  else if $Idea == 2$
    Compute $\alpha_n$
    Solve $\min f^*(\mathbf{y})$
  end
  Compute the objective function at the new point
  Update $S_n$
  Compute the sum of the constraint violations for the linear constraints ($fPen$)
  Compute $C_a$ for the new point
  Update $S_{n_C}$
  if $fPen < violTol$
    if $C_a < cTol$
      $Feasible = 1$
      Update $fMin$ if necessary
    end
  end
  if $Feasible == 1$
    check convergence
    if convergence
      $Continue = 0$
    end
  end
  check maximum Iteration and maximum evaluations of objective function
  if maximum reached
    $Continue = 0$
  end
end
```

Algorithm 3 Multiple Constraint Interpolation Surfaces

```
Create initial Points $X = (\mathbf{x}_1, \mathbf{x}_2, ...)$
Compute $F = (f(\mathbf{x}_1)f(\mathbf{x}_2)...)$
Compute the sum of the violation of the linear constraints ($fPen$)
Add the sum of the violation of the non-linear constraints to $fPen$
Create the interpolation surface out of $F$
Initiate the global and local sub-solvers to use the original linear constraints as linear constraints.
Create interpolation surfaces for each non-linear constraint
Add interpolation surfaces to global and local sub-solvers as non-linear constraints
Set $Feasible = 0$
$Continue = 1$
while $Continue == 1$
  Compute bounds for non-linear constraints as shown in eq. (38)
  Solve $\min S_n(\mathbf{y})$
  if $Idea == 1$
    Compute $f_n^*$
    Solve $\min g_n(\mathbf{y})$
  else if $Idea == 2$
    Compute $\alpha_n$
    Solve $\min f^*(\mathbf{y})$
  end
  Compute the objective function at the new point
  Update $S_n$
  Compute the sum of the constraint violations for the linear constraints ($fPen$)
  Add the sum of the constraint violations of the non-linear constraints to $fPen$
  Update interpolation surfaces for the non-linear constraints.
  if $fPen < violTol$
    $Feasible = 1$
    Update $fMin$ if necessary
  end
  if $Feasible == 1$
    check convergence
    if convergence
      $Continue = 0$
    end
  end
  check maximum Iteration and maximum evaluations of objective function
  if maximum reached
    $Continue = 0$
  end
end
```

7 Numerical Results

The numerical results from performed tests are summarized in three bar graphs. Each bar is the logarithm of the quote between the number of function evaluations that DIRECT requires and the amount that the specific RBF implementation requires to solve the specified problem to required accuracy. The different RBF implementations are

No Costly Constraints:	This means that all constraints of the test problems are forwarded to the sub optimizers. (Algorithm 1)
Single Interpolation Surface:	The non linear constraints are treated as costly constraints and C_a is used to construct a constraint interpolation. (algorithm 2)
Multiple Interpolation Surfaces:	The non linear constraints are treated as costly constraints and are interpolated separately. (3)

All RBF implementations uses idea 1 and cubic splines interpolation ($rbfType == 2$) in these tests. The actual numerical results for each problem and algorithm are given in appendix B. A summary of the test problems is given in table 2, and the full statement of the problems is given in Appendix A.

Table 2: Summary of used test problems. The different numbers means the number of instances. For instance the numbers in column integers means the number of integer variables, and Lower bounds means the number of constraints with lower bounds.

Number	Name	Dim.	Linear constraints			Non linear constraints				Integers
			Rows	Lower bounds	Upper bounds	Equations	Lower bounds	Upper bounds	Equality constraints	
1	Gomez 2	2	0			1	0	1	0	0
2	Gomez 3	2	0			1	0	1	0	0
3	Hock-Schittkowski 59	2	0			3	3	0	0	0
4	Hock-Schittkowski 65	3	0			1	1	0	0	0
7	Schittkowski 234	2	0			1	1	0	0	0
8	Schittkowski 236	2	0			2	2	0	0	0
9	Schittkowski 237	2	0			3	3	0	0	0
10	Schittkowski 239	2	0			1	1	0	0	0
12	Schittkowski 332	2	0			1	1	1	0	0
13	Schittkowski 343	3	0			2	2	0	0	0
15	Floudas-Pardalos 3.3 TP2	5	0			3	3	3	0	0
16	Floudas-Pardalos 3.4 TP3	6	3	1	3	2	2	0	0	0
17	Floudas-Pardalos 3.5 TP4	3	2	0	2	1	1	0	0	0
18	Floudas-Pardalos 4.10 TP9	2	0			2	2	0	0	0
19	Floudas-Pardalos 12.2 TP1	5	3	0	3	2	2	2	2	3
21	Floudas-Pardalos 12.2 TP3	7	5	0	5	4	0	4	0	4
23	Floudas-Pardalos 12.2 TP5	2	3	0	3	1	0	1	0	2
24	Floudas-Pardalos 12.2 TP6	2	2	0	2	1	0	1	0	1

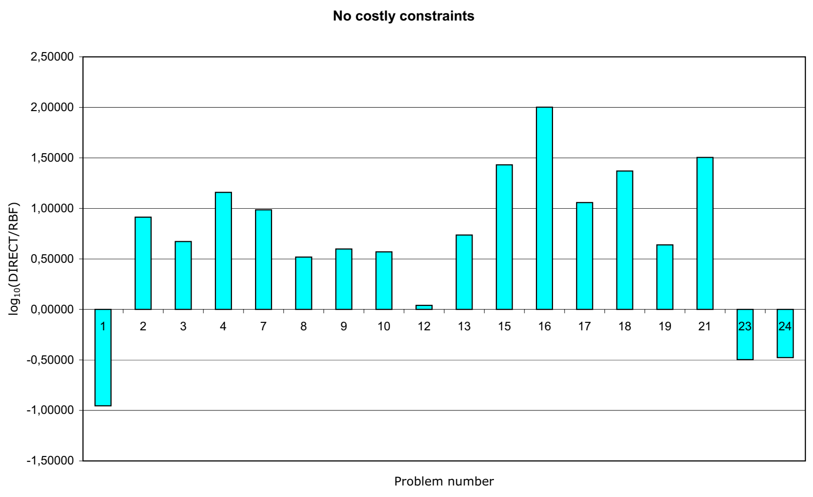


Figure 11: $\log_{10}$ of the quote between the number of function evaluations that DIRECT requires and RBF requires in order to locate the known global minimizers on the test problems with RBF assuming that all constraints are cheap.

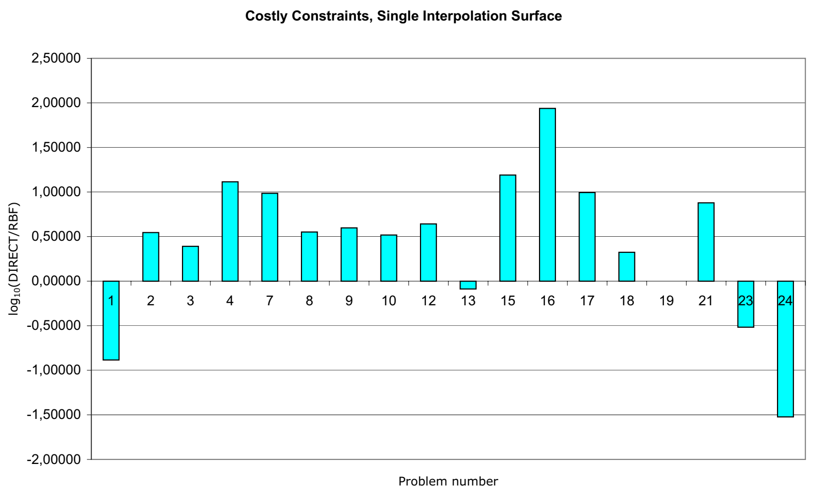


Figure 12: $\log_{10}$ of the quote between the number of function evaluations that DIRECT requires and RBF requires in order to locate the known global minimizers on the test problems with RBF assuming that the non linear constraints are costly and using the single weighted constraint interpolation surface approach.

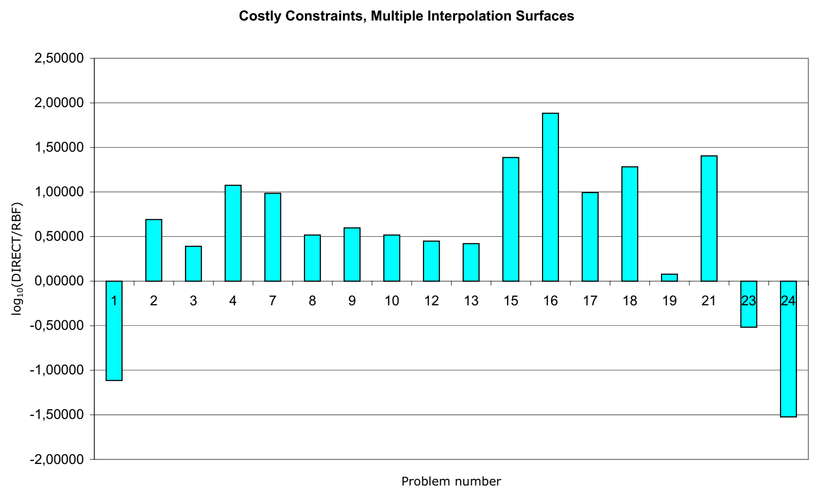


Figure 13: $\log_{10}$ of the quote between the number of function evaluations that DIRECT requires and RBF requires in order to locate the known global minimizers on the test problems with RBF assuming that the non linear constraints are costly and using the multiple constraint interpolation surfaces approach.

The No costly constraints graph can be looked upon as a reference set since it does not handle costly constraints and treats all constraints as non costly, and thereby has a huge advantage in comparison to the other implementations. Even thou it has this advantage it is not that much better than the other two implementations.

For the Single Interpolation Surface implementation it becomes clear from problem 19 that it can not handle equality constraints. The algorithm fails after a few iterations and no result can be presented for this problem.

When doing a comparison between The Single Interpolation Surface and the Multiple Interpolation Surfaces implementations it is difficult to see which one to prefer. They are both, in most cases, at least three times better than DIRECT (ie. DIRECT requires three times more function evaluations to solve the problem than RBF needs). The Multiple Interpolation Surfaces implementation seems more stable in its performance thou, and it can handle equality constraints.

In order to understand what happens with, for instance, problem 1, where DIRECT seems to be almost ten times better than RBF it is necessary to recall that RBF requires 21 evaluations in order to initiate the algorithm for a two dimensional problem. It then becomes clear by studying the problem and the numerical results (Appendix A and B) that DIRECT finds the optimum by accident (first evaluated point) and RBF after just a few iterations. It is possible to do similar investigations for problem 23 and 24. The results from these two investigations differ. Problem 23 has a similar result to problem 1, whereas RBF can't solve problem 24.

8 Conclusions and Further Work

The multi surface extension to RBF presented in this paper in order to handle constraints gives promising results requiring down to 1/100 of the number of function evaluations that DIRECT requires. Performed tests shows that it is not likely that a penalty approach to constraint handling will be able to give equally good results. Non of these tests are presented in this paper. The extension does not add any extra algorithm parameters for the user to decide the setting of. Even if it is clear that the bounds on the interpolations of the constraints need to be cycled it is still needed to be investigated how to best chose these bounds in each cycle step.

One issue that has not been discussed at all in this paper is the possibility that the objective function is undefined for infeasible points. This would impose an extra problem to the response surfaces since it is required that the algorithm is aware of all evaluated points (so that they are not evaluated again). At the same time it is not possible to use these infeasible points when constructing the interpolation surface of the objective function since it is undefined. The solution is to use different set of points when constructing interpolation surface of the objective function (reduced set) and for the constraint functions (full set). It is also required to use the full set of points when computing μ_n (see Section 3), so that the same points does not get chosen for evaluation more than once.

The RBF algorithm still has a lot of parameters for the user to set, work is being undertaken on this issue in order to reduce the number of parameters, or at least create a decision tree to aid the user.

Acknowledgements

First, I would like to thank Dr. Hans-Martin Gutmann for his valuable comments on this paper and details of the RBF algorithm. I would also like to thank professor Kenneth Holmstrm for his support on the Tomlab environment.

References

[1] M. Björkman and K. Holmström. Global Optimization Using the DIRECT Algorithm in Matlab. *Advanced Modeling and Optimization*, 1(2):17–37, 1999.

[2] M. Björkman and K. Holmström. Global Optimization of Costly Nonconvex Functions Using Radial Basis Functions. *Optimization and Engineering*, 1(4):373–397, 2000.

[3] Hans-Martin Gutmann. A radial basis function method for global optimization. Technical Report DAMTP 1999/NA22, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1999.

[4] Hans-Martin Gutmann. *Radial Basis Function Methods for Global Optimization*. PhD Thesis, University of Cambridge, 2001.

[5] K. Holmström. The TOMLAB Optimization Environment in Matlab. *Advanced Modeling and Optimization*, 1(1):47–69, 1999.

[6] Donald R. Jones. DIRECT. *Encyclopedia of Optimization*, 2001.

[7] M. J. D. Powell. Recent research at Cambridge on radial basis functions. Technical Report DAMTP 1998/NA05, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1998.

[8] Michael James Sasena. *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations*. PhD Thesis, University of Michigan, 2002.

A Test Problems

The problems used to test the implementation presented in this paper are taken from the `glc_prob` file in TOMLAB [5], which consists of a large set of test problems for global constrained optimization. The problems are stated here for convenience, and are presented in MATLAB form. Some of the problems might differ somewhat from the original ones.

1. Gomez 2

```
x_L = [-1 -1]';
x_U = [ 1  1]';
b_L = []; b_U = []; A = [];
c_L = [];
c_U = 0;
x_opt = [0 0];
f_opt = 0;

f = 0.1*x(1)^2+0.1*x(2)^2;
cx = -sin(4*pi*x(1))+2*(sin(2*pi*x(2)))^2;
```

2. Gomez 3

```
x_L = [-1 -1]';
x_U = [ 1  1]';
b_L = []; b_U = []; A = [];
c_L = [];
c_U = 0;
x_opt = [0.109 -0.623];
f_opt = -0.9711;

f = (4-2.1*x(1)^2+x(1)^4/3)*x(1)^2+x(1)*x(2)+(-4+4*x(2)^2)*x(2)^2;
cx = -sin(4*pi*x(1))+2*(sin(2*pi*x(2)))^2;
```

3. Hock-Schittkowski 59

```
u = [75.196     3.8112    0.0020567  1.0345E-5   6.8306  0.030234   1.28134E-3 ...
     2.266E-7  0.25645   0.0034604  1.3514E-5  28.106   5.2375E-6  6.3E-8     ...
     7E-10     3.405E-4  1.6638E-6  2.8673     3.5256E-5];
x_L = [0 0]';
x_U = [75 65]';
b_L = []; b_U = []; A = [];
c_L = [0 0 0];
c_U = [];
x_opt = [13.55010424 51.66018129];
f_opt = -7.804226324;

f = -u(1)+u(2)*x(1)+u(3)*x(1)^3-u(4)*x(1)^4+u(5)*x(2)-u(6)*x(1)*x(2) ...
   +u(7)*x(2)*x(1)^2+u(8)*x(1)^4*x(2)-u(9)*x(2)^2+u(10)*x(2)^3 ...
   -u(11)*x(2)^4+u(12)/(x(2)+1)+u(13)*x(1)^2*x(2)^2 ...
   +u(14)*x(1)^3*x(2)^2-u(15)*x(1)^3*x(2)^3+u(18)*exp(0.0005*x(1)*x(2)) ...
   -u(19)*x(1)^3*x(2)-u(16)*x(1)*x(2)^2+u(17)*x(1)*x(2)^3-0.12694*x(1)^2;
cx = [x(1)*x(2)-700;x(2)-x(1)^2/125;(x(2)-50)^2-5*(x(1)-55)];
```

4. Hock-Schittkowski 65

```
x_L = [-4.5 -4.5 -5]';
x_U = [ 4.5  4.5  5]';
b_L = []; b_U = []; A = [];
c_L = [0];
c_U = [];
x_opt = [3.650461821,3.650461821,4.6204170507];
f_opt = 0.9535288567;

f = (x(1)-x(2))^2+(x(1)+x(2)-10)^2/9+(x(3)-5)^2;
cx = [48-(x(1)^2+x(2)^2+x(3)^2)];
```

7. Schittkowski 234

```
x_L = [0.2 0.2]';
x_U = [ 2   2 ]';
b_L = []; b_U = []; A = [];
c_L = 0;
c_U = [];
x_opt = [0.2 0.2];
f_opt = -0.8;

f = (x(2)-x(1))^4-(1-x(1));
cx = [-x(1)^2-x(2)^2+1];
```

8. Schittkowski 236

```
B = [75.1963666677  -3.8112755343   0.1269366345  -2.0567665E-3   1.0345E-5 ...
     -6.8306567613   3.02344793E-2 -1.2813448E-3   3.52559E-5    -2.266E-7 ...
      0.2564581253  -3.460403E-3    1.35139E-5   -28.1064434908  -5.2375E-6 ...
     -6.3E-9         7E-10          3.405462E-4   -1.6638E-6     -2.8673112392]';
x_L = [0 0]';
x_U = [75 65]';
b_L = []; b_U = []; A = [];
c_L = [0;0];
c_U = [];
x_opt = [75 65];
f_opt = -58.9034;

f = B(1)+B(2)*x(1)+B(3)*x(1)^2+B(4)*x(1)^3+B(5)*x(1)^4+B(6)*x(2)+ ...
   B(7)*x(1)*x(2)+B(8)*x(1)^2*x(2)+B(9)*x(1)^3*x(2)+B(10)*x(1)^4*x(2)+ ...
   B(11)*x(2)^2+B(12)*x(2)^3+B(13)*x(2)^4+B(14)*(1/(x(2)+1))+ ...
   B(15)*x(1)^2*x(2)^2+B(16)*x(1)^3*x(2)^2+B(17)*x(1)^3*x(2)^3+ ...
   B(18)*x(1)*x(2)^2+B(19)*x(1)*x(2)^3+B(20)*(exp(5E-4*x(1)*x(2)));
f=-f;
cx = [x(1)*x(2)-700;x(2)-5*(x(1)/25)^2];
```

9. Schittkowski 237

```
B = [75.1963666677  -3.8112755343   0.1269366345  -2.0567665E-3   1.0345E-5 ...
     -6.8306567613   3.02344793E-2 -1.2813448E-3   3.52559E-5    -2.266E-7 ...
      0.2564581253  -3.460403E-3    1.35139E-5   -28.1064434908  -5.2375E-6 ...
     -6.3E-9         7E-10          3.405462E-4   -1.6638E-6     -2.8673112392]';
x_L = [54 0]';
```

```
x_U = [75 65]';
b_L = []; b_U = []; A = [];
c_L = [0;0;0];
c_U = [];
x_opt = [75 65];
f_opt = -58.9034;

f = B(1)+B(2)*x(1)+B(3)*x(1)^2+B(4)*x(1)^3+B(5)*x(1)^4+B(6)*x(2)+ ...
   B(7)*x(1)*x(2)+B(8)*x(1)^2*x(2)+B(9)*x(1)^3*x(2)+B(10)*x(1)^4*x(2)+ ...
   B(11)*x(2)^2+B(12)*x(2)^3+B(13)*x(2)^4+B(14)*(1/(x(2)+1))+ ...
   B(15)*x(1)^2*x(2)^2+B(16)*x(1)^3*x(2)^2+B(17)*x(1)^3*x(2)^3+ ...
   B(18)*x(1)*x(2)^2+B(19)*x(1)*x(2)^3+B(20)*(exp(5E-4*x(1)*x(2)));
f=-f;
cx = [x(1)*x(2)-700;x(2)-5*(x(1)/25)^2;(x(2)-50)^2-5*(x(1)-55)];
```

10. Schittkowski 239

```
B = [75.1963666677  -3.8112755343   0.1269366345  -2.0567665E-3   1.0345E-5 ...
     -6.8306567613   3.02344793E-2 -1.2813448E-3   3.52559E-5    -2.266E-7 ...
      0.2564581253  -3.460403E-3    1.35139E-5   -28.1064434908  -5.2375E-6 ...
     -6.3E-9         7E-10          3.405462E-4   -1.6638E-6     -2.8673112392]';
x_L = [0 0]';
x_U = [75 65]';
b_L = []; b_U = []; A = [];
c_L = [0];
c_U = [];
x_opt = [75 65];
f_opt = -58.9034;

f = B(1)+B(2)*x(1)+B(3)*x(1)^2+B(4)*x(1)^3+B(5)*x(1)^4+B(6)*x(2)+ ...
   B(7)*x(1)*x(2)+B(8)*x(1)^2*x(2)+B(9)*x(1)^3*x(2)+B(10)*x(1)^4*x(2)+ ...
   B(11)*x(2)^2+B(12)*x(2)^3+B(13)*x(2)^4+B(14)*(1/(x(2)+1))+ ...
   B(15)*x(1)^2*x(2)^2+B(16)*x(1)^3*x(2)^2+B(17)*x(1)^3*x(2)^3+ ...
   B(18)*x(1)*x(2)^2+B(19)*x(1)*x(2)^3+B(20)*(exp(5E-4*x(1)*x(2)));
f=-f;
cx = [x(1)*x(2)-700];
```

12. Schittkowski 332

```
tmp1 = [1:100]';
t    = pi*(1/3+(tmp1-1)/180);
x_L = [0 0]';
x_U = [1.5 1.5]';
b_L = []; b_U = []; A = [];
c_L = -30;
c_U = 30;
x_opt = [0.9114 0.02928];
%f_opt = 114.95; % This is f_opt given in "More testexampls .."
f_opt = 29.92437939227878;

f = pi/3.6*sum((log(t)+x(2)*sin(t)+x(1)*cos(t)).^2+(log(t)+x(2)*cos(t)-x(1)*sin(t)).^2);
pmax = max(180/pi*atan(abs((1./t-x(1))./(log(t)+x(2)))));
cx = pmax;
```

13. Schittkowski 343

```
x_L = [0 0 0]';
x_U = [36 5 125]';
b_L = []; b_U = []; A = [];
c_L = [0;0];
c_U = [];
x_opt = [16.51 2.477 124];
f_opt = -5.68478;

f = -0.0201*x(1)^4*x(2)*x(3)^2*1E-7;
cx = [675-x(1)^2*x(2);0.419-1E-7*x(1)^2*x(3)^2];
```

15. Floudas-Pardalos 3.3 TP 2

```
x_L = [78 33 27 27 27]';
x_U = [102 45 45 45 45]';
b_L = []; b_U = []; A = [];
c_L = [-85.334407;9.48751;10.699039];
c_U = [6.665593;29.48751;15.699039];
x_opt = [78 33 29.9953 45 36.7758];
f_opt = -30665.5387;

f = 37.293239*x(1)+0.8356891*x(1)*x(5)+5.3578547*x(3)^2-40792.141;
cx = [-0.0022053*x(3)*x(5)+0.0056858*x(2)*x(5)+0.0006262*x(1)*x(4)
       0.0071317*x(2)*x(5)+0.0021813*x(3)^2+0.0029955*x(1)*x(2)
       0.0047026*x(3)*x(5)+0.0019085*x(3)*x(4)+0.0012547*x(1)*x(3)];
```

16. Floudas-Pardalos 3.4 TP 3

```
x_L = [0 0 1 0 1 0]';
x_U = [6 6 5 6 5 10]';  % Upper bounds on x(1), x(2) added from lin.eq. 3
A = [ 1 -3 0 0 0 0
     -1  1 0 0 0 0
      1  1 0 0 0 0];
b_L = [-inf -inf  2 ]';
b_U = [  2    2   6 ]';
c_L = [4 4]';
c_U = [];
x_opt = [5 1 5 0 5 10];
f_opt = -310;

f = -25*(x(1)-2)^2-(x(2)-2)^2-(x(3)-1)^2-(x(4)-4)^2-(x(5)-1)^2-(x(6)-4)^2;
cx = [(x(3)-3)^2+x(4);(x(5)-3)^2+x(6)];
```

17. Floudas-Pardalos 3.5 TP 4

```
x_L = [0 0 0]';
x_U = [2 2 3]'; % Upper bounds on x(2) added by hkh, from lin.eq.2
A = [1 1 1
     0 3 1];
b_L = []';
b_U = [4 6]';
```

```
bvtmp=[3 1 2]';
rtmp=[1.5 -0.5 -5]';
c_L = [0.25*bvtmp'*bvtmp-rtmp'*rtmp]';
c_U = [];
x_opt = [0.5 0 3; 2 0 0];
f_opt = [-4;-4];

f = -2*x(1)+x(2)-x(3);
B=[0 0 1;0 -1 0;-2 1 -1];
r=[1.5 -0.5 -5]';
cx = [x'*B'*B*x-2*r'*B*x];
```

18. Floudas-Pardalos 4.10 TP 9

```
x_L = [0 0]';
x_U = [3 4]';
A = []; b_L=[]; b_U=[];
c_L = [-2 -36]';
c_U = [];
x_opt = [2.3295 3.1783];
f_opt = -5.5079;

f = -x(1)-x(2);
cx = [2*x(1)^4-8*x(1)^3+8*x(1)^2-x(2);4*x(1)^4-32*x(1)^3+88*x(1)^2-96*x(1)-x(2)];
```

19. Floudas-Pardalos 12.2 TP 1

```
% Kocis and Grossmann (1988)
x_L = [ 0 0 0  0   0  ]';
x_U = [ 1 1 1 1.6 1.5 ]'; % Bnd x_1 from 3rd ineq., Bnd x_2 from 4th ineq
A = [1 0 0 1 0; 0 1 0 0 1.333; -1 -1 1 0 0];
b_L=[-inf -inf -inf]';
b_U=[1.6 3 0];
c_L = [1.25  3 ]';
c_U = [1.25  3 ]';
x_opt = [0 1 1 1.12 1.31];
f_opt = 7.6672;
IntVars = [1:3];

y = x(1:3); % Integer variables
f = 2*x(4)+3*x(5)+1.5*y(1)+2*y(2)-0.5*y(3);
cx = [x(4)^2+y(1); x(5)^1.5 + 1.5* y(2)];
```

21. Floudas-Pardalos 12.2 TP 3

```
% Yuan et al. (1988)
x_L = zeros(7,1);
x_U = [1.2 1.8 2.5 1 1 1 1]'; % Added bnds x(1:3) from lin.eq 2-4
A = [1 1 1 1 1 1 0; eye(3,3),eye(3),zeros(3,1);1 0 0 0 0 0 1];
b_L=-inf*ones(5,1);
b_U=[  5 1.2 1.8 2.5 1.2]';
c_L=-inf*ones(4,1);
c_U = [  5.5 1.64 4.25 4.64]';
```

```
x_opt = [0.2 0.8 1.908 1 1 0 1];
f_opt = 4.5796;
IntVars = [4:7];

y = x(4:7); % Integer variables
f = (y(1)-1)^2 + (y(2)-2)^2 + (y(3)-1)^2 - log(y(4)+1) + ...
    (x(1)-1)^2 + (x(2)-2)^2 + (x(3)-3)^2;
cx = [x(1)^2+x(2)^2+x(3)^2 + y(3)^2; ...
      x(2)^2+y(2)^2; x(3)^2+y(3)^2; x(3)^2+y(2)^2];
```

23. Floudas-Pardalos 12.2 TP 5

```
% Prn et al. (1987)
x_L = ones(2,1);
x_U = 5*ones(2,1);
A   = [-1 -2 ; -3 1;4 -3];
b_L = -inf*ones(3,1);
b_U = [  -5 1 11]'; % Wrong sign in book for 1st constraint
c_L = -inf;
c_U =   -24; % Wrong sign in book
x_opt = [3 1];
f_opt = 31;
IntVars = [1:2];

f = 7*x(1) + 10*x(2);
cx = [x(1)^1.2*x(2)^1.7-7*x(1)-9*x(2)];
```

24. Floudas-Pardalos 12.2 TP 6

```
% Prn et al. (1987)
x_L = ones(2,1);
x_U = [10 6]';
A   = [1 -1 ; 3 2];
b_L = -inf*ones(2,1);
b_U = [3 24]';
c_L = -inf;
c_U =   39;
x_opt = [4 1];
f_opt = -17;
IntVars = [2];

f = -5*x(1) + 3*x(2);
y = x(2);
cx = [2*y^2 - 2*sqrt(y) - 2*sqrt(x(1))*y^2+11*y+8*x(1) ];
```

31. Floudas-Pardalos 12.2 TP 1 IntV last

```
% Kocis and Grossmann (1988)
x_L = [0    0  0 0 0 ]';
x_U = [1.6 1.5 1 1 1 ]'; % Bnd x_1 from 3rd ineq., Bnd x_2 from 4th ineq
A = [1 0 1 0 0;0 1.333 0 1 0; 0 0 -1 -1 1];
b_L=[-inf -inf -inf]';
b_U=[1.6 3 0];
```

```
c_L = [1.25  3 ]';
c_U = [1.25  3 ]';
x_opt = [1.12 1.31 0 1 1];
f_opt = 7.6672;
IntVars = [3 4 5];

y = x(3:5); % Integer variables
f = 2*x(1)+3*x(2)+1.5*y(1)+2*y(2)-0.5*y(3);
cx = [x(1)^2+y(1); x(2)^1.5 + 1.5* y(2)];
```

H

$Scale$	=	0
$Replace$	=	1
$Local$	=	0
$infStep$	=	0
$AddMP$	=	0
$fStarRule$	=	2
$DeltaRule$	=	1

Table 3: Static algorithm parameters

H

$idea$	=	1 or 2, see Section 3.
$rbfType$	=	1 - Thin Plate Splines interpolation, 2 - cubic interpolation.

Table 4: Used algorithm parameters

B Numerical Results

The `rbfSolve` code that this implementation is built upon has several algorithmic parameters. The aim of this work is not to do a comprehensive test on all these parameters and results are only presented for some of them. The restoring parameters are kept static for all tests. The static parameters, and their given value are shown in Table 3. The non static parameters are shown in Table 4. The results are presented for each problem for all possible combinations of these two parameters. These two parameters are further discussed in Section 3. All tests are performed within the TOMLAB Optimization Environment [5]. Information can also be found at `www.tomlab.biz`. As Global sub optimizer the `glcFast` code is used, which is a Matlab / Fortran implementation of the DIRECT algorithm by Jones et. al. [6]. As local sub optimizer the `minlpBB` code is used. The Dace Init Strategy was used in order to select the starting points.

Table 5 gives the results for all problems if solved directly with `glcFast` as a reference set. Table 6 to 23 gives the computational results for the algorithm presented in this paper. The computations where performed with a maximum of 300 evaluations of the objective function. This means that a run that results in 300 $FuncEv$ did not converge to the known global minimum. Instead the best value found is presented. The algorithm is said to converge if it finds a feasible point with a function value within the relative error of 0.1% from the known global minimum value. The algorithm can sometimes fail due to error in interpolation, or failure in sub optimizers. In these cases no minimum value, or number of function evaluations are reported. Instead $FuncEv$ is set equal to 0 and $fMin$ equal to -1. The multi surface idea, and the No Costly Constraints solver does not use the $cTolStrat$ parameter, therefore that parameter is always set to -1 for No Costly constraints and 2 for multi surface (since the cycle strategy in multi surface corresponds to cycle strategy number 2 in the single surface idea).

The failures of the single surface approach is, in all cases, due to failure in the global sub-optimizer when it failed to find a feasible point in respect to the interpolated constraint surface.

Column conType shows which RBF implementation that is used. 0 for No Costly Constraints, 1 for Single Constraint Interpolation surface and 2 for Multiple constraint interpolation surfaces. For the Single Constraint Interpolation implementation, where several different cycle strategies exists, only the results from strategy 2 is reported.

Table 5: Results for DIRECT

Problem	FuncEv	fMin	fGoal
1	3	0.000000	0.000000
2	221	-0.970800	-0.971100
3	155	-7.799364	-7.804226
4	821	0.954463	0.953529
6	377	1137.118200	1136.000000
7	213	-0.799568	-0.800000
8	89	-58.854305	-58.903400
9	107	-58.847368	-58.903400
10	89	-58.854305	-58.903400
12	329	29.949163	29.924379
13	245	-5.682311	-5.684780
15	1537	-30636.509000	-30665.539000
16	7727	-309.728320	-310.000000
17	445	-3.996113	-4.000000
18	633	-5.507316	-5.507900
19	248	7.673266	7.667200
21	2268	4.581849	4.579600
23	7	31.000000	31.000000
24	9	-7.000000	-17.000000
31	260	7.673266	7.667200

Table 6: Problem 1

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	0.000069	27
1	1	1	2	0.000216	33
1	1	2	2	0.000189	26
1	2	0	-1	0.000006	27
1	2	1	2	0.000000	23
1	2	2	2	0.000144	39
2	1	0	-1	0.000084	22
2	1	1	2	0.000564	22
2	1	2	2	0.000833	22
2	2	0	-1	0.000692	22
2	2	1	2	0.000573	22
2	2	2	2	0.000741	22

Table 7: Problem 2

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-0.970839	33
1	1	1	2	-0.970364	45
1	1	2	2	-0.971093	57
1	2	0	-1	-0.970604	27
1	2	1	2	-0.970828	63
1	2	2	2	-0.970625	45
2	1	0	-1	-0.971096	25
2	1	1	2	-0.970549	181
2	1	2	2	-0.957870	300
2	2	0	-1	-0.970413	22
2	2	1	2	-0.576546	300
2	2	2	2	-0.971098	141

Table 8: Problem 3

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-7.800375	33
1	1	1	2	-7.797624	57
1	1	2	2	-7.800362	51
1	2	0	-1	-7.796788	33
1	2	1	2	-7.802780	63
1	2	2	2	-7.802789	63
2	1	0	-1	-7.800896	24
2	1	1	2	-7.798139	165
2	1	2	2	-7.797871	37
2	2	0	-1	-7.798359	28
2	2	1	2	-7.802667	233
2	2	2	2	-7.802734	73

Table 9: Problem 4

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	0.953883	57
1	1	1	2	0.953852	69
1	1	2	2	0.953808	69
1	2	0	-1	0.953541	57
1	2	1	2	0.953576	63
1	2	2	2	0.953569	69
2	1	0	-1	0.954401	45
2	1	1	2	0.953530	69
2	1	2	2	0.953529	69
2	2	0	-1	0.953575	49
2	2	1	2	0.953637	49
2	2	2	2	0.953634	49

Table 10: Problem 7

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-0.799998	22
1	1	1	2	-0.799998	22
1	1	2	2	-0.799998	22
1	2	0	-1	-0.799998	22
1	2	1	2	-0.799998	22
1	2	2	2	-0.799998	22
2	1	0	-1	-0.799995	22
2	1	1	2	-0.799995	22
2	1	2	2	-0.799995	22
2	2	0	-1	-0.799995	22
2	2	1	2	-0.799995	22
2	2	2	2	-0.799995	22

Table 11: Problem 8

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-58.903235	23
1	1	1	2	-58.903327	24
1	1	2	2	-58.903327	24
1	2	0	-1	-58.903436	27
1	2	1	2	-58.903327	25
1	2	2	2	-58.903436	27
2	1	0	-1	-58.903235	24
2	1	1	2	-58.903110	24
2	1	2	2	-58.903235	24
2	2	0	-1	-58.903235	24
2	2	1	2	-58.903235	24
2	2	2	2	-58.903235	24

Table 12: Problem 9

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-58.903402	26
1	1	1	2	-58.903436	27
1	1	2	2	-58.903436	27
1	2	0	-1	-58.903436	27
1	2	1	2	-58.903436	27
1	2	2	2	-58.903436	27
2	1	0	-1	-58.903132	24
2	1	1	2	-58.903436	25
2	1	2	2	-58.903436	25
2	2	0	-1	-58.903436	25
2	2	1	2	-58.903436	25
2	2	2	2	-58.903436	25

Table 13: Problem 10

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-58.903235	23
1	1	1	2	-58.903327	24
1	1	2	2	-58.903327	24
1	2	0	-1	-58.903235	24
1	2	1	2	-58.903436	27
1	2	2	2	-58.903436	27
2	1	0	-1	-58.903235	24
2	1	1	2	-58.903235	24
2	1	2	2	-58.903235	24
2	2	0	-1	-58.903235	24
2	2	1	2	-58.903235	24
2	2	2	2	-58.903235	24

Table 14: Problem 12

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	54.001367	300
1	1	1	2	29.954025	123
1	1	2	2	29.952759	159
1	2	0	-1	54.001367	300
1	2	1	2	29.943665	75
1	2	2	2	29.945702	117
2	1	0	-1	29.925331	22
2	1	1	2	30.249293	300
2	1	2	2	30.505124	300
2	2	0	-1	29.925331	23
2	2	1	2	30.255343	300
2	2	2	2	30.328805	300

Table 15: Problem 13

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-5.684782	39
1	1	1	2	-5.204876	300
1	1	2	2	-5.682031	159
1	2	0	-1	-5.684782	45
1	2	1	2	-4.848352	300
1	2	2	2	-5.684552	93
2	1	0	-1	-5.679342	44
2	1	1	2	-1.621868	300
2	1	2	2	-5.684772	185
2	2	0	-1	-5.684782	49
2	2	1	2	-2.312871	300
2	2	2	2	-5.684370	89

Table 16: Problem 15

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-30665.539000	57
1	1	1	2	-30603.105000	300
1	1	2	2	-30641.630000	69
1	2	0	-1	-30665.539000	57
1	2	1	2	-30640.707000	99
1	2	2	2	-30656.031000	63
2	1	0	-1	-30662.078000	54
2	1	1	2	-30403.208000	300
2	1	2	2	-30656.162000	55
2	2	0	-1	-30664.317000	54
2	2	1	2	-30652.802000	123
2	2	2	2	-30665.539000	67

Table 17: Problem 16

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-310.000000	71
1	1	1	2	-310.000000	101
1	1	2	2	-309.994960	107
1	2	0	-1	-310.000000	77
1	2	1	2	-310.000000	89
1	2	2	2	-310.000000	101
2	1	0	-1	-310.000000	69
2	1	1	2	-310.000000	77
2	1	2	2	-310.000000	69
2	2	0	-1	-310.000000	77
2	2	1	2	-298.000000	300
2	2	2	2	-310.000000	69

Table 18: Problem 17

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-4.000000	39
1	1	1	2	-3.998585	57
1	1	2	2	-3.996497	51
1	2	0	-1	-4.000000	39
1	2	1	2	-3.997940	45
1	2	2	2	-4.000000	45
2	1	0	-1	-3.999568	35
2	1	1	2	-3.998929	53
2	1	2	2	-3.996347	45
2	2	0	-1	-3.999822	35
2	2	1	2	-3.999390	45
2	2	2	2	-3.999946	45

Table 19: Problem 18

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-5.508013	27
1	1	1	2	-5.408450	300
1	1	2	2	-5.508013	57
1	2	0	-1	-5.508013	27
1	2	1	2	-5.406384	300
1	2	2	2	-5.505462	33
2	1	0	-1	-5.508013	25
2	1	1	2	-5.243047	300
2	1	2	2	-5.178509	300
2	2	0	-1	-5.508013	25
2	2	1	2	-5.277321	300
2	2	2	2	-5.508013	45

Table 20: Problem 19

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	7.667180	57
1	1	1	2	-1.000000	0
1	1	2	2	7.667183	75
1	2	0	-1	7.667180	57
1	2	1	2	-1.000000	0
1	2	2	2	7.667180	207
2	1	0	-1	7.667180	55
2	1	1	2	-1.000000	0
2	1	2	2	7.667180	71
2	2	0	-1	7.667180	103
2	2	1	2	-1.000000	0
2	2	2	2	7.667178	71

Table 21: Problem 21

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	4.579582	71
1	1	1	2	4.879921	300
1	1	2	2	4.579669	89
1	2	0	-1	4.579582	71
1	2	1	2	4.772026	300
1	2	2	2	4.579714	89
2	1	0	-1	4.579582	73
2	1	1	2	4.624295	300
2	1	2	2	4.579912	73
2	2	0	-1	4.579582	69
2	2	1	2	4.726970	300
2	2	2	2	4.583187	73

Table 22: Problem 23

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	31.000000	22
1	1	1	2	31.000000	23
1	1	2	2	31.000000	23
1	2	0	-1	31.000000	22
1	2	1	2	31.000000	23
1	2	2	2	31.000000	23
2	1	0	-1	31.000000	22
2	1	1	2	31.000000	22
2	1	2	2	31.000000	22
2	2	0	-1	31.000000	22
2	2	1	2	31.000000	22
2	2	2	2	31.000000	22

Table 23: Problem 24

Idea	rbfType	conType	cTolStrat	fMin	funcEv
1	1	0	-1	-17.000000	27
1	1	1	2	-12.000000	300
1	1	2	2	-12.000000	300
1	2	0	-1	-17.000000	27
1	2	1	2	-12.000000	300
1	2	2	2	-12.000000	300
2	1	0	-1	-17.000000	25
2	1	1	2	-12.000000	300
2	1	2	2	-12.000000	300
2	2	0	-1	-17.000000	25
2	2	1	2	-12.000000	300
2	2	2	2	-12.000000	300