Thesis for the Degree of Master of Science

# Derivative-free Algorithms in Engineering Optimization

Anders Holmström

*Göteborg, October 2000*

## Abstract

Algorithms for derivative free optimization are overviewed, summarized and examined. Among the methods studied are Quasi Newton methods, the Nelder Mead Simplex algorithm, the Multidirectional search algorithm, Trust-region methods, the Response surface methodology and Model based optimization. A best and worst choice has been indicated for problems with certain specific characteristics.
Model based optimization is studied in more detail, as it seems to have wide applicability in solving problems of relevancy in industry. Furthermore, a new model based optimization algorithm have been developed and evaluated. The initial results are promising.

## Acknowledgements

First I want to thank my supervisor, Sven Ahlinder, at Volvo Technological Development for suggesting the subject area and contributing ideas along the way. I also want to thank Michael Patriksson, my supervisor at Chalmers University of Technology, for valuable insight in the subject of optimization and for constructive criticism concerning this report. For valuable ideas and inspiration I would furthermore like to thank Lennart Lunqvist at Volvo Technology Development and Göran Axbrink at Volvo Truck Corporation. For linguistic improvements and encouragement I would finally like to thank Arne Holmström.

# Contents

# 1 Introduction

This work has been carried out at Volvo Technology Development where I participate in development activities mainly concerning diesel combustion engines. Here, a lot of conflicting demands are put forward already in the choices governing the technical performance, which in turn have to be matched against economical considerations both from the manufacturer and the future vehicle owner. Last but not least, the environmental requirements put forward by the authorities have to be met. Therefore we almost always face optimization problems.

The first stage of this diploma work was devoted to summarizing published methods for optimization of objective functions where derivatives are not available. The insights gained from this work then served as a foundation for suggesting a new model based optimization algorithm, which was subsequently evaluated trough numerical studies of two examples.

The main purpose of the overview is to provide an engineer with the options that are available so as to enable an improved optimization. It may also contribute to the successful solution of problems not readily solvable by standard methods.

Engineering objective function values are in most cases obtained as a result of either computer simulations or measurements in a rig. Computer simulations usually don't return derivative information. Furthermore it may impose further obstacles in that the accuracy is often limited to a few digits and in unfavorable cases result in data subject to noise.

An engineering problem that was solved during this work is to adjust some unknown parameters in the simulation software GT-Power such that the result from the simulation corresponds to the measured data. GT-Power models the combustion and gas-exchange process in an internal combustion engine. A typical and very informative output from such a simulation is cylinder pressure, as a function of crank-angle, which was used to calculate the mean square error, relative to measurements, that is used as objective function. The unknown parameters influence the combustion rate calculated by GT-Power.
The Matlab algorithm FMINS is used to solve this minimization problem. FMINS uses a Nelder Mead Simplex optimization technique that is described on page 7. When adjusting 6 parameters in GT-Power the optimizer converges after 5 minutes and 100 simulations. 5 minutes is an acceptable amount of time for this task.

Measurements in a rig always behave stochastic, ally consequently alleviating the definition of derivatives. Often the objective function is so time consuming to evaluate that the remaining numerical computations carried out by the optimization algorithm constitute only a minor part of the total work. Therefore the time that is required for numerical computations in the optimization algorithm is not critical to the overall performance of the method.

Most engineering tasks can be translated into an optimization problem with simple constraints on the variables, such as the example below. Therefore, more complex constraints are not included in this study.

$$\underset{x \in R^n}{\text{minimize}}\ f(x)$$
$$\text{subject to}\ \ l \le x \le u$$

Model based optimization is according to my opinion a method with potential for wide applicability in solving engineering problems relevant to industry. This is my reason for studying this area in greater detail and also evaluating it by numerical examples.

## 2 Available methods for derivative free optimization

Derivative free optimization methods have a long history, see (Dixon 1972), (Himmelblau 1972) and (Polyak 1987) for extensive discussions and references. These methods come essentially in four different classes, a classification strongly influenced by (Conn and Toint 1996).

The first class contains algorithms which use finite-difference approximation of the objective function's derivatives in the context of a gradient based method, such as nonlinear conjugate gradients or quasi-Newton methods (see, for instance (Nash and Sofer 1996)).

The methods in the second class are often referred to as pattern search methods, because they are based on the exploration of the variable space using a well-specified geometric pattern, typically a simplex. The algorithm described in (Nelder and Mead 1965) is still the most popular minimization technique in use today in this context. More recent developments of pattern search methods include proposals by (Torczon 1991).

The methods of the third class are, for example, based on the progressive building and updating of a model of the objective function. Design of Experiment and interpolation models are proposed by for example (Owen 1992), (Booker 1994), (Mitchell, Sacks et al. 1989), (Morris, Currin et al. 1991) and (Sacks, Wynn et al. 1992). Response surface methodology is described in (Box, Hunter et al. 1977). Trust-region methods also belong to this class, see (Nash and Sofer 1996) and (Conn and Toint 1996).

The fourth class of optimization algorithms consists of methods that don’t utilize either a model or a direction. Simulated Annealing, Genetic Algorithms, and Grid Search are methods that belong to this class, which is not further described in this work.

## *2.1 The first class. Newton based methods*

### 2.1.1 Quasi Newton methods, as described in (Nash and Sofer 1996)

These methods need derivative values. Since we can't get the derivatives from the simulation software they need to be approximated. This is often done by finite differences using a step length $h$. The step length should depend on the machine accuracy $\varepsilon_{mach}$ to get the best possible approximation. A simple relation is,

$$h \approx \sqrt{\varepsilon_{mach}} .$$

In one dimension the finite difference approximations based on forward differences are

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x+h)}{h^2}.$$

The derivatives of multidimensional functions can be estimated by applying the finite difference formulas to each component of the gradient and the hessian matrix.
If we define the vector

$$e_i = (0 \quad ... \quad 0 \quad 1 \quad 0 \quad ... \quad 0)^T$$

having a one in the i-th component and zeros elsewhere, then the gradient can be approximated by

$$\left[\nabla f(x)\right]_i \approx \frac{f(x+he_i) - f(x)}{h} \qquad i = 1,...,n .$$

If the gradient is known then the hessian can be approximated via

$$\left[\nabla^2 f(x)\right]_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{\left[\nabla f(x+he_j) - \nabla f(x)\right]_i}{h} \qquad i = 1,...,n \qquad j = 1,...,n .$$

In many dimensions this approximation demands many objective function evaluations.

Newton's method is an algorithm for finding a solution to the equation $g(x) = 0$.
In one dimension the method can be described by

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} \qquad k = 0,1,... ,$$

see Figure 1,

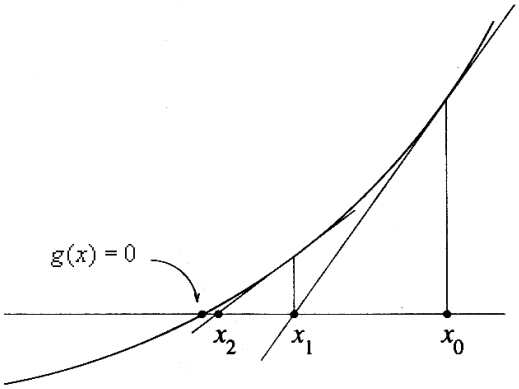


**Figure 1**

so in many dimensions we get

$$x_{k+1} = x_k - [\nabla g(x_k)]^{-1} g(x_k) .$$

In optimization Newton's method is used to find a local minimizer to the objective function. A local minimizer is a point $x_*$ that satisfies the condition

$$f(x_*) \le f(x) \text{ for all } x \text{ such that } \|x - x_*\| < \varepsilon$$

for all $\varepsilon > 0$ that are sufficiently small. If $f$ is differentiable and its first and second derivatives are continuos in a neighborhood of $x_*$ then the first and second order necessity of a local minimizer can be derived using Taylor series expansion around $x_*$. The first order necessary condition is

$$\nabla f(x_*) = 0 ,$$

and the second order necessary condition is further that

$$\nabla^2 f(x_*) \text{ is positive semidefinite.}$$

To use Newton's method for optimization, we apply it to the first-order necessary condition for a local minimizer:

$$g(x) = \nabla f(x) = 0 .$$

Since $\nabla g(x) = \nabla^2 f(x)$, this leads to the formula

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k) .$$

Newton's method is often more generally written as

$$x_{k+1} = x_k + p_k , \quad k = 0,1,\ldots ,$$

in its standard form $p_k$ is the solution to the system of equations

$$\nabla^2 f(x_k) p_k = -\nabla f(x_k).$$

There are many different methods based on approximating the hessian $\nabla^2 f(x_k)$ by another matrix $B_k$, that is available within fewer objective function evaluations. Then the search direction is obtained by solving

$$B_k p_k = -\nabla f(x_k).$$

The steepest descent method may be interpreted as a quasi-Newton method where the hessian $\nabla^2 f(x_k)$ is approximated by the identity matrix

$$B_k = I.$$

Quasi-Newton methods are generalizations of a method for one-dimensional problems called the secant method. The secant method uses the approximation

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}.$$

In the multidimensional case, we write it

$$\nabla^2 f(x_k)(x_k - x_{k-1}) \approx \nabla f(x_k) - \nabla f(x_{k-1}).$$

From this we obtain a condition used to define the quasi-Newton approximations $B_k$:

$$B_k(x_k - x_{k-1}) = \nabla f(x_k) - \nabla f(x_{k-1})$$

We call this the secant condition. This is not sufficient to determine $B_k$; for further details see (Nash and Sofer 1996).

In order to prove that the algorithm converges it is a necessity that the search direction is a descent direction, which is enforced by requiring that $p_k^T \nabla f(x_k) < 0$.

After introduction of a step length $\alpha_k$

$$x_{k+1} = x_k + \alpha_k p_k$$

where $\alpha_k$ is a scalar chosen so that $f(x_{k+1}) < f(x_k)$ it will be possible to guarantee that the method will converge to a stationary point and possibly a local minimizer. The technique is called a “line search” because a search for a new point $x_{k+1}$ is carried out along the half line $x_k + \alpha p_k$, $\alpha \geq 0$. Intuitively we would like to choose $\alpha_k$ as the solution to

$$\min_{\alpha \geq 0} f(x_k + \alpha p_k).$$

That is $\alpha_k$ would be a result of a one-dimensional minimization problem.

Usually we don't search the line exactly but rather estimate the optimal $\alpha$, perhaps through interpolations; a simple line search method is described below. The sufficient-decrease condition on $\alpha_k$ ensures that some nontrivial reduction in the function value is obtained at each iteration. The reduction is predicted in terms of the Taylor series. A linear approximation to $f(x_k + \alpha p_k)$ is obtained from

$$f(x_k + \alpha p_k) \approx f(x_k) + \alpha p_k^T \nabla f(x_k).$$

In the line-search we will demand that the step length $\alpha_k$ produce a decrease in the function value that is at least some fraction of the decrease predicted by the above linear approximation. More specifically we require that

$$f(x_k + \alpha_k p_k) \leq f(x_k) + \mu \alpha_k p_k^T \nabla f(x_k) \qquad (*)$$

where $\mu$ is some scalar satisfying $0 < \mu < 1$, see Figure 2. When $\mu$ is smaller this condition is easier to satisfy since only a small decrease in the function value is required. A simple means to satisfy (*) is to use backtracking: define $\alpha_k$ to be the first element of the sequence

$$1, \frac{1}{2}, \frac{1}{4}, ..., 2^{-i}, ...$$

to satisfy the sufficient decrease condition. If the descent direction condition, $p_k^T \nabla f(x_k) < 0$, is satisfied then it is possible to prove that there exists a finite *i* which satisfies (*).

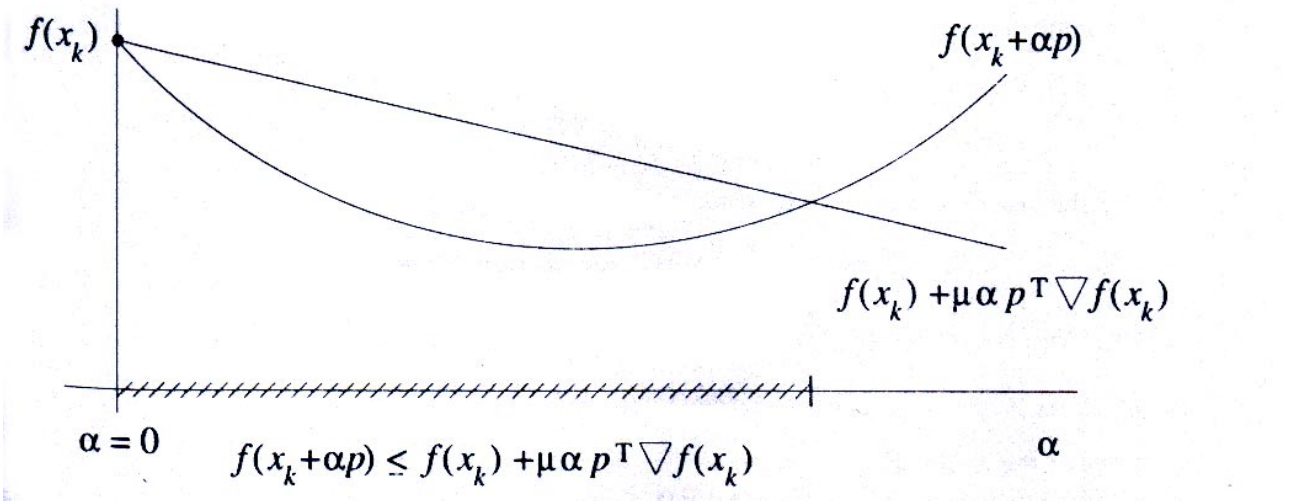


**Figure 2**

## *2.2 The second class. Pattern search methods*

### 2.2.1 The Nelder Mead simplex algorithm

Here is the description of the simplex algorithm as described in (Nelder and Mead 1965).
We consider, initially, the minimization of a function of $n$ variables, without constraints. The method is based on different geometrical operations on a simplex. The simplex consists of ($n$+1) points and can in two dimensions be interpreted as the corners of a triangle. Three geometrical operations are used — reflection, expansion and contraction. All these operations will be defined by replacing the worst point in the simplex, $P_h$, with one new point, but if the contraction failed then all points but the best point, $P_l$, are replaced. The operations are defined as follows.

Initialization

$P_0, P_1, ..., P_n$ are the $(n+1)$ points in n-dimensional space defining the current simplex. We write $f_i$ for the objective function value at $P_i$, and define

$$h \text{ as the index such that } f_h = \max_i(f_i)$$

and

$$l \text{ as the index that } f_l = \min_i(f_i).$$

Further we define $\overline{P}$ as the centroid of all points $P_i$ satisfying $i \neq h$.

Reflection

The reflection of $P_h$ is denoted by $P^*$ and it is defined by the relation

$$P^* = (1+\alpha)\overline{P} - \alpha P_h,$$

where $\alpha$ is a positive constant, the reflection coefficient. The objective value in $P^*$ is denoted $f^*$. If $f^*$ lies between $f_h$ and $f_l$, then $P_h$ is replaced by $P^*$ and we start again with the initialization of the new simplex.

Expansion

If $f^* < f_l$, that is, if the reflection has produced a new minimum, then we expand $P^*$ to $P^{**}$ by the relation

$$P^{**} = \gamma P_h + (1-\gamma)\overline{P},$$

where the expansion coefficient $\gamma > 1$. If $f^{**} < f_l$ we replace $P_h$ by $P^{**}$ and restart the process, but if $f^{**} > f_l$ then we have a failed expansion, and we replace $P_h$ by $P^*$ before restarting.

Contraction

If on reflecting $P$ to $P^*$ we find that $f^* > f_i$ for all $i \neq h$, i.e. that replacing $P$ by $P^*$ leaves $f^*$ at the maximum, then we define a new $P_h$ to be either the old $P_h$ or $P^*$, whichever has the lower $f$ value, and form

$$P^{**} = \beta P_h + (1-\beta)\overline{P}.$$

The contraction coefficient $\beta$ lies between 0 and 1. We then accept $P^{**}$ for $P_h$ and restart unless $f^{**} > \min(f_h, f^*)$, i.e. the contracted point is worse than the better of $P_h$ and $P^*$.

## Failed contraction

For such a failed contraction we replace all the $P_i$ by $(P_i + P_l)/2$ and restart the process.

## Typical parameter values

The coefficients $\alpha, \beta, \gamma$ give the factor by which the volume of the simplex is changed by the operations of reflection, contraction or expansion respectively. Typical coefficient values are

$$\alpha = 1, \beta = \frac{1}{2}, \gamma = 2 \ .$$

## The stopping criterion

The stopping criterion is based on comparing the standard error of the $f$ 's with the preset value $\varepsilon$; terminate if

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n+1}(f_i - \bar{f})^2} < \varepsilon \ .$$

## A numerical example

The Matlab function FMINS is an implementation of the algorithm described above. This Matlab function is used to illustrate the method on the optimization problem.

$$\underset{x \in R^2}{\text{minimize}} \ f(x) = x^T x$$

with the initial solution

$$x_0 = [1 \quad 1]^T \ .$$

The iterations is visualized in Figure 3 to Figure 5, Figure 3 shows the starting simplex and the following iterations. The starting simplex in the algorithm consists of the initial point $x_0$ and two additional points: $[1 \quad 1.05]^T$ and $[1.05 \quad 1]^T$. In the figures every objective function evaluation is numbered by the order in which they were evaluated. Additionally a description of how the algorithm proceeds is given next to each figure.

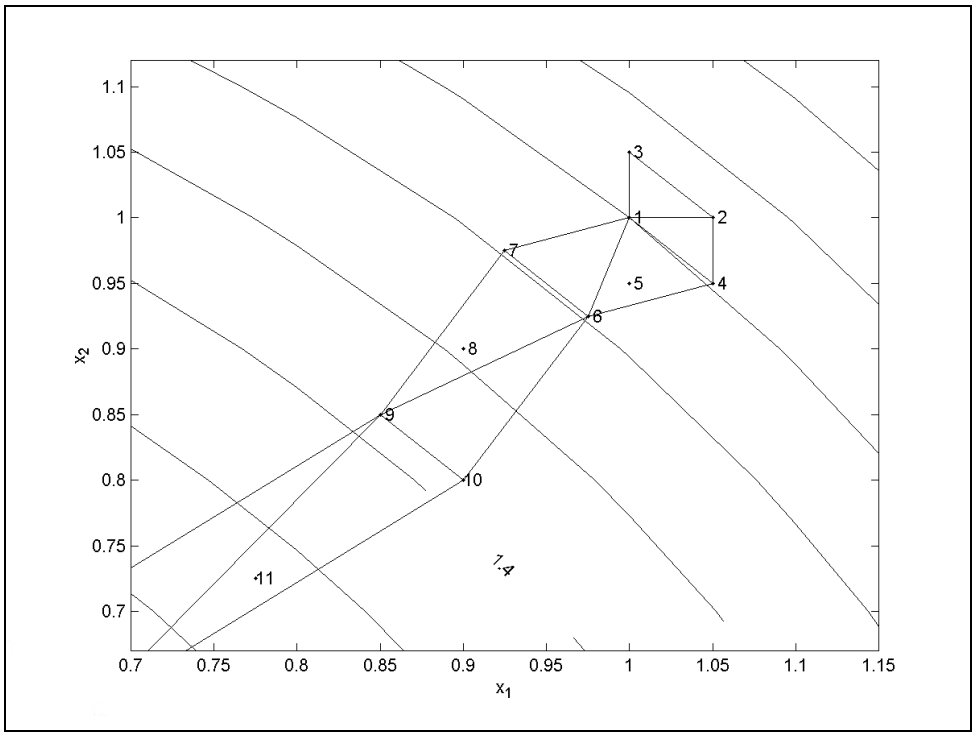


**Figure 3**

The description of how the algorithm proceeds is constructed as follows. <i,j,k> marks the points in the simplex at every new iteration, underline marks the worst point $P_h$ in the simplex. Reflection, expansion or contraction is indicated with r, e and c.

In Figure 3 the steps are

<1,2,3>r  <1,2,4>e  <1,6,4>r  <1,6,7>e  <9,6,7>r  <9,6,10>e

In Figure 4 it can be seen how the simplex is deformed on its way towards the optima.

The algorithm proceeds with the subsequent steps
< 9, 6,10>e < 9,12,10>r < 9,12,13>e <15,12,13>r <15,12,16>r <15,17,16>r <15,17,19>c.

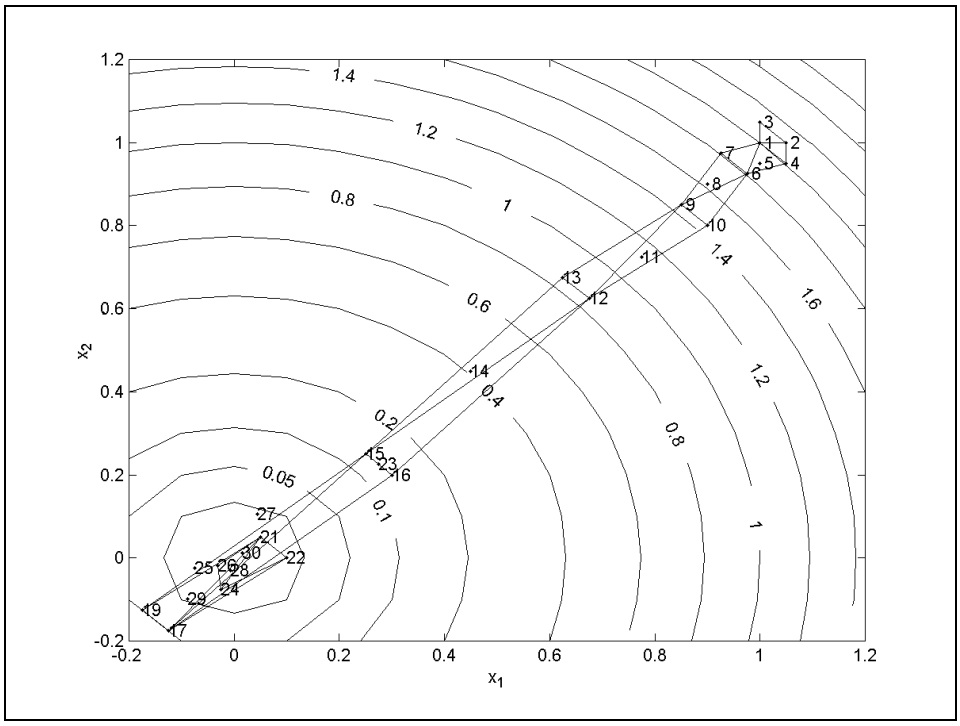


**Figure 4**

Figure 5 illustrates how the simplex is contracted close to the optimal solution. The steps are <15,17,19>c <21,17,19>r <21,17,22>c <21,24,22>c <21,24,26>c <21,28,26>c <30,28,26>.

The algorithm will continue in a similar way until the stopping criterion is met.

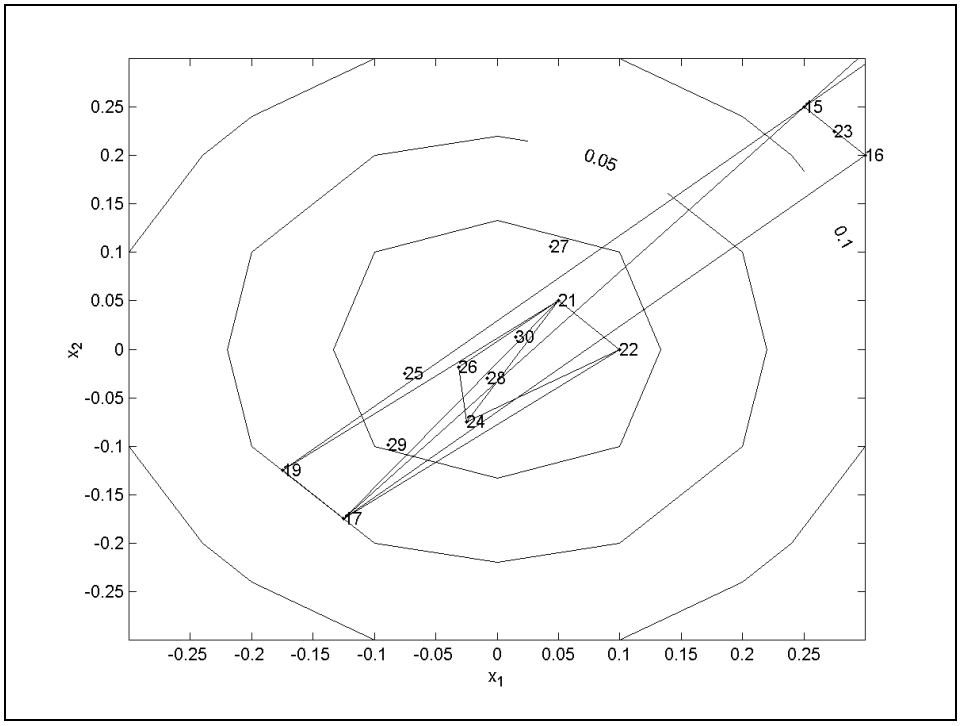


**Figure 5**

Other numerical experiments have shown that the Nelder Mead Simplex algorithm does not always converge to a local optimum. This shortcoming can be handled by restarting the algorithm with the previous result as the initial solution until no further improvement is made.

### 2.2.2 The multidirectional search algorithm

The multidirectional search algorithm was introduced by Dennis and Torczon in 1989 (Torczon 1989) as a step towards a general-purpose optimization algorithm with promising properties for parallel computation.

According to (Dennis and Torczon 1991) the Nelder-Mead Simplex algorithm can converge to non-minimizers when the dimension of the problem becomes large enough. This behavior occurred even on such a simple problem as

$$\min_{x \in R^n} f(x) = x^T x$$

for $n \geq 16$. One of the advantages of the multidirectional search is that unlike the Nelder-Mead simplex algorithm, it is backed by convergence theorems that numerical testing also indicate are borne out in practice (Dennis and Torczon 1991).

If the method is appropriately designed, and if the function $f$ is continuously differentiable, then it is possible to prove that

$$\liminf_{k\to\infty} \|\nabla f(x_k)\| = 0,$$

where $x_k$ is the trial point at the k-th iteration.

An iteration of the basic multidirectional search algorithm begins with a simplex $S$ in $R^n$, with vertices $v_0, v_1, ..., v_n$. The best vertex $v_0$ is designated to be a vertex for which $f(v_0) \le f(v_j)$ for $j = 1, ..., n$. We now describe a complete iteration to arrive at a new simplex $S_+$. The first move of the iteration is to reflect $v_1, ..., v_n$ through the best vertex $v_0$. Figure 6 illustrates an example for $n = 2$.

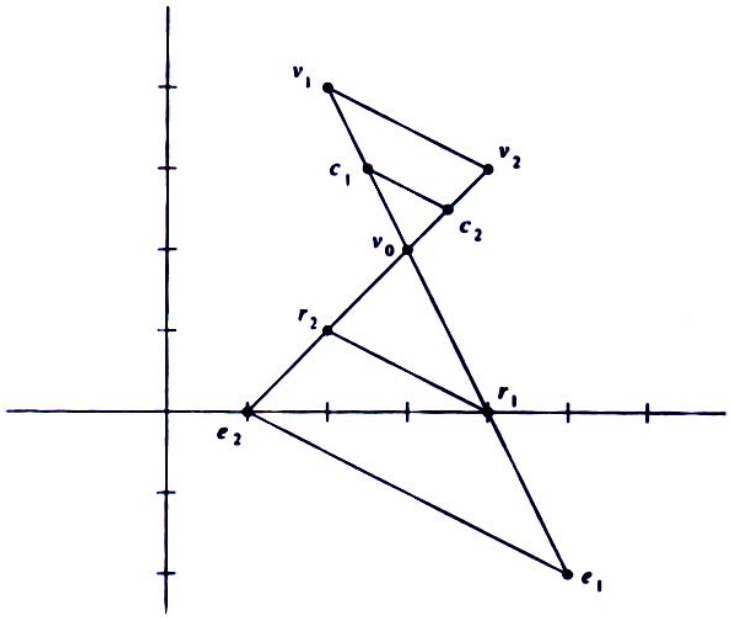


**Figure 6**

Continuing with *n*=2, the reflected vertices are labeled $r_1$ and $r_2$. If a reflected vertex gives a better function value than the best vertex, then the reflection step is deemed successful and the algorithm tries an expansion step. The expansion step consists of expanding each reflected edge $(r_j - v_0)$ to twice its length to give a new expansion vertex $e_j$. Above the expansion vertices are labeled $e_1$ and $e_2$. In an iteration of this basic algorithm, the expansion step would be tried only if the reflection step was successful, and it would be taken only if some expansion vertex was better than all the reflection vertices. Thus, if we try the expansion step, then the new simplex $S_+$ is either the expansion simplex $< v_0, e_1, e_2 >$ or the reflection simplex $< v_0, r_1, r_2 >$.

The other branch of the basic algorithm is the case where the reflection step was unsuccessful, i.e., no reflection vertex has a better function value than $f(v_0)$. In this case, we take $S^+$ to be the contraction simplex formed by replacing each vertex of the worst $n$-face in the original simplex by the point midway from it to be the best vertex. Thus, the contraction step takes $S_+$ to be $< v_0, c_1, c_2 >$.

To complete the iteration of the basic algorithm, we take $v_0^+$ to be the best vertex of $S_+$.

## *2.3 The third class. Methods that approximate the objective over a region*

### 2.3.1 Trust-region algorithms

Some of the algorithms in this class use the trust region approach to guarantee convergence. The convergence proof relies on that the model can be trusted inside the trust-region. As described in (Nash and Sofer 1996). Trust-region methods make explicit reference to a model of the objective function. For Newton's method this model is a quadratic model derived from the Taylor series for $f$ about the point $x_k$.

$$\varphi_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p .$$

In the method this model will only trusted within a limited neighborhood, the trust-region, of the point $x_k$, defined by the constraint

$$\|p\| \le \Delta_k , \quad \Delta_k > 0 .$$

Where $\|\cdot\|$ denotes Euclidean norm. This will serve as the limit to the size of the step taken from $x_k$ to $x_{k+1}$, see Figure 7. The value of $\Delta_k$ will be adjusted based on the agreement between the model $\varphi_k$, at $p$, and the objective function $f$, at $x_k + p$. If the agreement is good, then the model can be trusted and $\Delta_k$ will be increased.

Here follows a more detailed description of the algorithm. The notations below are the same as in chapter 2.1.1.

The Trust-Region algorithm described in, (Nash and Sofer 1996), in more detail

Specify some initial guess of the solution $x_0$. Select the initial trust-region bound $\Delta_0 > 0$. Specify the constants $0 < \mu < \eta < 1$ i.e. $\mu = 1/4$ and $\eta = 3/4$.

For k=0,1…

If $x_k$ is optimal, stop.

Solve

$$\min_p \varphi(p)$$

$$\text{subject to } \|p\| \le \Delta_k$$

for the trail step $p_k$.

Compute

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{f(x_k) - \varphi_k(p_k)} = \frac{\text{actual reduction}}{\text{predicted reduction}}$$

If $\rho_k \le \mu$ then $x_{k+1} = x_k$ (unsuccessful step), else $x_{k+1} = x_k + p_k$ (successful step).

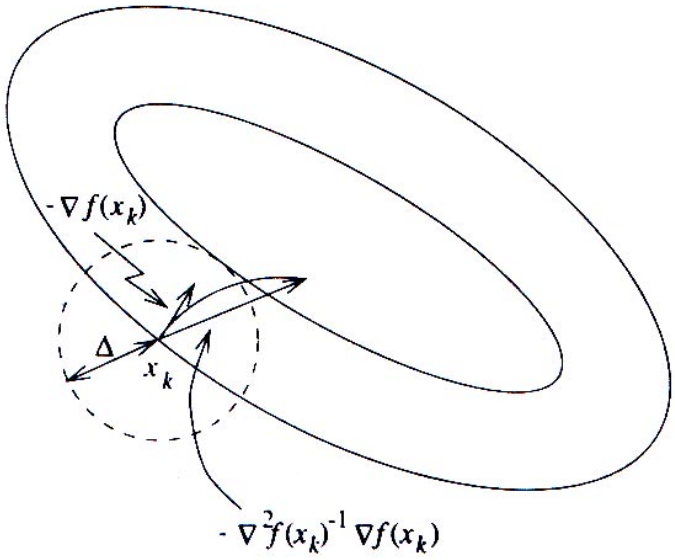


**Figure 7**

Update $\Delta_k$:

$$\rho_k \le \mu \Rightarrow \Delta_{k+1} = \frac{1}{2}\Delta_k$$
$$\mu < \rho_k < \eta \Rightarrow \Delta_{k+1} = \Delta_k$$
$$\rho_k \ge \eta \Rightarrow \Delta_{k+1} = 2\Delta_k$$

The value of $\rho_k$ indicates how well the model predicts the reduction in the function value. If $\rho_k$ is small ($\rho_k < \mu$), then the actual reduction in the function value is much smaller than that predicted by $\varphi_k(p_k)$, indicating that the model cannot be trusted for a bound as large as $\Delta_k$. In this case the step $p_k$ will be rejected and $\Delta_k$ will be reduced. If $\rho_k$ is large ($\rho_k \ge \eta$), then the model is adequately predicting the reduction in the function value, suggesting that the model can be trusted over an wider region. In this case the bound $\Delta_k$ will be increased.

## More recent Trust-region methods

More recent work, e.g. (Conn and Toint 1996), develop trust region methods that don't use finite differences to make a model but use interpolating models instead. This makes the algorithm more robust towards noise. Some potential new problems are instead introduced. Enough points p

$$p = \frac{1}{2}(n+1)(n+2)$$

to determine a quadratic function is needed and they must support a quadratic model since 6 points on a line in two dimensions isn't enough to construct a quadratic model. A method for determining the support of a quadratic function is presented in (Conn and Toint 1996).

A promising method called The Filter Method is described in (Conn, Gould et al. 2000). It is advantageous because of the way the algorithm handles constraints. The Filter Method

approximates the constraints in the optimization problem separated from the objective function instead of adding a penalty to the objective function. It is called The Filter Method because the method rejects points that don't improve either the objective function value or the constraints unfeasibility value.

For most, Trust-region algorithms, it can be established that

$$\lim \|\nabla f(x_k)\| = 0 .$$

### 2.3.2 Response surface methodology as described in (Box, Hunter et al. 1977)

This method focuses on optimizing measured systems and is based on experimental design. When applying it to simulation software it is not desirable to use replicated points if the software is deterministic and always gives the same answer for the same input. After determining which variables are important, the experimenter first employs the method of steepest ascent to get close to the maximum and then approximates the surface in the vicinity of the maximum with a fitted second-degree equation. The first-order and second-order models of the objective function are called the response surface. See Figure 8 for an overview of the method. Some of the techniques in the overview are not further described in this work (i.e. blocking variables and randomized experiments), for details see (Box, Hunter et al. 1977).

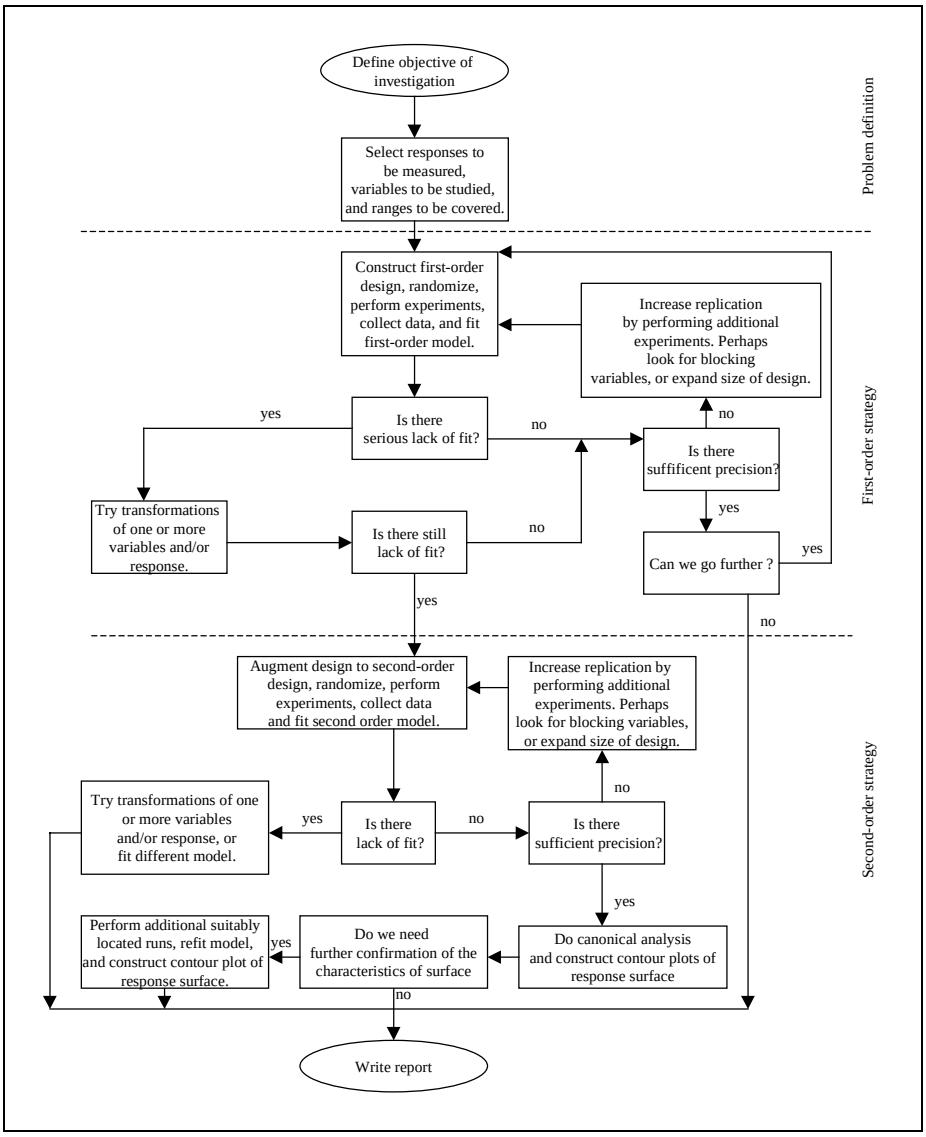


**Figure 8**

<u>Illustration of response surface methodology, a chemical example (Box, Hunter et al. 1977).</u>

The objective of the following laboratory investigation was to find settings of time(t) and temperature (T) that produced maximum yield subject to constraints discussed later. The best conditions known at the beginning of the work were t=75minutes , T=130C, and past experience had suggested that the experimental error standard deviation was about $\sigma = 1.5$.

## A first order Design

Taking as a starting point the best conditions known by the investigator time varied from 70 to 80 minutes and temperature from 127.5 to 132.5 C according to the design shown in the Table below.

| Run | time | temperature | $x_1$ | $x_2$ | $f$ |
|---|---|---|---|---|---|
| 1 | 70 | 127.5 | -1 | -1 | 54.3 |
| 2 | 80 | 127.5 | +1 | -1 | 60.3 |
| 3 | 70 | 132.5 | -1 | +1 | 64.6 |
| 4 | 80 | 132.5 | +1 | +1 | 68.0 |
| 5 | 75 | 130.0 | 0 | 0 | 60.3 |
| 6 | 75 | 130.0 | 0 | 0 | 64.3 |
| 7 | 75 | 130.0 | 0 | 0 | 62.3 |

The levels of the variables in coded units are

$$x_1 = \frac{t-75}{5}, \quad x_2 = \frac{T-130}{2.5}.$$

The design employed was a $2^2$ factorial design with three center points. It is called a first order design because it allows efficient fitting and checking of the first-degree polynomial model,

$$f = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

It was chosen because, at this stage of the investigation, the experimenter believed, but was not certain, that he was some distance away from the maximum (some way down the hillside that represents the true response surface). In these circumstances it would be likely that the predominant local characteristics of the surface were its gradients and that the local surface could be roughly represented by the planar model having slope $\beta_1$ in the $x_1$ direction and slope $\beta_2$ in the $x_2$ direction. If this idea was correct, by estimating $\beta_1$ and $\beta_2$ it would be possible to follow a direction of increasing yield up the hillside $f$. The design chosen:

1) Allows the planar model to be efficiently fitted
2) Allows checks to be made to determine whether the planar model is representationaly adequate
3) Provides some estimate of the experimental error

## Least squares fit

The least squares estimate of $\beta_1$ is

$$b_1 = \tfrac{1}{4}(-54.3 + 60.3 - 64.6 + 68.0) = 2.35$$

The coefficient $\beta_1$ is the change in the response when $x_1$ is changed by one unit. The linear main effect in a factorial design is the change in response when $x_1$ is changed from –1 to +1, that is, by two units. Thus $b_1$ is half the linear main effect obtained from the difference of average at the shorter and longer reaction times. Similarly $b_2$ is 4.50. The least squares estimate of $\beta_0$ is the average of all seven observations, 62.01. We thus obtain the fitted equation

$$\hat{f} = 62.01 + 2.35x_1 + 4.50x_2$$
$$(\pm 0.57)\ \ (\pm 0.75)\ \ (\pm 0.75)$$

where the standard errors, shown beneath the coefficients of the equation, are calculated based on the assumption that $\sigma = 1.5$. Although least squares calculations tentatively assumes adequacy of the first-degree (planar) model, the design was chosen to allow checks of this assumption to be made.

## Interaction check

The planar model supposes that the effects of the variables are additive. Interaction between the variables would be measured by the coefficient $\beta_{12}$ of an added cross-term $x_1x_2$ in the model. This coefficient is estimated by

$$b_{12} = \tfrac{1}{4}(54.3 - 60.3 - 64.6 + 68.0) = -0.65$$

The standard error of this estimate is 0.75, the same as it for $\beta_1$ and $\beta_2$ (we use $\sigma = 1.5$ as before).

## Curvature check

Another check of local planarity is supplied by comparing $\bar{f}_f$, the average of the four points of the $2^2$ factorial, with $\bar{f}_c$, the average at the center of the design. By thinking of the design as sitting on a saucer-like surface, it is seen that $\bar{f}_f - \bar{f}_c$ is a measure of overall curvature of the surface . It can also be shown that, if $\beta_{11}$ and $\beta_{22}$ are coefficients of the terms $x_1^2$ and $x_2^2$ , this curvature measure will be an estimate of $\beta_{11} + \beta_{22}$. Thus the estimate of the "overall curvature" is

$$b_{11} + b_{22} = \tfrac{1}{4}(54.3 + 60.3 + 64.6 + 68.0) - \tfrac{1}{3}(60.3 + 62.3 + 64.3) = -0.50$$

Using $\sigma = 1.5$, we obtain 1.15 for the standard error of this estimate.
In summary then, the planarity checking functions are

$$b_{12} = -0.65 \pm 0.75, \quad b_{11} + b_{22} = -0.50 \pm 1.15$$

and there is no reason to question the adequacy of the planar model since the estimated values are smaller than the standard error for the estimates.

## Estimation of error

An estimate of the experimental error variance, which is very approximate since it has only two degrees of freedom, is obtained from the replicate observations at the center.

$$s_1^2 = \frac{60.3^2 + 62.3^2 + 64.3^2 - (186.9)^2/3}{2} = 4.0$$

Thus

$$s_1 = 2.0$$

This agrees fairly well with the preliminary value of $\sigma = 1.5$ used above.

## Path of steepest ascent

The path of steepest ascent, which is perpendicular to the contour lines, can be calculated as follows. Starting at the center of the experimental region, the path is followed by simultaneously

moving $b_2 = +4.5$ units in $x_2$ for every $b_1 = +2.35$ units moved in $x_1$. A convenient set of points on the path of steepest ascent is shown in table below.

| Run | $x_1$ | $x_2$ | time | temperature | $f$ |
|---|---|---|---|---|---|
| 5,6,7 | 0 | 0 | 75 | 130.0 | 62.3 (Average) |
| 8 | 1 | 1.91 | 80 | 134.8 | 73.3 |
| 9 | 5 | 9.57 | 100 | 153.9 | 58.2 |
| 10 | 3 | 5.74 | 90 | 144.4 | 86.8 |

Runs made at points 8,9, and 10 gave yields as indicated. Run 8 (f=73) was encouraging and led to a large jump being taken to run 9. This yield(f=58.2) was low, however, indicating that too large a move had been made. Run 10 (f=86.8) at intermediate conditions proved a very substantial improvement over any results obtained so far. This result suggests that subsequent experiments should be made in the neighborhood of run 10.
As the region of interest moves up the surface, it can be expected that first-order effects will become smaller. Since the blurring effects of experimental errors have to be faced, the opportunity may be taken at this time to increase the second design by a factor of, say, two to increase the absolute magnitude of the effects in relation to the error.
(Furthermore, as a region of interest moves up the surface, the possibility increases that a second-degree approximation will be needed. Expanding the design is also sensible in this eventuality because a second-degree approximation should provide an adequate representation over a larger region than a first-degree approximation.)

## A second design

A new $2^2$ factorial design with two center points situated close to run 10 had coded variables

$$x_1 = \frac{time - 90}{10}, \qquad x_2 = \frac{temperature - 145}{5}$$

The data obtained from the six runs performed in random order are shown in the table below. Analyzing this second first-order design, we obtain the following results.

| Run | time | temperature | $x_1$ | $x_2$ | $f$ | |
|---|---|---|---|---|---|---|
| 11 | 80 | 140 | -1 | -1 | 78.8 | |
| 12 | 100 | 140 | +1 | -1 | 84.5 | |
| 13 | 80 | 150 | -1 | +1 | 91.2 | second |
| 14 | 100 | 150 | +1 | +1 | 77.4 | first-order |
| 15 | 90 | 145 | 0 | 0 | 89.7 | design |
| 16 | 90 | 145 | 0 | 0 | 86.8 | |
| 17 | 76 | 145 | $-\sqrt{2}$ | 0 | 83.3 | runs |
| 18 | 104 | 145 | $+\sqrt{2}$ | 0 | 81.2 | added |
| 19 | 90 | 138 | 0 | $-\sqrt{2}$ | 81.2 | from a |
| 20 | 90 | 152 | 0 | $+\sqrt{2}$ | 79.5 | composite |
| 21 | 90 | 145 | 0 | 0 | 87.0 | design |
| 22 | 90 | 145 | 0 | 0 | 86.0 | |

## Least squares fit

On the assumption that a first-degree polynomial model is again applicable, least squares estimation yields the fitted equation

$$f = 84.73 - 2.025x_1 + 1.325x_2$$
$$(\pm 0.61) \quad (\pm 0.75) \quad (\pm 0.75)$$

## Interaction and Curvature checks

The checking functions yield the values

$$b'_{12} = -4.88 \pm 0.75,$$

from which it is evident that in the present region the first-degree equation is quite inadequate to represent the local response function.

## Augmenting the design to Fit a second order model

Since the first–degree polynomial approximation had been shown to be quite inadequate the new experimental region, a second degree polynomial approximation

$$f = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2$$

was now contemplated. To estimate efficiently all six coefficients in the model and to provide for appropriate checking and error determination, the second factorial group of points(runs 11 through 16) was augmented with a “star “ design consisting of four axial points and two center points(runs 17 trough 22).

## Least Sqaures Fit

The second-degree equation fitted by least squares to the 12 results from the composite design (runs 11 through 22) is

$$\hat{f} = 87.36 - 1.39x_1 + 0.37x_2 - 2.15x_1^2 - 3.12x_2^2 - 4.88x_1 x_2$$

## Checks

We have seen that the first-order design was chosen so that estimates of selected second-order terms or combinations of them could be isolated and thereby supply checks on the adequacy of the first-degree equation. The composite design has been chosen in the same way so that selected combinations of third-order terms can be isolated and check the adequacy of the second-degree equation. For example, consider the distribution of points along the $x_1$ axis. If the surface is exactly quadratic in this direction, it can be shown that the estimate of the slope obtained from the axial points $f_{17}$ and $f_{18}$ will be the same as that obtained from the factorial points $f_{11}, f_{12}, f_{13}, f_{14}$. A measure of the discrepancy in the two measures of slope is

$$\frac{1}{\sqrt{8}}(f_{18} - f_{17}) - \tfrac{1}{4}((f_{12} + f_{14} - f_{11} - f_{13}).$$

We will denote this measure by $\beta_{111} - \beta_{122}$ (where $\beta_{111}$ and $\beta_{122}$ are coefficients of $x_1^3$ and $x_1 x_2^2$ respectively, in a third-degree polynomial). Both of the latter coefficients would be zero if the surface were described by a second-degree equation. If we consider the distribution of points along the $x_2$ axis, a similar estimate $b_{222} - b_{112}$ of $\beta_{222} - \beta_{112}$ is obtained. For this estimates we find.

$$b_{111} - b_{122} = 1.28 \pm 1.06$$
$$b_{222} - b_{112} = -1.93 \pm 1.06$$

There is an indication of some slight model inadequacy in the $x_2$-direction, but we shall ignore it in this elementary account[1].

## Nature of the fitted surface

Before attempting to interpret the fitted surface, we need to consider whether or not it is estimated with sufficient precision. This can be done by a special application of the analysis of variance (Box and Wetz 1973). However, in this present short survey we approach the problem from a simpler but equivalent point of view.

It can be shown that, no matter what the design, the average variance of the fitted values $\hat{f}$ is

$$\bar{V}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} V(\hat{f}_i) = \frac{p\sigma^2}{n}$$

where *p* is the number of parameters fitted. In this example $p = 6$ and $n = 12$, so that the average variance of the fitted value is $\frac{6(1.5)^2}{12} = 1.125$, and the corresponding average standard error of the $\hat{f}$'s range from 77 to 90. Thus in this example (1) we have failed to show any substantial lack of fit, and (2) the predicted change of $\hat{f}$ is 12 times the average standard error of $\hat{f}$. It therefore appears worthwhile[2] to interpret the fitted surface.

## Maxima, Ridges and canonical analyses

The strategy we have outlined (1) uses first-order designs to determine the local slope of the surface, (2) employs steepest ascent to approach a maximal region, and (3) uses a second order

[1] When a particular mode does not fit adequately, instead of immediately considering a higher order polynomial model it is often advantageous to contemplate improvements by transformations of the variables or response, or to consider some entirely different form of model.

[2] A common misapplication of response surface methodology is to interpret an inadequately estimated response function.

design to obtain a local representation of this maximal region. We now concider this fitted second-degree equation somewhat more carefully, using for illustration the case of just two variables. The fitted equation will be of the form

$$\hat{f} = b_0 + b_1 x_1 + b_2 x_2 + b_{11} x_1^2 + b_{22} x_2^2 + b_{12} x_1 x_2$$

Now, depending on the coefficients $b_0, b_1, b_2, b_{11}, b_{22}, b_{12}$, this equation can take a number of different forms.

Canonical analysis consists of (1) shifting the origin to a new point S and (2) rotating the axes so that they correspond to the axes of the contours. The rotated axes with the new origin are labeled $X_1$ and $X_2$. When related to this new system of axes, the various forms of the second-order equations are greatly simplified and the geometrical nature becomes obvious. Our object here is to explain the ideas. Details of the necessary calculations will be found in indicated references.

Consider, for example, the fitted second-degree equation shown on page 20. It may be shown that we can compute a new origin S' lying on the ridge at minimum distance from the design center and also calculate a rotation of axes so that $X_2$ lies along the ridge. The fitted equation then becomes

$$\hat{f} - 88.2 = -5.12 X_1^2 + 2.4 X_2 - 0.15 X_2^2 .$$

In this equation the term $B_{22} = 0.15$ is negligible so that, by inspection of this form of the equation only, we know that the system is closely approximated by a rising ridge. It is also possible to calculate the equation of the ridge:

$$0.63 x_1 + 0.77 x_2 + 0.33 = 0$$

and thus, if we wish, conduct experiments along it.
In this example the canonical analysis merely confirms what we can observe from a contour diagram. The importance of canonical analysis is that it enables us to analyze systems of maxima and minima in many dimensions and, in particular to identify complicated ridge systems, where direct geometric representation is not possible.

Using this method in many dimensions, it is important to make 'reduced experimental designs'. The first-degree design should contain more than $(n+1)$ observations and the second-degree design should contain more than $(n+1)(n+2)/2$ observations. This method does not converge to a local minimizer rather it has a high probability of reaching a major decrease in the objective function within few objective function evaluations.

### 2.3.3 Model based optimization

This section contains information summarized from several sources, i.e. (Booker, Dennis et al. 1999). The author makes himself some of the comments.

#### Basic strategy

Model based optimization solves optimization problems using the basic strategy outlined below.
1) Place a number of starting points
2) Evaluate the objective function at these points
3) Make a model of the objective function using all available points
4) Place a number of points depending on the model and previous function evaluations
   Go to 2

## Generating starting points

Make a designed experiment. There are some choices, D-optimal design, factorial design, space filling design or latin square design. A valuble reference is (Box, Hunter et al. 1977). If a high probability of finding the global optimum is crucial then it is necessary to place many starting points.

## The model

It is a major advantage if the model is fast to evaluate.
If the shape of the objective function is unknown then it is necessary to have a very flexible model that can adjust to any shape of the objective. In that case Moving Least Square or Radial Basis Function regression is preferred, see (Mitchell 1997).

The more that is known about the objective function the more efficient optimization methods can be designed, an example being if the objective is well approximated with a polynomial. Determining the coefficients of the polynomial can then solve the optimization problem. Ideally the function is previously known in its form but a few parameters need to be adjusted. The optimization problem is then solved using a designed experiment with the same number of runs as there are unknown parameters.

If the objective are subject to noise (as in the case when the objective function is constructed from measured data) or a rounding off error (as in the case when the objective function is constructed from computer calculations) it is appropriate to make more runs than there are parameters in the model.

## Method to search the model

If time is not critical the best choice is to search the whole region of interest on a grid. The step length in the grid should be chosen to meet the need of accuracy in the objective. With a grid search the risk of making a local optimization of the model is minimized. A more time efficient method is to use many local optimizers starting at points that are experimentally designed. An appropriate number of optimizers could be 3 times the number of dimensions of the problem.

## Placing the next points in the optimization process

When deciding where to put the next points in the optimization process there are two conflicting demands:

- firstly; we want to improve the accuracy of the model
- secondly; we want to explore the most promising regions suggested by the model.

The conflict can be handled in two ways:

- either; let every new point be a compromise between these to demands
- or; determine the criteria determining when to improve the model and when to explore promising regions.

How the algorithm handles these demands should be governed by the problem solvers need for local or global optimization. Anyway, it is important to start with improving the model and end with exploring interesting regions. The starting points are, as a matter of fact, placed to support the best possible model [3].

[3] In the author's opinion that is almost the same as not placing any starting points and initially improving the model with the same number of points as was originally placed as starting points.

# 3 The described optimization methods ability to handle objective functions with specific characteristics and demands put forward by the user

Depending on the problem that should be solved there are several criteria that should be met by an optimization method. Some valuable characteristics are listed below. If a specific characteristic is very important compared to the others a suggestion of the best and the worst choice of optimization method is made. There hasn't been enough time, or resources to numerically verify these choices so the suggestions are based on the authors own experiences from industrial work.

1) High initial improvement is wanted.
   Best choice; a method that test which variables are of importance, i.e. Response surface methodology
   Worst choice; Multidirectional search

2) High probability of finding the globally best region is wanted.
   Best choice; methods with many well spaced starting points, i.e. Model based optimization.
   Worst choice; methods with one starting point, i.e. Quasi Newton method.

3) It is possible to do objective function evaluations in parallel.
   Best choice; Model based optimization
   Worst choice; Simplex method

4) A convergence proof to a local optimum is wanted.
   Best choice; Multidimensional search, Quasi Newton method or Trust-region methods
   Worst choice; Response surface methodology or Simplex method

5) High efficiency on stochastic objective function is wanted.
   Best choice; Model based optimization
   Worst choice; Quasi Newton method

6) The algorithm should be able to handle discontinuous objective functions.
   Best choice; Simplex method or Multidimensional search
   Worst choice; Model based optimization with a model that can't handle discontinuity

7) High efficiency is wanted when the objective function is unknown (i.e. unknown weights).
   Best choice; Model based optimization
   Worst choice; not any particular

8) The algorithm should be easy to implement.
   Best choice; Simplex method or Multidirectional search
   Worst choice; Response surface methodology or Model based optimization.

9) The objective function is only accurate to a few digits.
   Best choice; a method that uses well spaced points, i.e. Pattern search methods
   Worst choice; a method that uses finite differencing, i.e. Quasi Newton methods

10) The objective function is convex.
   Best choice; a method that converges to a local optima, i.e. Quasi Newton methods
   Worst choice; not any particular

### 3.1 When to use model based optimization

Some remarks on cases when it is appropriate to choose model based optimization are given below.

- First we have the case when it is favorable to evaluate several objective function values at the same time. Such is the case when a large parallel computer is available. Then model based optimization is time efficient since the evaluations to determine the model can be executed in parallel.
- Another instance may be when a large number of optimization problems are to be solved using different constraints or weights in the objective function. This is the case when the objective function can be described as $f(g(x))$ where $g(x)$ is expensive to evaluate and several different $f's$ are to be evaluated. Then a fast model $\hat{g}(x)$, that approximates $g(x)$ sufficiently well, is constructed and many different $f(\hat{g}(x))$ can be evaluated with very little extra effort.
- Model based optimization is furthermore useful when optimizing an objective function that is based on measurements that are subject to noise. The model is in such a case used to diminish the noise in the measured data.
- It is also an advantage that the models that are created by the algorithm can be used to illustrate the optimization problem in a way understandable to the problem solver and others. This helps the problem solver to see through the problem at hand.

## 4 Description of a new model based optimization method

Here a model based optimization algorithm is suggested. It is an implementation based on the method described in section 2.3.3.

### 4.1 Generating starting points

The idea behind this algorithm is to place some well-spaced points that can be used to create the best possible model over the whole region of interest. All points, but one point, are placed on a hypersphere and the remaining point is placed in the center of the hypersphere. The method to spread the points on the hypersphere is by minimizing the maximal correlation between directions to points. The directions are defined from the center-point to the points on the hypersphere. This idea was implemented after suggestions from S Ahlinder who had noticed that almost every experimental design is constructed in such a way that the points are placed in order to minimize the maximal correlation between any two points. The Matlab implementation is named ‘min_corr_design’ and it is presented in appendix 2. The algorithm can shortly be described as follows;

```
Place all, but one, points randomly
Move the points so that the centerpoint is zero

Do for every point
           Do for every possible rotation (every combination of two
           variables)

          If a positive or negative rotation makes the maximal
           correlation between directions decrease, make that rotation.

Stop when time limit is reached or further improvement is not possible.

Place a centerpoint at the origin.
```

Figure 9 shows a typical result when running the ‘min_corr_design’ algorithm in 2 dimensions with 6 points.

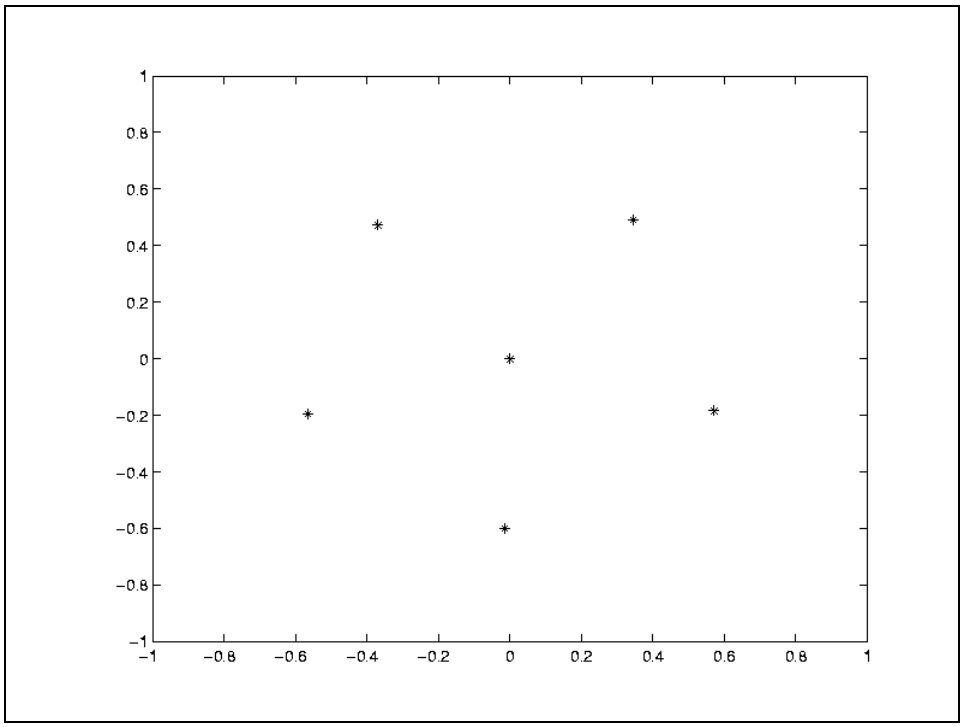


**Figure 9**

## *4.2 The model*

The model that is used here is usually referred to as Moving Least Square or Locally Weighted Regression. A usual method to approximate data with a model is by solving

$$\min_{\beta} \sum_{i=1}^{n} \left( f_i - \hat{f}(x_i, \beta) \right)^2 ,$$

where $f_i$ is the dependent variable and $x_i$ are the corresponding independent variables in the measurements. The model is denoted $\hat{f}$ and it contains the unknown parameters $\beta$ .

In locally weighted regression the weighted variant

$$\min_{\beta} \sum_{i=1}^{n} w_i^2 \left( f_i - \hat{f}(x_i, \beta) \right)^2 \qquad (*)$$

is used. The weights are local because $w_i$ is defined monotonically decreasing as function of the distance between the modeled point $x$ and the measured point $x_i$

$$w_i = w_i(|x - x_i|) .$$

The following conditions also have to be met by *w*.

$$w_i(0) = 1, \qquad w_i(\infty) = 0$$

$$\sum_{i=1}^{n} w_i = \text{number of parameters, } \beta, \text{ in the model}$$

When the minimization problem (*) is solved it results in a linear polynomial, $\hat{f}$, which is used to approximate the objective function. This is done every time the model is evaluated. The model can approximate any continuous objective function sufficiently well because in a small region the continuos objective function will be well approximated by a linear model. When the model is evaluated at *x*, only the same number of points that are needed to determine a linear polynomial will be used. The points used are the ones closest to *x*, so with sufficiently many well-spaced points the points that support the linear model will be situated in a sufficiently small region around *x* that the approximation of *f* is guaranteed to be good. More discussions about Locally Weighted Regression can be found in (Mitchell 1997).

### *4.3 Method to search the model*

The model is searched on a rectangular grid with 11 levels for each variable. It is supposed that the interesting region can be transformed into $-1 \le x \le 1$.

### *4.4 Placing the next points in the optimization process*

In my algorithm a point is placed either at the optimum predicted by the model or at a point that have maximal distance to the nearest evaluated point. A parameter $\lambda$ is used to direct the algorithm in such a way that:

- When $\lambda = 0$ the algorithm places the next point at the optimum predicted by the model, (a local optimization method).
- On the other hand when $\lambda = 1$ the algorithm tries to explore new areas and the next point to evaluate will be the one with maximal distance to nearest evaluated point, (a global optimization method).

Before running the algorithm, $\lambda$ is set by the problem solver.

# 5 Two experiments with the new model based optimization metod

In order to estimate the performance of the new algorithm an optimization problem in two dimensions has been evaluated. Example 1 illustrates how the algorithm works when $\lambda = 0$ and in example 2 the case when $\lambda = 1$ is treated.
The objective function is an interpolation of the points given in Appendix 3. The interpolation was done with the Matlab function griddata. This is a triangle-based linear interpolation which is discontinuous in both the first and zero-th derivative.

## *5.1 Example 1*

**Layout of figures**

In the upper left box the already evaluated points and the next point to be evaluated are shown as stars, with the new point being given above the box. The grid illustrates the mesh points where the model is evaluated.
The upper right box shows the model evaluated (in all the mesh points). The model is always based on all evaluated points. The mesh point that minimizes the model is given above the box.
In the lower left box the true objective function is shown (for all mesh points). The mesh point that minimizes the true objective function is shown above the box.
The lower right box shows the distance to the nearest evaluated point for all mesh points. The mesh point that has the longest distance to the nearest evaluated point is shown above the box.

Figure 10 shows the results after the first iteration. The model is based on 6 starting points. These starting points are shown in the upper left box. They are not located in such away that they coincides with the mesh points. The information about distance to nearest evaluated point is not utilized in example 1.

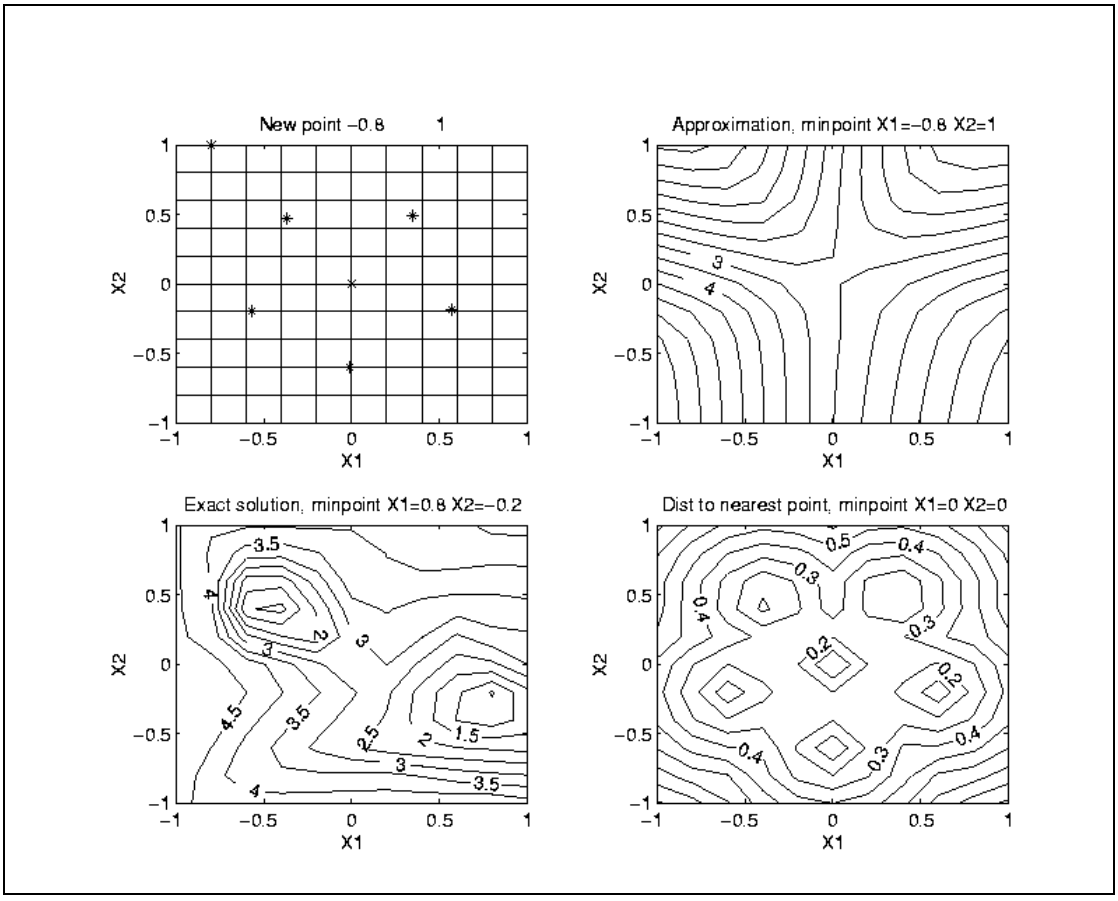


**Figure 10 Numerical experiment, example 1, iteration 1**

In Figure 10 the 6 starting points are symmetrically spaced in order to get the best chance of catching the basin that contains a global optimum. The model in the upper right box is visualized with contour lines. Only the lines on level 3 and 4 are marked. The model predicts high values at (x1=low, x2=low) and (x1=high, x2=high), low values are predicted at (x1=high, x2=low) and (x1=low, x2=high). When the mesh points are searched the minimum to the model was found at (x1=-0.8, x2=1).

The next iteration where all 7 evaluated points are used to make up the model is shown in Figure 11.

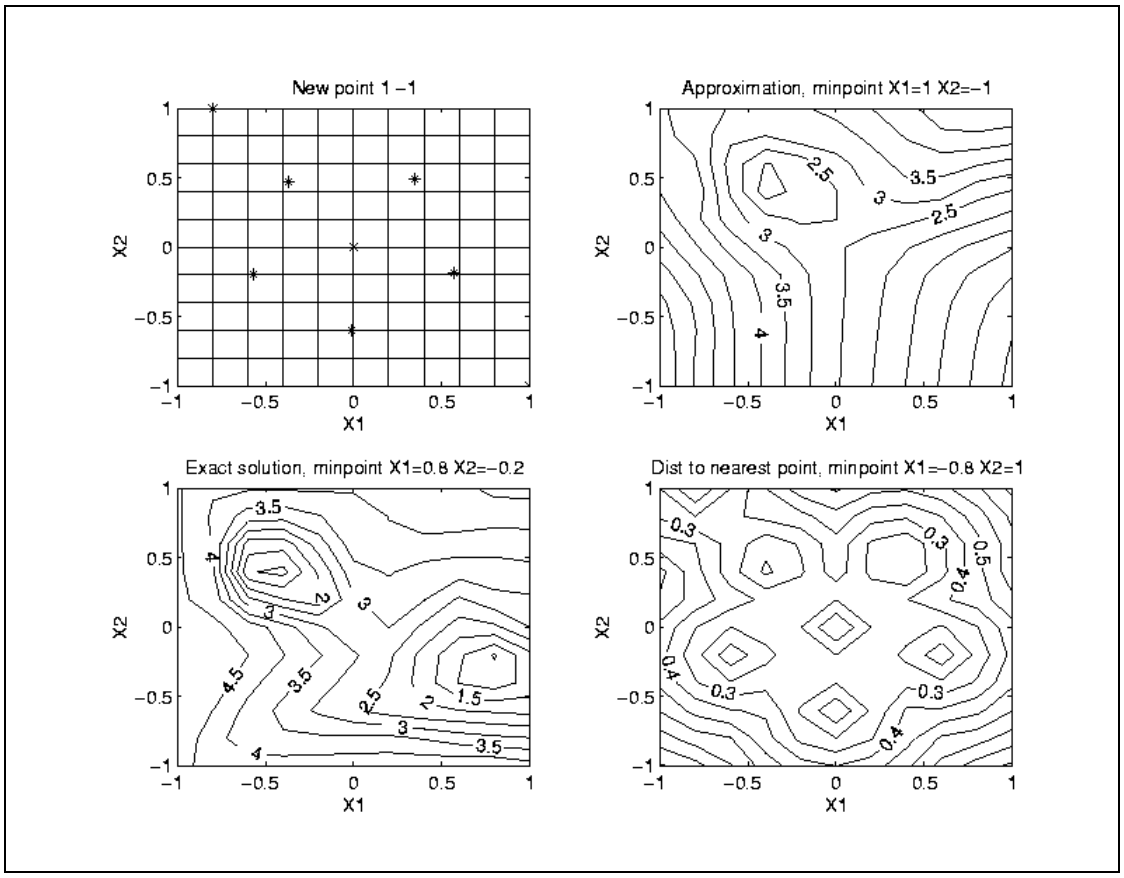


**Figure 11 Numerical experiment, example 1, iteration 2**

The point (x1=-0.8, x2=1) reveals that the corner where x1 is low and x2 is high is not the optimal solution. A basin seems however to be situated in the neighborhood of (x1=-0.5, x2=0.5) and we can easily verify that this is the case by comparing with the true objective function in the lower left box.The corner where (x1=high, x2=low) is still unexplored and according to the model this is a very promising region. So when the model is searched the optimum is found at (x1=1, x2=-1). This point is hardly visible in Figure 11 but there is an indication at (x1=1, x2=-1) in the upper left box.

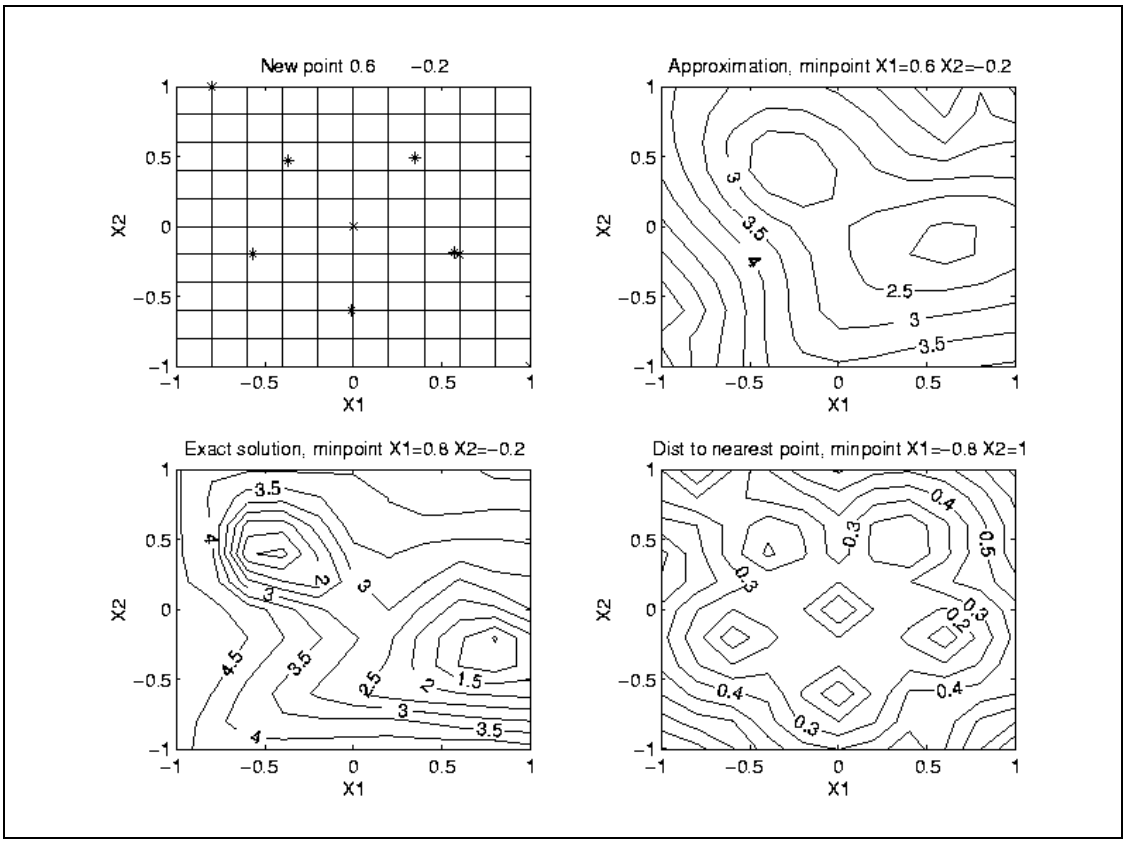


**Figure 12 Numerical experiment, example 1, iteration 3**

When the point at (x1=1, x2=-1) is used to update the model the result is shown in the upper right box in Figure 12. In very few (8) objective function evaluations the algorithm has detected the two basins in the objective function. At the present stage the most promising point is located at (x1=0.6, x2=-0.2); this point is very close to one of the starting points, but the algorithm did not detect this so in spite of this fact the point is evaluated.

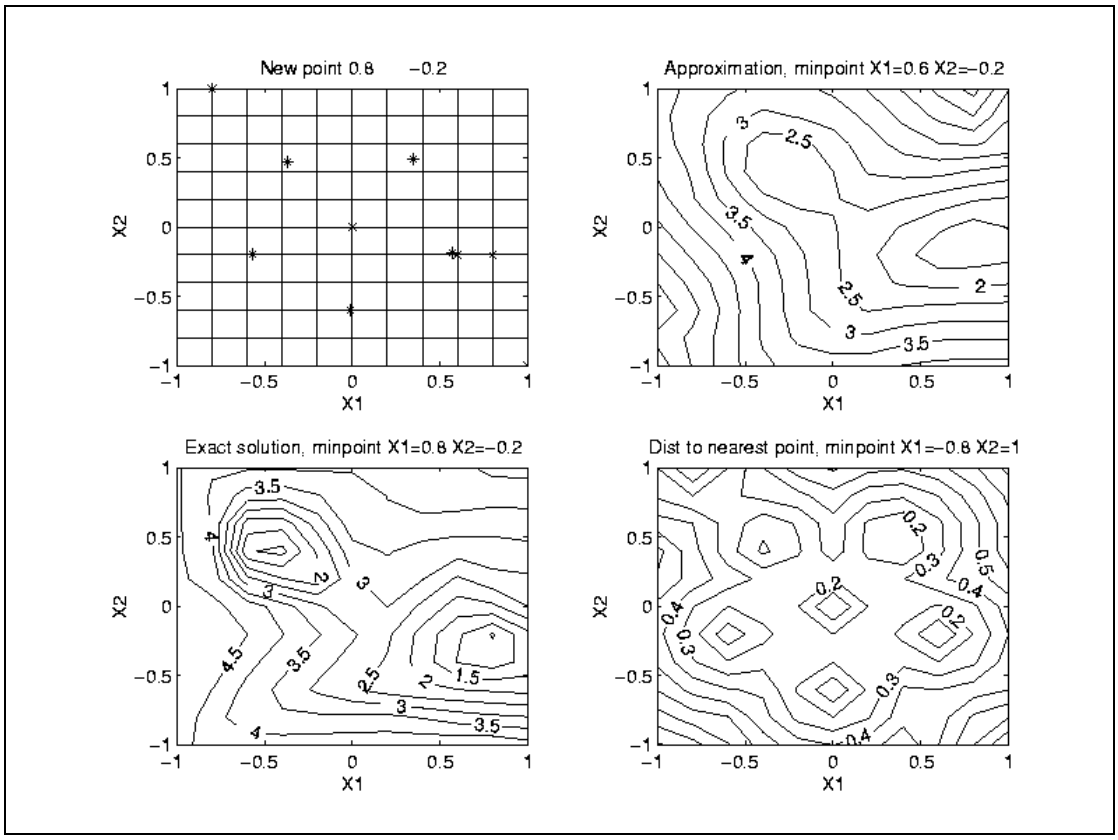


**Figure 13 Numerical experiment, example 1, iteration 4**

After the point in (x1=0.6, x1=-0.2) had been evaluated, the new status is as shown in Figure 13. As expected the model hasn't changed much since the previous iteration. The optimum of the model is still predicted to be in (x1=0.6, x2=-0.2); then the algorithm chooses to evaluate the second best mesh point. According to the model that is (x1=0.8, x2=-0.2).
This point is in fact the global optimum of the objective function.

### 5.1.1 Observations

The algorithm detects promising regions very early in the process, but the depth of these regions is not especially well predicted.

In the 4'th iteration the algorithm suggests the next point to be a point which already has been evaluated. This is maybe a good stopping criterion or a stage where the strategy should be reconsidered.

If the model deviated much from the objective function where the next point is to be placed the algorithm takes a large step forward towards the objective function in the region under evaluation.

When there only are a few objective function values then the approximation will be close to linear and therefore often has the optimum in a corner. Many corners will be searched until the model can have its optimum inside the region. This is a potential problem when *n* is large, since number of corners are $2^n$.

## 5.2 Example 2

By setting $\lambda = 1$ the next point to evaluate will be chosen as the point with the largest distance to the nearest evaluated point. In doing so an approximation of a space filling design is sequentially reached. The primary goal of this strategy is to find new interesting areas to explore. The first point is placed at (x1=1, x2=-1), a little indication is shown in Figure 14.

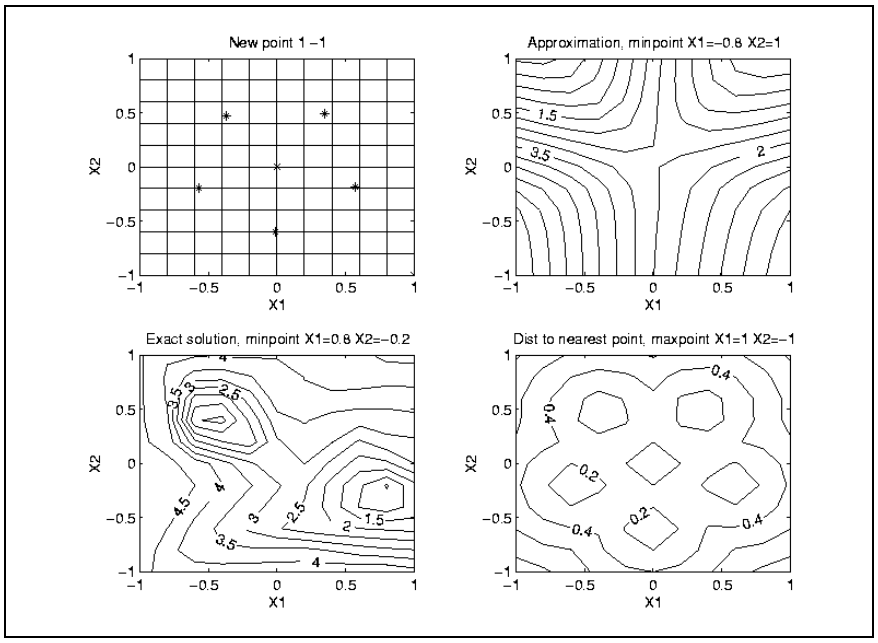


**Figure 14 Numerical experiment, example 2, iteration 1**

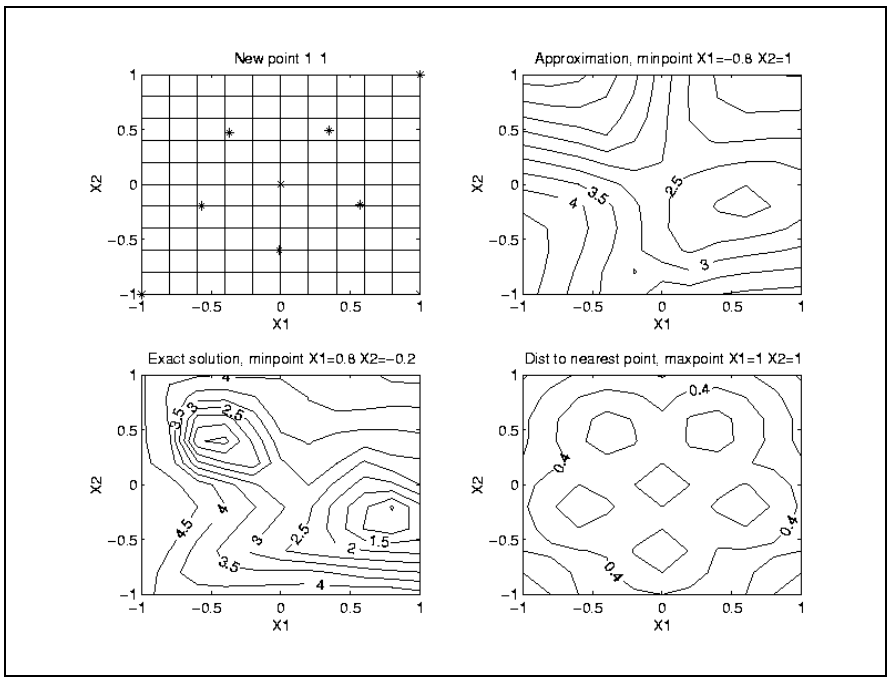


**Figure 15 Numerical experiment, example 2, iteration 3**

After evaluating 9 points (the 6 starting points included) the algorithm detects and localizes the basin to the right, see Figure 15.

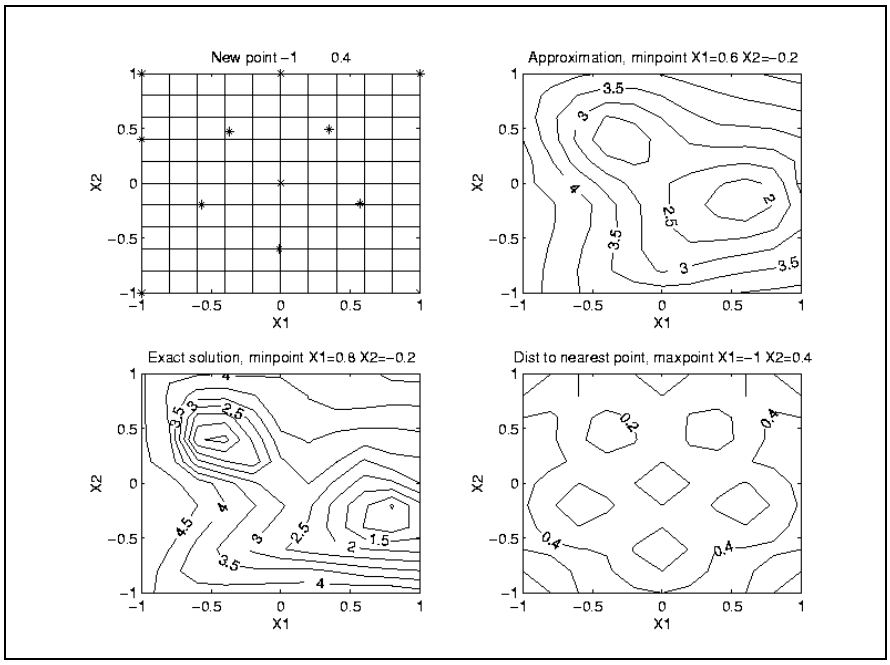


**Figure 16 Numerical experiment, example 2, iteration 6**

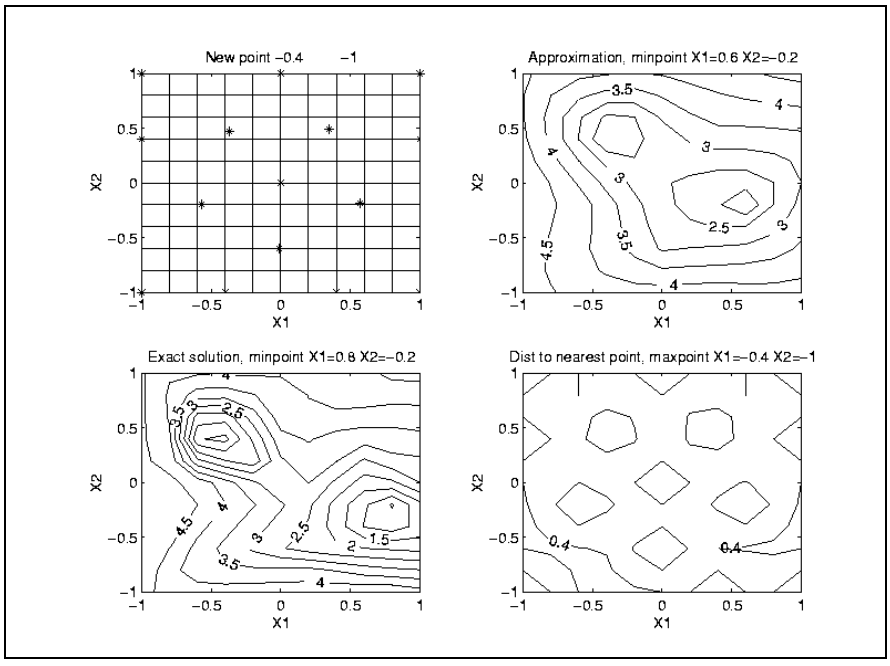


**Figure 17 Numerical experiment, example 2, iteration 9**

After evaluating 12 points (Figure 16) the algorithm has detected both basins. At this stage the objective function is modeled with good agreement over the whole region and little further improvement can be expected, see Figure 17.

### 5.2.1 Observations

Two favorable characteristics of the method are shown in example 2. Firstly the method could detect new interesting regions after one minimum had been fully explored. Secondly, the method can be used to sample an object and continuously monitor how large the difference is between the next predicted model value and the objective function. After that a number of consecutive points

have been predicted with good precision the sampled system is represented with sufficient accuracy. Then optimization is carried out on the model, which is a great advantage.

# 6 Conclusions and future work

## *6.1 Conclusions*

Optimization using models is indeed a very flexible way to construct an optimization algorithm. Since the algorithm can be tailor-made to a certain problem it can be adjusted to perform both rapidly and adequately on that specific problem.

The model based algorithm uses all the objective function values that have been evaluated so far in every step. No other optimization method described in this report does that. But this does not result in faster optimization methods, as indicated in Appendix 1.

Here I will try to answer the question: ”Why is it not improving the algorithm if every already evaluated point is considered equally when choosing the next point to be evaluated?”

The problem is similar to the one described above where a choice has to be made between a local and a global optimization algorithm. If one uses all evaluated points in each iterate then one has constructed a global optimization method. If a local optimization method is wanted instead, the points that support the model should be chosen so that predictions will be the best possible in the most promising regions, and not elsewhere.

If one wants to have an algorithm with the highest convergence speed and little fuzz in a sequential case then the reasonable choice is not really a model based algorithm. If however objective function evaluations can be made in parallel or if we want to solve many different optimization problems with only minor changes in the objective function or if we have the case when the objective function is subject to noise, then the model based algorithm might be the best choice.

## *6.2 Future work*

The model based optimization algorithm should be tested on several engineering optimization problems and the characteristics of these problems should be examined: Some characteristics that should be considered are mentioned in section 3.

The model based optimization method should be improved. The questions to be addressed in particular are:

- Where should the next objective function evaluation take place?
- How could a stable continuous approximation of the objective function be reached?

It should be possible to measure an optimization algorithm’s ability to handle a certain problem with specific characteristics. If the ability to handle the characteristics in section 3 could be quantified it should facilitate the choice of the appropriate optimization method.
If a well-defined problem for each characteristic could be constructed then it should be possible to compare all available optimization methods for this characteristic by running the algorithms on this problem. Such an achievement would be welcome.

## References


Booker, A. J. (1994). D O E for computer output. Technical Report BCSTECH-94-052, Boeing Computer Services.

Booker, A. J., J. E. Dennis, et al. (1999). "A Rigorous Framework for Optimisation of Expensive Functions by Surrogates." Structural Optimization **17**(1): 1-13.

Box, G. E. P., W. G. Hunter, et al. (1977). Statistics for Experimenters. An Introduction to Design, Data Analyses, and Model Building. New York, John Wiley & Sons.

Box, G. E. P. and J. Wetz (1973). Criteria for Judging Adequacy of Estimation by an Approximating Response Function. University of Wisconsin-Madison, Department of Statistics.

Conn, A. R., N. I. M. Gould, et al. (2000). Trust-Region Methods. Philadelphia, MPS-SIAM Series on Optmization.

Conn, A. R. and P. L. Toint (1996). An algorithm using quadratic interpolation for unconstrained derivative free optimization. Nonlinear Optimization and Applications. G. D. Pillo and F. Giannessi, Plenum: 27-47.

Dennis, J. E. and V. Torczon (1991). "Direct search methods on parallel machines." SIAM Journal on Optimization **1**(4): 448-474.

Dixon, I. C. W. (1972). Nonlinear Optimization. London, The English Universities Press Ltd.

Dixon, I. C. W. and G. P. Szegö (1978). Towards Global Optimization 2. Amsterdam, North Holland.

Himmelblau, D. M. (1972). Applied Nonlinear Programming. New York, McGRAW-HILL.

Mitchell, T. J., J. Sacks, et al. (1989). "Design and analysis of computer experiments." Statistical science **4**(4): 409-435.

Mitchell, T. M. (1997). Machine Learning. New York, McGraw-HILL.

Morris, M., C. Currin, et al. (1991). "Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments." Journal of the American Statistical Association **86**(416): 953-963.

Nash, S. G. and A. Sofer (1996). Linear and Nonlinear Programming. Singapore, McGRAW-HILL.

Nelder, J. A. and R. Mead (1965). "A Simplex method for function minimization." Computer Journal **7**: 308-313.

Owen, A. B. (1992). "Orthogonal arrays for computer experiments, integration and visualisation." Statistica Sinica **2**: 439-452.

Polyak, B. T. (1987). Introduction to Optimization. New York, Optimization Software Inc.

Sacks, J., H. P. Wynn, et al. (1992). "Screening predicting and computer experiments." Technometrics **34**(1): 15-25.

Torczon, V. (1989). Multi-directional search: A direct search algorithm for parallell machines. Department of Mathematical Sciences. Houston, Rice University.

Torczon, V. (1991). "On the convergence of the multidirectional search algorithm." SIAM Journal on Optimization **1**(1): 123-145.

## Notation

| | | |
|---|---|---|
| $\alpha$ | (1) | Step-length for standard Newton method $\alpha \equiv 1$ |
| | (2) | The reflection coefficient in the Nelder Mead simplex algorithm |
| $\beta$ | (1) | The contraction coefficient in the Nelder Mead simplex algorithm |
| | (2) | Coefficients in the Response Surface model |
| $\Delta$ | | The size of the trust region in the Trust-region algorithm |
| $\gamma$ | | The expansion coefficient in the Nelder Mead simplex algorithm |
| $\rho$ | | Actual reduction divided by predicted reduction in the Trust-region algorithm |
| $\varphi$ | | The model in the Trust-region algorithm |

| | | |
|---|---|---|
| $B$ | | Approximation of $\nabla^2 f(x)$ |
| $c$ | | Contraction vertex in the Multidirectional search algorithm |
| $e$ | | Expansion vertex in the Multidirectional search algorithm |
| $f$ | | Objective function |
| $g$ | | Any function |
| $k$ | | Iteration number |
| $l$ | | Lower limit for $x$ |
| $n$ | | Dimension of $x$ |
| $p$ | (1) | Step direction, the solution to $B_k p = -\nabla f(x_k)$ in the Newton algorithm |
| | (2) | Step $(x_{k+1} - x_k)$ in the Trust-region algorithm |
| P | | Points in the Nelder Mead simplex algorithm |
| $r$ | | Reflected vertex in the Multidirectional search algorithm |
| $u$ | | Upper limit for $x$ |
| $t$ | | Time in Response Surface Method |
| $T$ | | Temperature in Response Surface Method |
| $v$ | | A vertex in the Multidirectional search algorithm |
| $x$ | | Any variable |
| $x_*$ | | Local minimum to $f(x)$ |

# Appendix

# Appendix 1 Benchmark

The Hartman function with 6 variables as presented in (Dixon and Szegö 1978) has been used as test-problem. The Matlab routine that is used to evaluate this function is shown in Figure 18.

```
function f = hartman6(x)
% f = hartman6(x)
% x is a 6 element row vector 0 < xi < 1
% f is scalar
% The six variable Hartman problem
% A testproblem in global optimization
% Discribed in Towards Global Optimization 2
% edited by L.C.W. Dixon and G.P. Szegö, North-Holland 1978
[row,col]=size(x);
if row~=1 | col ~=6
   error('argument should be of size 1x6')
end

m=4;
a=[10  3   17  3.5 1.7 8
   .05 10  17  .1  8   14
   3   3.5 1.7 10  17  8
   17  8   .05 10  .1  14];

c=[1 1.2 3 3.2]';

p=[.1312 .1696 .5569 .0124 .8283 .5886
   .2329 .4135 .8307 .3736 .1004 .9991
   .2348 .1451 .3522 .2883 .3047 .6650
   .4047 .8828 .8732 .5743 .1091 .0381];

f = 0;
for i = 1 : m
f=f - c(i) * exp(- sum( a(i,:) .* ( ( x - p(i,:) ).^2 ) ) );
end
```

**Figure 18**

In (Booker, Dennis et al. 1999) the Hartman function is minimized with two model based optimization algorithms (MMF/MP) and (MMF/DACE) as well as one Trust-region algorithm (DFO). All these optimization algorithms belong to the third class, see page 13.

To be able to compare these algorithms with standard optimization methods such as Newton type methods (first class) and the Nelder Mead Simplex method (second class), three algorithms, FMINS, FMINU, and CONSTR, included in Matlabs optimization toolbox, were tested.

FMINS is a simplex type algorithm described in (Nelder and Mead 1965). If the optimization problem is $n$ dimensional, the Simplex algorithm uses n+1 points to search for better solutions as described on page 7. FMINU and CONSTR are Newton type algorithms. This type of algorithm is described on page 3 and more detailed description can be found in (Nash and Sofer 1996). The difference between FMINU and CONSTR is that CONSTR is especially designed to handle constraints.

As shown in Figure 19, MMF/MP and MMF/DACE both reached the objective value -3 in 50 evaluations and they converged to the optimum after 70 evaluations. As shown in Figure 19 the convergence speeds of both MMF/MP and MMF/DACE are comparable to FMINU. DFO converged after 100 evaluations which is comparable to CONSTR.

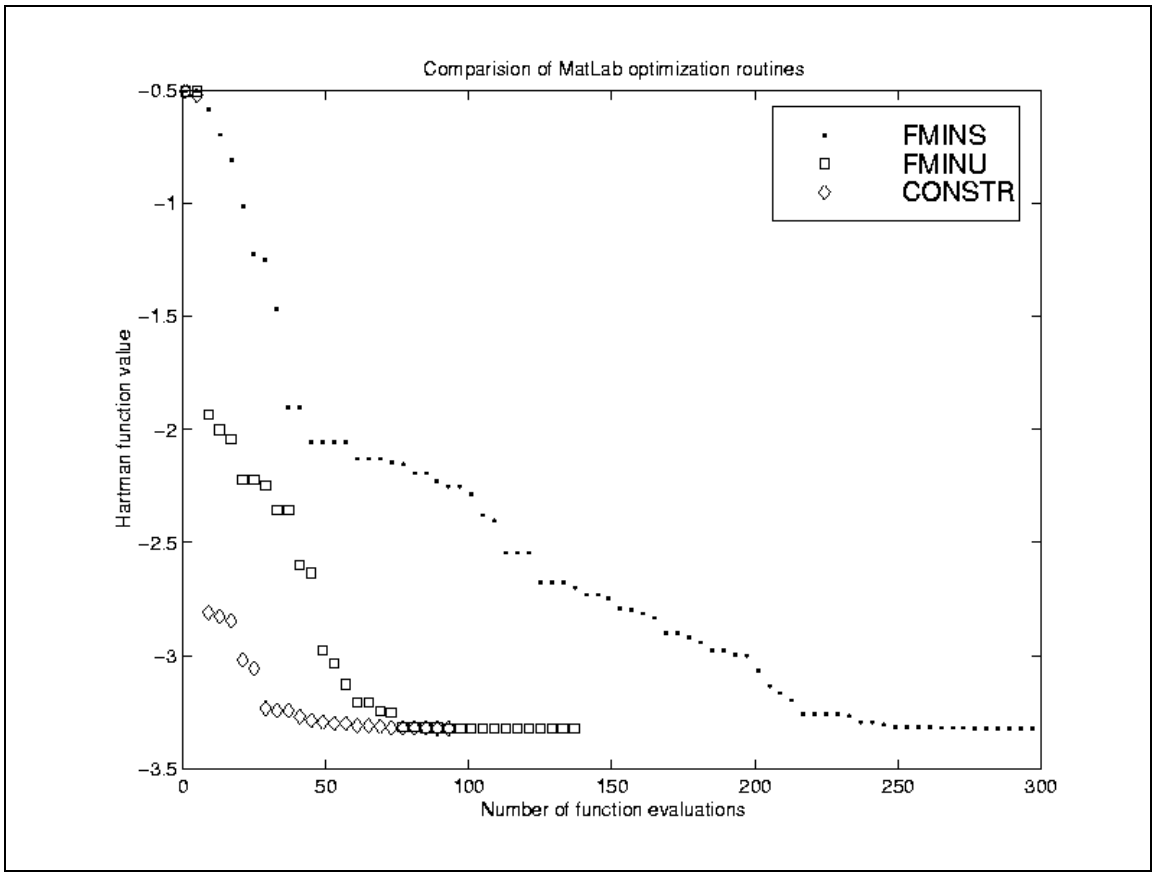


**Figure 19**

The above results show that these model based algorithms do not converge faster then standard Newton type algorithms on a mathematical test-problem. Further does the Nelder Mead Simplex algorithm converge slower then the Newton type algorithms on this problem.

## Appendix 2 Matlab implementation of min_corr_design

```
function [X,corr_mat]= min_corr_design(n_var,n_points)
% [X,corr_mat]=gen_tetraed(n_var,n_points)
% fast if n_var+1 == n_points
% otherwise iterates until tolerans tol or toltimelimit maxtime is achieved
% corr_mat holds the correlation number for every combination of points


if n_var+1==n_points
            c=cos(15*pi/180);s=sin(15*pi/180);
            X=c^(n_var-1)*s*ones(n_var,n_var);
            for i=1:n_var
                        X(i,i)=c^n_var;
            end
            X=[zeros(1,n_var);X];


   % center the near tetraed
            for i=1:n_var
                        X(:,i) = X(:,i)-sum(X(:,i))/n_points;
            end


   %normalize all vectors
   for i=1:n_points
            X(i,:) = X(i,:) / (sqrt(sum(X(i,:).^2)));
            end


            corr_mat=0;
   ii=0;
   for i = [1:n_points-1]
      for j= i+1:n_points
         ii=ii+1;
                  corr_mat(ii)= ( X(i,:)*X(j,:)' ) /( norm(X(i,:))*norm(X(j,:)) );
               end
   end


else
   tic
            tol = 0.05
            maxtime = 120
            %rotating angle
            ang = 5


            giv_pos = [cos(ang*pi/180) sin(ang*pi/180)
                        -sin(ang*pi/180) cos(ang*pi/180)];


            giv_neg = [cos(ang*pi/180) -sin(ang*pi/180)
                        sin(ang*pi/180) cos(ang*pi/180)];


   X=rand(n_points,n_var)*2-1;

            % normailze all vectors
            for i=1:n_points
               X(i,:) = X(i,:) / (sqrt(sum(X(i,:).^2)));
            end


            corr_mat=[1 0]
            while max(max(corr_mat))-min(min(corr_mat)) > tol & toc < maxtime
                        % do for every point
```

```
                        for p = 1:n_points
                           corr_sum = cal_corr_sum (X,p);


                           % do for every rotating direction


                           for i=1:n_var-1
                              for j=i+1:n_var
                                 rot_mat = eye(n_var);

                                 rot_mat([i,j],[i,j]) = giv_pos;
                        Xpos = X;
                        Xpos(p,:)=( rot_mat * X(p,:)' )';
                                 pos_corr_sum = cal_corr_sum (Xpos,p);


                                 rot_mat([i,j],[i,j]) = giv_neg;
                                 Xneg = X;
                                 Xneg(p,:)=( rot_mat * X(p,:)' )';
                                 neg_corr_sum = cal_corr_sum (Xneg,p);

                                 if pos_corr_sum < corr_sum
                                    X = Xpos;
                                    corr_sum = pos_corr_sum;
                                 elseif neg_corr_sum < corr_sum
                                    X = Xneg;
                                    corr_sum = neg_corr_sum;
                                 end
                              end

                           end
                        end
                        %calculate correlation between every point
                  corr_mat=0;
                  ii=0;
                  for i = [1:n_points-1]
                     for j= i+1:n_points
                        ii=ii+1;
                              corr_mat(ii)= ( X(i,:)*X(j,:)' ) /(
norm(X(i,:))*norm(X(j,:)) );
                           end
                        end

               end
end
```

## Appendix 3 Objective function data

| x1 | x2 | f(x) |
|---|---|---|
| -1.13 | 1.15 | 5.00 |
| 1.18 | 1.15 | 5.00 |
| 1.25 | -1.15 | 5.00 |
| -1.15 | -1.25 | 5.00 |
| -0.57 | 0.50 | 0.00 |
| -0.53 | 0.83 | 3.00 |
| -0.81 | 0.45 | 4.00 |
| -0.70 | -0.04 | 5.00 |
| -0.73 | -0.70 | 4.00 |
| -0.28 | -0.63 | 3.00 |
| 0.32 | -0.58 | 2.00 |
| 0.10 | -0.10 | 3.00 |
| 0.05 | 0.60 | 4.00 |
| 0.54 | 0.32 | 3.00 |
| 0.86 | -0.28 | 0.00 |
| 1.24 | -0.27 | 2.00 |