Implementation of a One-Stage Efficient Global Optimization (EGO) Algorithm

Nils-Hassan Quttineh* and Kenneth Holmström*

Abstract Almost every Costly Global Optimization (CGO) solver utilizes a surrogate model, or response surface, to approximate the true (costly) function. The EGO algorithm introduced by Jones et al. utilizes the DACE framework to build an approximating surrogate model. By optimizing a less costly utility function, the algorithm determines a new point where the original objective function is evaluated. This is repeated until some convergence criteria is fulfilled. The original EGO algorithm finds the new point to sample in a two-stage process. In its first stage, the estimates of the interpolation parameters are optimized with respect to already sampled points. In the second stage, these estimated values are considered true in order to optimize the location of the new point. The use of estimate values as correct introduces a source of error. Instead, in the one-stage EGO algorithm, both the parameters and the location of a new point are optimized at the same time, removing the source of error. This new subproblem becomes more difficult, but eliminates the need of solving two subproblems. Difficulties in implementing a fast and robust One-Stage EGO algorithm in TOMLAB are discussed, especially the solution of the new subproblem.

Keywords: Black-box, Surrogate model, Costly functions, Latin Hypercube Designs, Experimental Design.

1 Introduction

Global optimization of continuous black-box functions that are costly (computationally expensive, CPU-intensive) to evaluate is a challenging problem. Several approaches based on response surface techniques, most of which utilize every computed function value, have been developed over the years.

Problems that are costly to evaluate are commonly found in engineering design, industrial and financial applications. A function value could be the result of a complex computer program, an advanced simulation, e.g. computational fluid dynamics (CFD).

*Department of Applied mathematics, Mälardalen University, SE-721 23 Västerås, Sweden.

One function value might require the solution of a large system of partial differential equations, and hence consume anything from a few minutes to many hours. In the application areas discussed, derivatives are most often hard to obtain and the algorithms make no use of such information.

From an application perspective there are often restrictions on the variables besides the lower and upper bounds, such as linear, nonlinear or even integer constraints. These complicated problems are formulated as follows:

The Mixed-Integer Costly Global Black-Box Nonconvex Problem

$$\begin{aligned} \min_x \quad & f(x) \\ s/t \quad & \begin{array}{rccccl} -\infty < & x_L & \leq & x & \leq & x_U & < \infty \\ & b_L & \leq & Ax & \leq & b_U & \\ & c_L & \leq & c(x) & \leq & c_U & \\ & & x_j \in \mathbb{N} & \forall j \in \mathbb{I} & & , & \end{array} \end{aligned} \tag{1}$$

where $f(x) \in \mathbb{R}$ and $x_L,\ x,\ x_U \in \mathbb{R}^d$. Matrix $A \in \mathbb{R}^{m_1 \times d}$, $b_L,\ b_U \in \mathbb{R}^{m_1}$; defines the m_1 linear constraints and $c_L,\ c(x),\ c_U \in \mathbb{R}^{m_2}$ defines the m_2 nonlinear constraints. The variables x_I are restricted to be integers, where $\mathbb{I}$ is an index subset of $\{1,\ldots,d\}$.

1.1 Surrogate Models

One approach for solving CGO problems is to utilize response surfaces. The basic idea is to start with an initial sample of points (experimental design), where the costly function values are calculated. A surrogate model (response surface) is built from the sampled points, using for example radial basis functions (RBF) or the DACE framework. Use this interpolated surface to approximate the true function, and decide a new point to sample by some algorithmic strategy. Then iterate until some convergence criteria is fulfilled.

Surrogate Model Algorithm (pseudo-code)

- ▷ Find initial set of $n \geq d+1$ points $\mathbf{x}$ using Experimental Design.
- ▷ Compute costly $f(x)$ for initial set of n points. Best point (x_{Min}, f_{Min}).
- ▷ Use the sampled points $\mathbf{x}$ to build a response surface model as an approximation of the $f(x)$ surface.
- ▷ Choose next point x_{n+1} to be added:
  - Decided by the algorithm used, like EGO, ARBF or rbfSolve.
  - Update best point (x_{Min}, f_{Min}) if $f(x_{n+1}) < f_{Min}$.
- ▷ Iterate until f_{Goal} achieved, $n > n_{Max}$ or maximal CPU time used.

2 Background to DACE and EGO

Suppose we want to predict the function value at a point $\bar{x}$ not already sampled. The DACE framework, short for "Design and Analysis of Computer Experiments" and also referred to as Kriging, is based on modeling a function as a realization of random variables, normally distributed with mean μ and variance σ^2.

The original EGO algorithm, introduced by Jones, Schonlau and Welch [7] in 1998, is a two-stage method. Such methods fit a response surface to sampled points in the first step, then utilize the surface to find new search points in the second step.

Some years later, Jones presents the idea of a one-stage approach [6]. Except for an implementation by Jones himself, the one-stage approach have not been explored.

2.1 The Correlation function

Before going into mathematical details and formulation of the algorithm, we introduce some notations and variables to be used throughout this paper. Compared with Euclidean distance, where every variable is weighted equally, the distance formula

$$D\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \;=\; \sum_{k=1}^{d} \theta_k \cdot \left| x_k^{(i)} - x_k^{(j)} \right|^{p_k} \qquad\qquad \theta_k > 0 \;,\; p_k \in [1,2] \tag{2}$$

is designed to capture functions more precise. The parameter θ_k can be interpreted as a measure of the importance of variable x_k. Even small changes in x_k might lead to large differences in the function values at $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$.

The exponent p_k is related to the smoothness of the function in the k:th dimension. Values of p_k near 2 corresponds to smooth functions and values near 1 to less smoothness. Based on the distance formula (2), the correlation function

$$Corr\left[\epsilon(\mathbf{x}^{(i)}), \epsilon(\mathbf{x}^{(j)})\right] \;=\; e^{-D\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)} \tag{3}$$

has all the intuitive properties one would like it to have. When the distance between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is small, the correlation is close to one. For large distances, the correlation approaches zero. A large value for θ will affect the distance to grow faster, which leads to a decrease in the correlation. In this way active variables are accounted for, giving a more accurate correlation function.

We denote the evaluated function values by $\mathbf{y} = \left(f(\mathbf{x}^{(1)}), \ldots, f(\mathbf{x}^{(n)})\right)'$, where n is the number of points sampled so far. Denote the matrix of correlation values by $\mathbf{R}$, where $\mathbf{R}(i,j) \;=\; Corr\left[\epsilon(\mathbf{x}^{(i)}), \epsilon(\mathbf{x}^{(j)})\right]$. Also, let $\mathbf{r}$ denote the vector of correlations between $\bar{x}$ and the sampled points, that is $r_i(\bar{x}) \;=\; Corr\left[\epsilon(\bar{x}), \epsilon(\mathbf{x}^{(i)})\right]$.

2.2 The DACE interpolation model

To build a surrogate model from the sampled points, we need to estimate parameters θ_k and p_k. This is done by choosing them to maximize the likelihood of the sampled points. A more detailed analysis is found in Section 3.

With the parameter estimates in hand, we are now able to construct the DACE model, or DACE response surface. Using the evaluated function values $\mathbf{y}$ and the matrix of correlation values $\mathbf{R}$, the DACE interpolant is defined by

$$y(\bar{x}) \;=\; \mu + \mathbf{r}'\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\mu) \tag{4}$$

where $\mathbf{r}$ is the vector of correlations between $\bar{x}$ and the sampled points $\mathbf{x}$. The first term μ is the estimated mean, and the second term represents the adjustment to this prediction based on the correlation of sampled points $\mathbf{x}$. The derivation of this predictor can be found in [10].

The mean square error of the predictor, denoted by $s^2(\bar{\mathbf{x}})$, is given by

$$s^2(\bar{x}) \;=\; \sigma^2 \cdot \left[1 - \mathbf{r}'\mathbf{R}^{-1}\mathbf{r} + \frac{(1 - \mathbf{1}'\mathbf{R}^{-1}\mathbf{r})^2}{(\mathbf{1}'\mathbf{R}^{-1}\mathbf{1})}\right]. \tag{5}$$

As one would imagine, the mean square error for a sampled point is zero, $s^2(\mathbf{x}^{(i)}) = 0$.

We now have a formula for the expected value of a point yet to be sampled, along with an error estimation. Since this predictor is cheap to calculate, compared to the costly function, it can be used to locate a new point $\bar{x}$ with as low expected function value as possible.

3 The EGO algorithm

The Efficient Global Optimization (EGO) algorithm utilizes the DACE interpolation surface to approximate the costly function based on already sampled points. In order to use this for optimization, one must find a way to iteratively choose x^*, the next point to sample.

In the original EGO algorithm, described in Section 3.1, the Maximum Likelihood Estimation (MLE) of the parameters θ and p is based on the set of sampled points $\mathbf{x}$. These estimates are used to decide upon new sample points, found by optimizing some utility function, which hopefully converges towards the global optimum. This can be seen as a two-stage process.

A drawback with this approach is that the utility function depends on the estimated parameters, which might lead to inaccurate decisions. To overcome this flaw, the one-stage process described in Section 3.2 incorporates x^* into the MLE step. For a given target value f^*, the MLE tries to fit the surface to already sampled points, conditional upon the assumption that the surface goes through the point (x^*, f^*).

3.1 Two-stage process, Standard EGO

We need to estimate the parameters θ_k and p_k in order to construct an interpolation surface of our costly function. The estimates are found by choosing them to maximize the likelihood of the already sampled points $\mathbf{x}$.

The likelihood function

$$L(\theta, p) \;=\; \frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}|\mathbf{R}|^{\frac{1}{2}}} \cdot e^{-\frac{(\mathbf{y}-\mathbf{1}\mu)'\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2}} \tag{6}$$

Note that the dependence on parameters θ and p is via the correlation matrix $\mathbf{R}$. Assuming we know the values of θ and p, we find the values of μ and σ^2 that maximizes the log-likelihood function:

$$LL(\theta, p) \;=\; -\frac{n}{2}\log(\sigma^2) - \frac{1}{2}\log(|\mathbf{R}|) - \frac{(\mathbf{y}-\mathbf{1}\mu)'\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2} \tag{7}$$

By taking the derivatives of (7) with respect to μ and σ^2 respectively, solving them equal to zero, the solution is:

$$\hat{\mu} \;=\; \frac{(\mathbf{1}'\mathbf{R}^{-1}\mathbf{y})}{(\mathbf{1}'\mathbf{R}^{-1}\mathbf{1})} \tag{8}$$

and

$$\hat{\sigma}^2 \;=\; \frac{(\mathbf{y}-\mathbf{1}\hat{\mu})'\mathbf{R}^{-1}(\mathbf{y}-\mathbf{1}\hat{\mu})}{n} \tag{9}$$

Substituting equations (8) and (9) back into (7), one finds the concentrated log-likelihood function only depending on parameters θ and p via $\mathbf{R}$:

$$ConLL(\theta, p) \;=\; -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{R}|) \tag{10}$$

Maximizing (10) yields the estimates needed. Then use equations (8) and (9) to calculate the estimates of μ and σ^2.

Once the estimates are known, a utility function is optimized in order to find x^*, the next point to sample. There are many different utility functions that could be used, but the Expected Improvement (ExpI) is most commonly used.

The Expected Improvement relies on the predicted values of μ and σ^2 to find the location x^* where the probability of improving the objective value is maximized. For details on different utility functions and the Expected Improvement, see [11].

3.2 One-stage EGO, new approach

In his paper [6], Jones introduces the idea of a one-stage EGO algorithm, incorporating the new point x^* into the estimation process of parameters θ and p. The estimates are, like before, found by choosing them to maximize the likelihood of the sampled points $\mathbf{x}$. But this time we compute the likelihood of the observed data conditional upon the assumption that the surface goes through the point (x^*, f^*).

The conditional likelihood function

$$CL(\theta, p, x^*) \;=\; \frac{1}{(2\pi)^{n/2}(\sigma^2)^{n/2}|\mathbf{C}|^{\frac{1}{2}}} \cdot e^{-\frac{(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)'\mathbf{C}^{-1}(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)}{2\sigma^2}} \tag{11}$$

where

$$\mathbf{C} = \mathbf{R} - \mathbf{r}\mathbf{r}' \;, \quad \bar{\mathbf{y}} = \mathbf{y} - \mathbf{r} \cdot f^* \;, \quad \bar{\mathbf{r}} = \mathbf{1} - \mathbf{r}.$$

The dependence on the parameters θ, p and x^* is via the conditional correlation matrix $\mathbf{C}$, as vector $\mathbf{r}$ describes correlation between x^* and the n sampled points. Both $\mathbf{R}$ and $\mathbf{r}$ are affected by θ and p.

When using the conditional log-likelihood to evaluate the hypothesis that the surface passes through (x^*, f^*) we also estimate the values of θ and p, and like before find the values of μ and σ^2 that maximizes the conditional log-likelihood function:

$$CLL(\theta, p, x^*) \;=\; -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{C}|) - \frac{(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)'\mathbf{C}^{-1}(\bar{\mathbf{y}} - \bar{\mathbf{r}}\mu)}{2\sigma^2} \tag{12}$$

with optimal values for μ and σ:

$$\hat{\mu} \;=\; \frac{(\bar{\mathbf{r}}'\mathbf{C}^{-1}\bar{\mathbf{y}})}{(\bar{\mathbf{r}}'\mathbf{C}^{-1}\bar{\mathbf{r}})} \tag{13}$$

and

$$\hat{\sigma}^2 \;=\; \frac{(\bar{\mathbf{y}} - \bar{\mathbf{r}}\hat{\mu})'\mathbf{C}^{-1}(\bar{\mathbf{y}} - \bar{\mathbf{r}}\hat{\mu})}{n} \tag{14}$$

Substituting equations (13) and (14) back into (12), we find the concentrated form of the conditional log-likelihood function, depending on parameters θ, p and x^* via matrix $\mathbf{C}$ and vectors $\mathbf{r}$ and $\bar{\mathbf{y}}$:

$$ConCLL(\theta, p, x^*) \;=\; -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{C}|) \tag{15}$$

Maximizing (15) yields the estimates needed. Then use equations (13) and (14) to calculate the estimates of μ and σ^2.

To illustrate the nasty nature of (15), consider the equivalent formulation of the full Conditional Maximum Likelihood (CML) problem. Variable S and matrix $\mathbf{C}$ are both dependent on all other variables in a complicated manner. Matrix $\mathbf{R}$ and vector $\mathbf{r}$ depends on variables x^*, θ and p, which also affects vectors $\bar{\mathbf{r}}$ and $\bar{\mathbf{y}}$.

$$\min_{\theta,p,x^*} \quad n \cdot \log(S) + \log(|\mathbf{C}|) \qquad \textbf{[CML]}$$

$$
\begin{aligned}
s.t. \qquad S &= \tfrac{1}{n} \cdot \left[(\bar{\mathbf{y}} - \bar{\mathbf{r}} \cdot \mu)' \cdot \mathbf{C}^{-1} \cdot (\bar{\mathbf{y}} - \bar{\mathbf{r}} \cdot \mu)\right] \\
\mu &= \frac{(\bar{\mathbf{r}}' \cdot \mathbf{C}^{-1} \cdot \bar{\mathbf{y}})}{(\bar{\mathbf{r}}' \cdot \mathbf{C}^{-1} \cdot \bar{\mathbf{r}})} \\
\mathbf{C} &= \mathbf{R} - \mathbf{r} \cdot \mathbf{r}' \\
\bar{\mathbf{y}} &= \mathbf{y} - \mathbf{r} \cdot f^* \\
\bar{\mathbf{r}} &= 1 - \mathbf{r} \\
\mathbf{R}(i,j) &= \exp\left(-\sum_{k=1}^{d} \theta_k \cdot \left|x_k^{(i)} - x_k^{(j)}\right|^{p_k}\right) \qquad i,j = 1,\ldots,n \\
\mathbf{r}(i) &= \exp\left(-\sum_{k=1}^{d} \theta_k \cdot \left|x_k^{(i)} - x_k^{*}\right|^{p_k}\right) \qquad i = 1,\ldots,n \\
0 &< \theta_k \qquad\qquad\qquad\qquad k = 1,\ldots,d \\
1 \leq p_k &< 2 \qquad\qquad\qquad\qquad k = 1,\ldots,d \\
x_L \leq x^* &\leq x_U \\
b_L \leq Ax^* &\leq b_U \\
c_L \leq c(x^*) &\leq c_U
\end{aligned}
$$

Parameters :

f^*	given target value
d	dimension of problem
n	number of sampled points
$\mathbf{x}$	sampled points
$\mathbf{y}$	evaluated function values

3.3 Overview

Based on the DACE framework, we have presented two different approaches on how to perform iterations in order to find new sample points x^*, hopefully converging towards the global optimum. Each method is connected to good properties as well as some troublesome issues.

Two-Stage EGO

Stage 1. Find θ and p by MLE, interpolate surface.

Stage 2. Optimize some utility function to find x^*, a new point to sample.

\+ Two separate subproblems, easier to solve.

\- Relies on estimated parameters.

One-Stage EGO

Stage 1. Given a value for f^*, find θ, p and x^* by MLE, conditionally that the surface goes through (x^*, f^*).

\+ Only one subproblem to solve. The computation of x* is not dependent on previous biased estimates, more accurate computation.

\- We don't know the value of f^*. The new CML problem is more difficult.

In the upcoming Sections 4 and 5, we look deeper into the problems connected to a one-stage process and present methods to handle these issues.

4 Difficulties and Algorithm description

Although the one-stage process is theoretically more appealing, there are some practical issues that needs to be adressed. To start with, the optimal value f^* is obviously not known in advance and must be chosen in some way. This is the least of our troubles though, and methods to deal with this is presented in Section 4.1.

Section 5 is devoted to ideas on how to solve the challenging CML problem in a robust and efficient manner. This is essential since the subproblem needs to be solved multiple times each iteration.

There is also some numerical issues connected to the subproblem. When optimizing the concentrated conditional log-likelihood function (15) one need to evaluate points close to already sampled points $\mathbf{x}$, and this causes the function to collapse. Details and remedies are presented in Section 4.2.

Finally, a pseudo-code for the one-stage EGO algorithm is presented in Section 4.3.

4.1 Finding f^* values

The one-stage approach finds a new point x^* by computing the likelihood of the observed data conditional upon the assumption that the surface goes through the point (x^*, f^*). The value of f^* is not known in advance and must be chosen somehow.

Fortunately, the use of a target value is not unique for the one-stage EGO algorithm. When Gutmann introduced the radial basis function (RBF) method [3], he proposed a cycle of 5 target values lower than the minimum of the interpolated surface, ranging from far below (global search) to close below (local search). Set the value of f^* to

$$f_k^* \;=\; s_{min} - w_k \cdot \Big(\max_i f(x_i) - s_{min}\Big)$$

where s_{min} is the minimum of the interpolated surface. The weight factor w_k is defined using a cyclic scheme over k like

$$w_k \;=\; (1 - k/N^2), \qquad\qquad k = 0, 1, \ldots, N-1.$$

This is successfully done in TOMLABs CGO-solver `rbfSolve` [1].

A more powerful approach, but also more computer intensive, is to solve the subproblem for a wide range of f^* values every iteration. Each solution gives an x^* candidate, so how to proceed? It turns out that these solutions tend to cluster, and by applying a clustering process one could proceed with 1-3 new x^* values each iteration. This is successfully done in TOMLABs CGO-solver `ARBFMIP`, and details on the target values and the clustering process are found in [4] and [5].

Both methods overcome the problem of not knowing the optimal target value f^* in advance, but experience from solving a large number of test problems with `rbfSolve` and `ARBFMIP` suggests that using a range of target values adds robustness to the optimization process.

Another advantage of getting a cluster of new points is the possibility to benefit from parallel calculations, becoming more and more standard for computers today. Hence our implementation of the one-stage EGO algorithm will adopt the clustering methods of `ARBFMIP`.

4.2 Numerical Issues

The EGO algorithm is known to suffer from numerical problems due to the correlation matrix becoming more and more ill-conditioned as sampled points start to cluster in promising areas of the sample space. Equation (10) used for the MLE of parameters θ and p includes the logarithm of the determinant of the correlation matrix $\mathbf{R}$, which approaches zero as points are sampled close together.

For the one-stage process, the CML function (15) is optimized, and the numerical issues have not disappeared. The situation is even worse due to the intricate relations between sampled points and parameters θ and p, now also affected by parameters x^* and f^*. The problem with a rank deficient $\mathbf{C}$ matrix is that when evaluating (15) its inverse is needed (although not calculated implicitly, it is still problematic). It also contains the term $\log(|\mathbf{C}|)$, which is undefined as the determinant becomes 0. We now present three situations where numerical issues arise in the one-stage approach.

Solving the MLE

Numerical issues arise when maximizing (10), the standard MLE, to find parameters θ and p. The upper bound for parameter p is theoretically 2, but Sasena reports in [11] that a value strict less than two is more numerical stable. Here is an example clearly supporting the claim.

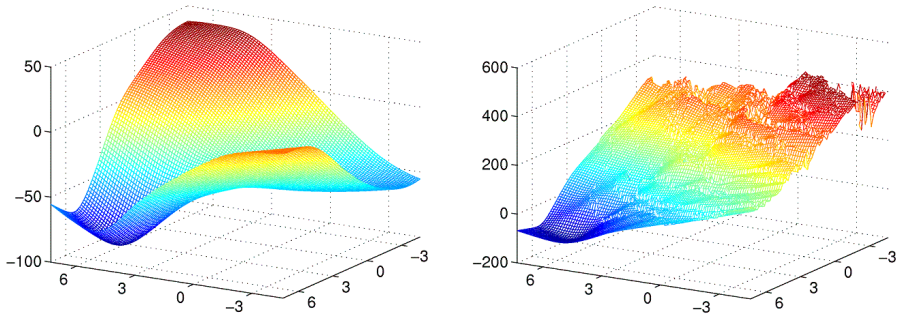

Figure 1: MLE of θ and p. To the left, $p = 1.99$ and to the right $p = 2$.

Figure 1 shows the same ML function to be minimized, but with the parameter p fixed to 1.99 and 2 respectively. Clearly the function where $p = 1.99$ is preferable, motivating the upper bound of p to be adjusted to 1.99.

Evaluating CML close to sampled points

Ill-conditioning of the correlation matrix is inherited from the two-stage approach, but also enhanced due to the definition of correlation matrix $\mathbf{C} = \mathbf{R} - \mathbf{r}\mathbf{r}'$. The correlation vector $\mathbf{r}(i)$ approaches 1 for $x^{(i)}$s close to x^*, hence creating a close-to-zero row and column in $\mathbf{C}$ whenever evaluating points close to $\mathbf{x}$, the set of so far sampled points.

In evaluating a point really close to $\mathbf{x}$, the logical thing would be for the CML function (15) to approach minus infinity. It is certainly unlikelihood for an already sampled point, with a function value not equal to f^*, to suddenly match f^*. But when zooming in on the CML function in a very small interval around a sampled point, we get something else. Figure 2 illustrates the phenomenon, showing pictures of (15) close to a sampled point. The CML functions are displayed as minimization problems.

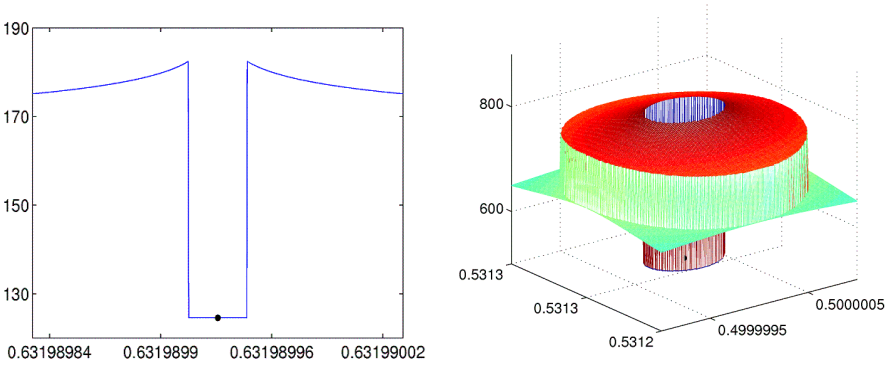

Figure 2: Evaluating CML close to an already sampled point. The function should increase continuously, but suddenly drop to a constant value.

So what happens when matrix $\mathbf{C}$ becomes very close to singular, due to reasons explained before, and we try to evaluate (15). In our implementation, first the QR-factorization of matrix $\mathbf{C}$ is found, then its determinant is calculated as the product of the diagonal elements of R which are greater than a specified lower bound ϵ, i.e.

$$\det \mathbf{C} = \prod_{i \in I} R(i,i) \quad \text{where} \quad I = \{i : |R(i,i)/R(1,1)| > \epsilon\}.$$

This pseudo-rank determinant causes the "valley" surrounding the sampled point, clearly seen in the one-dimensional example. The width of the valley depends on the value of ϵ, becoming more narrow as ϵ goes to zero. So by using a smaller value of ϵ than normally necessary, we at least decrease the critical region. To compensate for the pseudo-rank, we subtract a big number to the objective for all points inside the valley.

It is important to clarify the need of evaluating points close to already sampled points. If a function value $\mathbf{y}(i)$ corresponding to a sampled point $\mathbf{x}^{(i)}$ is much larger than f_{min}, the best found function value so far, it is not desirable to sample points close to $\mathbf{x}^{(i)}$. On the other hand, in search for a new point x^* that improves the objective value, it is natural to evaluate points close to sampled points with low function values. In order to find solutions with many digits of accuracy, it is even necessary.

Bad combinations

After some iterations, when having sampled some points, another numerical issue arises when maximizing both (10) and (15). In contrast to the previous problem, this is due to a very natural cause. Independent of the optimization method used to solve the MLE, many combinations of parameters θ and p are tested, and some of them are simply not feasible for the given set of sampled points $\mathbf{x}$.

Although the reasons are completely different, the result is the same as before. When matrix $\mathbf{C}$ becomes singular and its determinant is zero, the CML function returns minus infinity when taking the logarithm, causing the optimization process to halt. A numerical example is presented to the left in Figure 3.

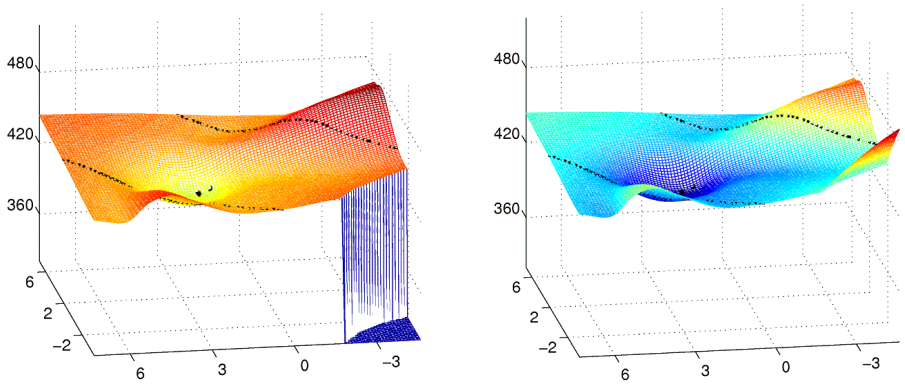

Figure 3: MLE of θ for a fixed value of p. To the left, the ML function without safeguard. To the right, the same ML function with safeguarded logarithm calculations.

To handle this situation, we safeguard the calculations of the logarithm. Whenever the determinant is zero, return a large negative value instead of calculating the logarithm. By investigation, MATLAB evaluates $\log(10^{-323})$ as -743.7469 but a smaller value like $\log(10^{-324}) = -\infty$, so whenever $|\mathbf{C}| < 10^{-323}$ we replace the term $\log(|\mathbf{C}|)$ by -743.75.

As seen in Figure 3, this removes the discontinuity caused by the determinant calculations breaking down. The CML function will act linear in the infeasible area. This might not be entirely satisfactory, but it prevents the optimization process from breaking down which is the important thing.

4.3 Pseudo Code

When implementing the one-stage EGO, the algorithmic structures are coded in MATLAB but all heavy calculations are implemented in Fortran and C code, and interfaced using so called mex file interfaces. Here follows a description of the one-stage EGO algorithm together with a pseudo-code presented in Algorithm 1.

After finding the initial set of n sample points, we estimate parameters θ and p by optimizing (10) and build an interpolation surface. We do this in order to find the surface minimum s_{min}, used to define the range of target values F. Then begins the process of solving the Conditional Maximum Likelihood (CML) for each target value $f_k^* \in F$, cluster the result and pick new points to sample. One could optionally add s_{min} as well, since the idea of the interpolation surface is to approximate the true costly function. This is repeated until the global optimum is found, or the maximum number of function evaluations is reached.

Our MATLAB implementation is named `osEGO`.

Algorithm 1 The one-stage EGO algorithm

```
Find n ≥ d + 1 initial sample points using some Experimental Design.
while n < MAXFUNC do
    Estimate parameters θ and p by optimizing the ML-function (10).
    Use θ and p to build the interpolation surface s(x).
    Find the minimum of the surface, denote it s_min.
    Define F, a range of f*-values.
    for all f*_k ∈ F do
        Maximize the Conditional ML (15) defined by f*_k.
    end for
    Apply a clustering process on the results, add new points and update.
    Optionally add s_min and update.
end while
```

5 The CML problem

Instead of solving two consecutive subproblems, the one-stage EGO combines all work into a single subproblem. This eliminates the use of estimated values for parameters θ and p, the main drawback of the original two-stage EGO algorithm. But incorporating x^* into the MLE complicate things. Not only is the dimension of the subproblem increased by $d+1$, we need to guess the value of f^*.

In order to solve this new, more complicated, conditional MLE problem efficiently for a whole range of f^* values, we need to find ways of generating good initial values for our parameters x^*, θ and p, and then solve the full problem. We do this by solving a series of restricted subproblems.

One might argue that in this way we still solve more than one subproblem, hence not gaining anything compared to the two-stage EGO algorithm. But the point is that while standard EGO first approximates the interpolation surface to already sampled points and then search for x^* in a separate problem, we do not separate x^* from the MLE of θ and p although solving the CML problem in several steps.

Subproblem 1

To get started, we solve a restricted version of CML where p_k is fixed to 1.99 for all k, and both x^* and θ are considered univariate, i.e. $\theta = \theta_1 = \theta_2 = \ldots = \theta_d$ and $x^* = x_1^* = x_2^* = \ldots = x_d^*$. The most difficult variables are x^*, so by scanning the space using a parametrization we find a suitable initial value of x^*. Notice that Subproblem 1 will always be a two-dimensional problem independent of the original problem. The peaks seen in Figure 4 on page 14 correspond to $\mathbf{x}$ lying close to the diagonal. To solve the subproblem, use starting points in between the peaks.

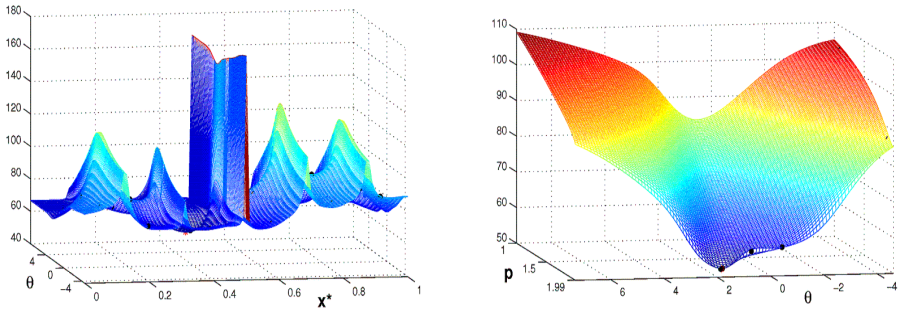

Figure 4: To the left, subproblem 1 with univariate x^* and θ for $p = 1.99$ fixed. To the right, subproblem 2 with univariate θ and p, using x^* found in subproblem 1.

Subproblem 2

Having found an initial guess x_0^*, we progress by keeping x^* fixed and consider univariate values for θ and p instead, solving another two-dimensional problem. From Subproblem 1 we get a good initial guess for θ, which makes Subproblem 2 relatively easy to solve. In Figure 4, to the right, a numerical example.

Subproblem 3

At this point, we have univariate initial values for all parameters. From solving Subproblem 1 we get values for x_0^* and θ_0, and the latter is refined in Subproblem 2 together with p_0. We are now ready to solve the full subproblem, with no parameterizations or fixations. The solution of CML, which is defined by the current target value f_k^*, is a suggestion for a new point to sample, denoted x_k^*. Figure 5 and Figure 6 show pictures of the CML problem, displayed as a maximization problem for best possible visualization.

Resolving for a new f_k^*.

Since we need to solve CML for $f_k^* \in F$, where F denotes a range of target values for the current iteration, one should exploit the possibility of an efficient reoptimization process. We notice that a relative small change in the value of f_k^* results in a very similar problem to solve. Therefore, if $f_k^* - f_{k+1}^*$ is small, the solution x_k^* serves as a good initial guess for x_{k+1}^*. In this way, by keeping track of when consecutive values of f_k^* differ a lot, it suffices to solve all subproblems only a few times. Whenever the change in f^* value is relatively small, solve the full problem directly using the previous solution as starting point. This will speed up the solution process significantly.

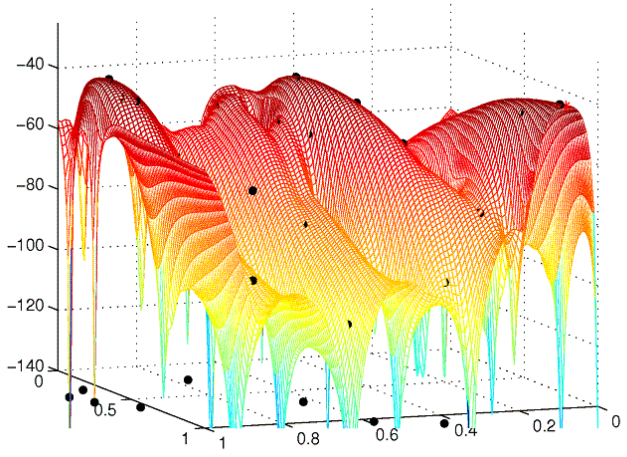

Figure 5: The full CML subproblem for fixed values of parameters θ and p. At this stage, the number of sampled points n is relatively small. Black dots indicate starting points for the solver used to maximize CML.

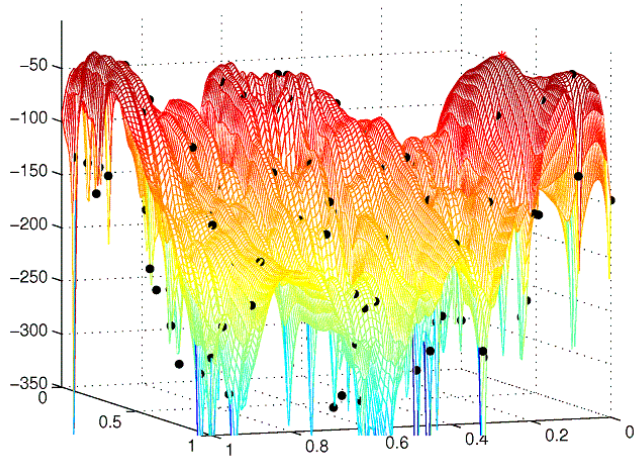

Figure 6: The full CML subproblem for fixed values of parameters θ and p. The number of sampled points n is much larger than before, and the problem has become much more difficult. Black dots indicate starting points used by the solver.

Modifications

It is possible to solve a restricted version of CML and accept the solution as x_k^* without solving the full subproblem. The most straightforward way is to consider univariate values for θ and p throughout the whole algorithm, hence decreasing the problem dimension from $3 \cdot d$ to $d + 2$. This will most likely be necessary when attempting to solve higher dimensional problems.

Experience over the years using the EGO algorithm also suggests that keeping p fixed to a single value at all times, eliminating this parameter completely from the optimization process, does not affect the results significantly. A value close to 2 is suggested, and from the earlier discussion in Section 4.2 on numerical issues, an upper bound of $p = 1.99$ is strongly recommended.

6 Benchmark and Tests

This paper aims at evaluating the one-stage EGO implementation `osEGO` by solving a set of test problems and compare the results with other algorithms. Three solvers from the TOMLAB/CGO environment are used. The `rbfSolve` and `arbfmip` solvers utilize radial basis functions, and the `EGO` solver is an implementation of the standard two-stage EGO algorithm.

All solvers run with their default parameter settings, controlling algorithmic options like variable scaling and the choice of merit function.

Following the definition of Dolan and Moré [2], a benchmark consists of a set P of benchmark problems, a set S of optimization solvers and a well defined convergence test. Since the problems are considered costly, we define a performance measure $t_{s,p} > 0$, the number of function evaluations required for problem $p \in P$ to converge using solver $s \in S$.

All solvers are set to break after 200 function evaluations or earlier if convergence to the known global optimum is obtained. The relative error is defined as

$$E_r = \frac{f_{min} - f_{opt}}{|f_{opt}|},$$

where f_{min} is the currently best feasible function value and f_{opt} is the known global optimum. Convergence is assumed if the relative error E_r is less than 10^{-4}. When $f_{opt} = 0$, stop when f_{min} is less than 10^{-4}. If convergence is not reached after 200 iterations, declare failure.

All CGO solvers need an initial set of points, or experimental design, in order to start the algorithm. Since the behavior of any such algorithm often depends heavily on this set, we solve each problem for a set of experimental designs E, summarized in Table 1 on page 17.

We consider two kinds of test problems in this benchmark, unconstrained box-bounded problems P_U and constrained problems P_C. A thorough presentation of the problems is found in Section 6.2, summarized in Table 2 on page 18.

6.1 Experimental Designs

Here follows a short description of the five different experimental designs (ExD) used in this benchmark. All but one design are defined by the number of initial points to sample, denoted by N. Some of them are able to handle constraints, while others can do this optionally.

The Corner Point Strategy (CPS) generates a fixed number of initial sample points, one for each corner point of the bounding box. This method is not able to handle constraints. The Deterministic Global Solver (DGS) approach applies the DIRECT algorithm to find N initial points, and is able to handle constraints.

The Maximin LHD strategy apply a Latin Hypercube Design of any given size N, where the points are separated subject to the maximin distance. This method could optionally handle constraints, using a method by Quttineh presented in [9]. The LHD designs used are taken from the webpage `http://www.spacefillingdesigns.nl` where a large collection of optimal maximin designs are available.

Finally, we also consider a combination of the CPS strategy with the Maximin LHD and DGS methods respectively. The $N = 2^d$ corner points of the bounding box are added to the designs generated by the Maximin LHD and DGS methods.

Table 1: The set of Experimental Designs (E). Five different ExD methods are listed, together with the available options. The Maximin LHD method can handle constraints optionally, hence $E_C = 4$ combinations for the constrained problems.

ExD	Experimental Design	Size of N	Constraints	E_U	E_C
CPS	Corner Points	Fixed	No	1	1
DGS	DG Solver	N_1 and N_2	Yes	2	2
LHD	Maximin LHD	N_1 and N_2	Yes/No	2	4
CP+DGS	Corners + DGS	N_1 and N_2	Yes	2	2
CP+LHD	Corners + LHD	N_1 and N_2	Yes/No	2	4

For all designs, except the CPS strategy, we use the settings $N_1 = (d+1)\cdot(d+2)/2$ and $N_2 = 10\cdot d + 1$, where d is the dimension of the problem to be solved. For the Maximin LHD design, it is possible to choose whether constraints should be considered or not. Therefore, each constrained problem will be solved in total 4 times using the Maximin LHD design, with and without constraints taken into account and for the options N_1 and N_2.

To summarize, we solve each box-bounded problem using nine different designs, nine being the sum of column E_U in Table 1. The constrained problems are solved using 13 different designs, the sum of column E_C. For a thorough description on the different experimental designs, see [9].

6.2 Test Problems

We define a set of 13 box-bounded unconstrained problems P_U, each solved for E_U different designs, and a set of 6 constrained problems P_C, each solved for E_C different designs. Most of them are 2-dimensional, except a few problems in 3 dimensions.

Table 2 and Table 3 give a compact description of the test problems. Column d is the number of variables, Ax the number of linear inequality constraints and $c(x)$ the number of nonlinear inequality constraints. In the *Domain* column, the lower and upper bounds for all variables are shown. The *Range* column shows the order of the objective.

None of the test problems have a global minimum in a corner point or midpoint.

Table 2: The set of box-bounded test problems P_U.

Nr.	Problem Name	d	Domain	Range
B4	Hartman 3	3	$[0,0,0] - [1,1,1]$	$4{\cdot}10^{0}$
B6	Branin RCOS	2	$[-5,0] - [10,15]$	$3{\cdot}10^{2}$
B7	Goldstein and Price	2	$[-2,-2] - [2,2]$	$1{\cdot}10^{6}$
B8	Six-Hump Camel	2	$[-3,-2] - [3,2]$	$2{\cdot}10^{2}$
B16	Shekels foxholes 2	2	$[0,0] - [10,10]$	$1{\cdot}10^{1}$
B19	Michalewiczs function 2	2	$[0,0] - [\pi,\pi]$	$2{\cdot}10^{0}$
B30	Myers smoothly fluctuating	2	$[-0.5,-0.5] - [3.5,3.5]$	$6{\cdot}10^{0}$
B32	LOG-Goldstein and Price	2	$[-2,-2] - [2,2]$	$1{\cdot}10^{1}$
B53	Dixon and Price	2	$[-10,-10] - [10,10]$	$9{\cdot}10^{5}$
L20	M20	2	$[0,0] - [5,5]$	$3{\cdot}10^{0}$
L25	M25	2	$[-2,-2] - [2,2]$	$1{\cdot}10^{0}$
L28	M28	2	$[3,3] - [9.99,9.99]$	$1{\cdot}10^{2}$
L49	M39	2	$[-500,-500] - [500,500]$	$2{\cdot}10^{3}$

Table 3: The set of constrained test problems P_C.

Nr.	Problem Name	d	Ax	$c(x)$	Domain	Range
C2	Gomez 3	2	0	1	$[-1,-1] - [1,1]$	$4{\cdot}10^{0}$
C3	Hock-Schittkowski 59	2	0	3	$[0,0] - [75,65]$	$1{\cdot}10^{2}$
C4	Hock-Schittkowski 65	3	0	1	$[-4.5,-4.5,-5] - [4.5,4.5,5]$	$2{\cdot}10^{2}$
C13	Schittkowski 343	3	0	2	$[0,0,0] - [36,5,125]$	$2{\cdot}10^{2}$
C19	Bump 2	2	1	1	$[\epsilon,\epsilon] - [10,10]$	$1{\cdot}10^{0}$
C22	HGO 468:1 + constraint	2	0	1	$[0,0] - [1,1]$	$6{\cdot}10^{0}$

6.3 Numerical Results

To present the benchmark results in a standardized manner, we utilize data profiles suggested by Moré and Wild [8], a kind of probability density function. Since function evaluations are expensive we are interested in the percentage of problems solved to a given accuracy within k function evaluations. Data profiles are designed to handle this, and are defined for a set of test problems P and a solver $s \in S$ by

$$d_s(k) = \frac{1}{|P|} \cdot \left| \left\{ p \in P : \frac{t_{p,s}}{n_p + 1} \leq k \right\} \right|$$

where n_p is the number of variables for each problem $p \in P$.

Our benchmark is defined by the set S of CGO solvers `rbfSolve`, `arbfmip`, `ego`, and `osEgo`, and the set of test problems defined by $\bar{P} = \{P \times E\}$. Since the choice of initial set of sample points have such a big impact on the performance of CGO solvers, each combination of test problem $p \in P$ and experimental design $e \in E$ is considered as a unique problem. In this way, the data profiles will show if a solver is robust or not with respect to the initial set of sample points.

Unconstrained problems

Figure 7 presents data profiles for the set of unconstrained box-bounded problems $\bar{P}_U = \{P_U \times E_U\}$. It consists of all combinations of the 13 box-bounded problems and 9 experimental designs, i.e. $|\bar{P}_U| = 117$. The tolerance of the relative error E_r is set to 1% in the left picture and to 0.01% in the right picture.

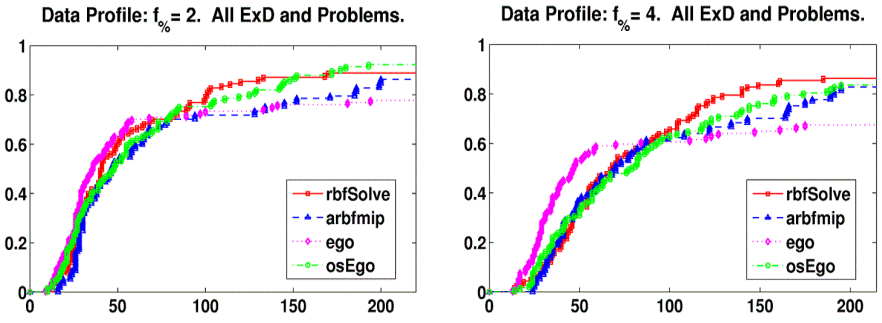

Figure 7: Data profiles for the box-bounded unconstrained problems $\bar{P}_U$ show the percentage of problems solved as a function of k, the number of function evaluations needed to converge, with tolerance 1% and 0.01%.

The `osEgo` algorithm is doing quite well, solving more problems to 2 digits of accuracy than any other solver in S. In finding 4 digits of accuracy, it is beaten by the `rbfSolve`.

Constrained problems

Figure 8 presents data profiles for $\bar{P}_C = \{P_C \times E_C\}$, the set of constrained box-bounded problems. It consists of all combinations of the 6 constrained problems and the 13 experimental designs, i.e. $|\bar{P}_C| = 78$. The tolerance is set to 1% in the left picture and to 0.01% in the right picture. All solvers perform very well on this set of constrained problems, finding the global optimum for almost all instances.

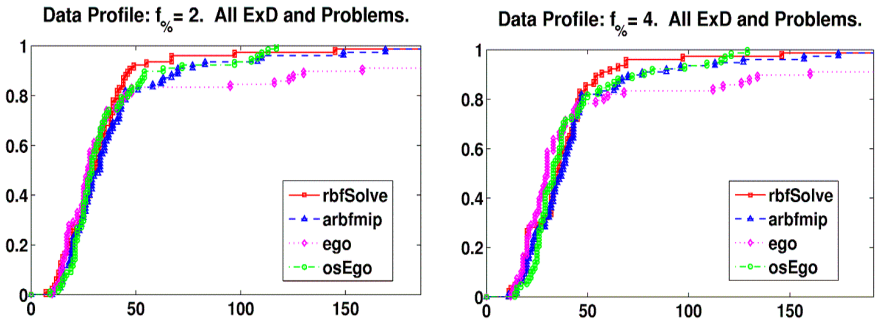

Figure 8: Data profiles for the constrained box-bounded problems $\bar{P}_C$ show the percentage of problems solved as a function of k, the number of function evaluations needed to converge, with tolerance 1% and 0.01%.

The `osEgo` algorithm is the most robust solver in S, converging to 4 digits of accuracy for all problems in less than 140 function evaluations. The `rbfSolve` and `arbfmip` algorithms are both very close to solve all problem instances as well, only the `ego` algorithm has significant failures.

Problem specific analysis

Table 4 and Table 5 present the results separately for each solver in S. The number of failures (in %) and the mean, min and max for the successful runs out of the total runs for each problem are reported. That is, the mean, min and max number of function evaluations needed to converge using the different experimental designs in E for each solver and problem.

The `osEGO` algorithm performs well, already confirmed by the data profiles. Some specific problems seem to cause a lot of trouble though. A common feature for problems B7 and B53 is the wide range of the objective. It seems like `osEGO`, just like the other solvers, is sensitive to such a large span in function values.

Another demanding problem is B16, the classical Shekel Foxholes. All solvers but `EGO` finds the global optimum for all experimental designs, in less than 200 function evaluations, which is impressive. Problem L49 is also challenging, filled with many local minima, even so both `rbfSolve` and `osEGO` are successful.

Table 4: Number of function evaluations to get within 1% of the optimal value.

Solver	%	**rbfSolve**			%	**arbfmip**			%	**ego**			%	**osEgo**		
	Fail	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max
B4	*0*	43	18	58	*23*	51	38	60	*0*	41	31	53	*0*	38	17	49
B6	*0*	32	24	42	*0*	39	30	47	*0*	29	19	35	*0*	40	29	49
B7	*78*	169	168	170	*34*	166	145	200	*23*	69	26	194	*56*	162	151	172
B8	*0*	36	21	46	*0*	45	28	57	*23*	30	21	41	*0*	58	35	70
B16	*0*	99	38	130	*0*	140	37	186	*100*	-	-	-	*0*	113	55	148
B19	*0*	39	26	64	*0*	29	19	43	*12*	24	12	37	*0*	26	13	40
B30	*0*	20	10	32	*0*	26	23	30	*0*	24	13	33	*0*	22	13	30
B32	*0*	42	29	53	*0*	53	31	65	*56*	52	48	55	*0*	64	31	79
B53	*0*	79	22	133	*0*	119	68	199	*23*	40	25	56	*0*	95	32	193
L20	*0*	24	13	30	*0*	23	16	28	*0*	21	13	30	*0*	25	15	35
L25	*34*	36	24	68	*45*	32	30	38	*12*	61	9	181	*45*	18	10	26
L28	*34*	25	22	27	*0*	42	26	77	*0*	21	11	26	*0*	40	10	103
L49	*0*	86	61	103	*78*	71	68	74	*45*	112	89	141	*0*	106	21	180
C2	*0*	22	10	32	*0*	23	10	33	*0*	20	10	28	*0*	23	9	35
C3	*0*	22	12	35	*0*	25	15	36	*8*	21	11	28	*0*	24	14	31
C4	*0*	34	20	46	*0*	32	19	45	*0*	31	17	48	*0*	38	20	54
C13	*0*	28	11	45	*0*	28	11	45	*24*	31	13	45	*0*	33	15	47
C19	*8*	50	7	145	*0*	68	15	169	*24*	89	30	158	*0*	74	21	117
C22	*0*	42	32	67	*8*	59	34	149	*0*	26	17	36	*0*	29	19	53

Table 5: Number of function evaluations to get within 0.01% of the optimal value.

Solver	%	**rbfSolve**			%	**arbfmip**			%	**ego**			%	**osEgo**		
	Fail	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max	*Fail*	mean	min	max
B4	*0*	103	54	160	*23*	82	58	122	*0*	51	45	59	*0*	50	24	63
B6	*0*	44	30	61	0	*55*	48	62	*0*	41	29	55	*0*	62	50	79
B7	*89*	185	185	185	*56*	183	175	191	*56*	73	28	127	*89*	187	187	187
B8	*0*	53	35	71	*0*	57	38	73	*23*	34	25	48	*0*	89	76	110
B16	*0*	113	48	140	*0*	147	41	195	*100*	-	-	-	*0*	125	64	165
B19	*0*	54	40	70	*0*	35	24	54	*12*	27	15	39	*0*	33	24	47
B30	*0*	22	14	34	*0*	34	25	48	*0*	29	16	38	*0*	26	14	37
B32	*0*	64	50	81	*0*	60	39	77	*89*	52	52	52	*0*	87	38	137
B53	*12*	123	71	161	*23*	121	87	165	*67*	104	40	161	*45*	160	118	195
L20	*0*	34	23	45	*0*	36	30	42	*0*	25	16	35	*0*	39	27	50
L25	*34*	58	44	110	*45*	37	32	49	*34*	32	13	70	*45*	32	17	41
L28	*45*	64	39	91	*0*	136	79	166	*0*	26	16	35	*34*	92	25	144
L49	*0*	100	81	121	*78*	74	70	78	*45*	147	120	175	*0*	118	28	191
C2	*0*	27	18	36	*0*	26	15	37	*0*	22	12	30	*0*	27	14	39
C3	*0*	25	12	36	*0*	31	19	44	*8*	22	15	29	*0*	27	17	33
C4	*0*	44	20	61	*0*	37	23	47	*0*	37	21	60	*0*	50	29	82
C13	*0*	28	11	45	*0*	28	11	45	*24*	31	13	45	*0*	33	15	47
C19	*8*	56	11	146	*0*	72	16	174	*24*	96	33	160	*0*	78	25	129
C22	*0*	45	32	69	*8*	65	39	157	*0*	28	20	39	*0*	35	26	60

Experimental Designs and the one-stage EGO

Some interesting details were found when analyzing the performance of the `osEgo` algorithm with respect to the different experimental designs in E. For the set of unconstrained problems P_U, the CP+DGS experimental design works best with the option $N_2 = 10 \cdot d + 1$.

For the constrained problem set P_C, the Maximin LHD allowing infeasible points and using the option $N_1 = (d+1)(d+2)/2$ was the most successful design. The results also indicate, somewhat ambiguously, that one should rather use a constrained LHD when in combination with the Corner Point Strategy. It should be noted that the set P_C is quite small, hence it is difficult to draw any conclusions.

7 Conclusions

The one-stage EGO approach is promising, although it still needs to be improved. We have successfully implemented an adaptive scheme for f^*, similar to the one used in `ARBFMIP`. From the test results, though, it seems like a good idea to implement and test the cyclic choice of f^* as well.

By breaking down the full CML problem into subproblems, using univariate variables for θ, p and x^*, we are able to find good starting points and find new candidates x^* in each iteration. Using a clustering process, we find the best candidates from each cluster and proceed with multiple points each iteration.

Numerical issues are a big problem, and it might not be possible to resolve all of them. We have presented good fixes for some of the issues, allowing the `osEgo` implementation to compete with the other CGO solvers in TOMLAB, outperforming the old `ego` implementation.

The nasty subproblem CML increase with the number of sampled points n, since the size of correlation matrices $\mathbf{R}$ and $\mathbf{C}$ are $n \times n$. This causes the calculations to get heavier as the iterations go by, and more parameter combinations become infeasible, complicating the optimization of the subproblems.

At the moment, the solver used to maximize the full CML subproblem utilize a set of starting points $\mathbf{x_0}$. These points are chosen as the sampled points $\mathbf{x}$, but slightly perturbed towards the midpoint of the sample space. This is motivated by inspection of the subproblem, realizing that the optimal solution is often very close to an already sampled point.

Due to our discussion in Section 4.2, we choose to perturb the starting points, avoiding numerical issues and not starting in a deep basin surrounding sampled points. By perturbing towards the midpoint, we keep feasibility with respect to the box-bounds.

7.1 Future work

We need to speed up the subproblem solving phase. As n increases, so does the number of starting points $\mathbf{x_0}$. As points tend to pile up in promising areas, many of

the points in $\mathbf{x_0}$ are very similar (close in Euclidean meaning), a bad feature for a set of starting points. A possible remedy, partly implemented already, is to cluster the sampled points $\mathbf{x}$ and thus reduce the size of $\mathbf{x_0}$ and hence decrease the solution times.

The numerical results indicate that `osEGO` is sensitive to large spans in function values. We should consider strategies of transforming the function values, perhaps consider the log values, in order to reduce the range.

References

[1] M. Björkman and K. Holmström: Global Optimization of costly nonconvex functions using radial basis functions. *Optimization and Engineering* **1** (4), 373–397 (2000).

[2] E. D. Dolan, J. J. Moré, and T. S. Munson: Optimality Measures for Performance Profiles. *Preprint ANL/MCS-P1155-0504* (2004).

[3] H.-M. Gutmann: A radial basis function method for global optimization. *Journal of Global Optimization* **19** (3), 201–227 (2001).

[4] K. Holmström: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. *Journal of Global Optimization* **41**, 447–464 (2008).

[5] K. Holmström, N.-H. Quttineh, and M. M. Edvall: An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. *Optimization and Engineering* **41**, 447–464 (2008).

[6] D. R. Jones: A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization* **21**, 345–383 (2002).

[7] D. R. Jones, M. Schonlau, and W. J. Welch: Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* **13**, 455–492 (1998).

[8] J. J. Moré and S. M. Wild: Benchmarking Derivative-Free Optimization Algorithms. *Preprint ANL/MCS-P1471-1207* (2007).

[9] N.-H. Quttineh and K. Holmström: The influence of Experimental Designs on the Performance of Surrogate Model Based Costly Global Optimization Solvers. *Studies in Informatics and Control* **18** (1), (2009).

[10] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn: Design and analysis of computer experiments (with discussion). *Statistical Science*, **4**, 409–435 (1989).

[11] M. J. Sasena: Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations. *Doctoral Dissertation* (2002).