A Derivative Free Optimization Algorithm in Practice

by A.R. Conn[1], K. Scheinberg[1] and Ph.L. Toint[2]

Report 98/11 July 7, 1998

[1] IBM T.J. Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598, USA
Email: arconn@watson.ibm.com

[2] Department of Mathematics, Facultés Universitaires ND de la Paix, 61, rue de Bruxelles, B-5000 Namur, Belgium, EU
Email: pht@math.fundp.ac.be
Current reports available by anonymous ftp from the directory "pub/reports" on thales.math.fundp.ac.be
WWW: http://www.fundp.ac.be/sciences/math/pht/publications.html

Accepted for the American Institute of Aeronautics and Astronautics Conference, St Louis, September 1998.

A Derivative Free Optimization Algorithm in Practice

A. R. Conn,

IBM T. J. Watson Center, Yorktown Heights, NY

K. Scheinberg

IBM T. J. Watson Center, Yorktown Heights, NY

Ph. L. Toint

Department of Mathematics, FUNDP, Namur, Belgium

July 7, 1998

Abstract

We consider an algorithm for optimizing a nonlinear function, without constraints, whose first order derivatives exist but are unavailable. The algorithm is based on approximating the objective function by a quadratic polynomial interpolation model and using this model within a trust-region framework. We study some practical properties of the algorithm and show how this algorithm can be extended to solve certain constrained optimization problems. Computational results for analytical problems and for applied problems from the aeronautic industry are presented.

General Framework

In the paper we are considering a general derivative free optimization (DFO) algorithm for minimizing a nonlinear function within some feasible set $F \subseteq \mathrm{R}^n$; i.e.,

$$\begin{aligned} \min \quad & f(x) \\ & x \in F. \end{aligned} \tag{0.1}$$

We will first consider an algorithm for solving an unconstrained problem; i.e., problem (0.1) with $F = \mathrm{R}^n$. Later we consider ways to modify the algorithm to solve problem (0.1) with certain type of constraints. We assume that $f(x)$ is twice continuously differentiable, although in the application noise is typically present. However, we also assume that the derivatives of $f(x)$ are, generally, *unavailable* and that every evaluation of $f(x)$ is *expensive* (time-wise or otherwise). Most typical optimization techniques are not applicable or not efficient in such contexts. The purpose of our general algorithm is to be able to solve problem (0.1) by using as few function evaluations as possible and using any function value as soon as it becomes available.

The applications of this framework are frequently encountered in engineering. Typically, such problems arise when one needs to minimize an objective measured by some experiment or by a complicated simulation package, which is treated as a black box. The number of variables in such problems is, usually, not large (< 100).

In this paper we discuss some particular applications from Boeing. One arises in helicopter rotor blade design (we will further refer to this problem as the "helicopter problem"). The problem has instances 11, 31 or about 50 variables, associated with the discretization of the rotor blade. The objective is to minimize the vibration of the rotor. In our experiments with a simplified simulation of the objective function evaluation, each such evaluation takes about a minute of computer time. For the 31 variable problem this means that estimating a *single* gradient by a finite difference scheme takes more than half an hour. Another example is an engine nozzle design problem. This problem has about 10 geometry parameters which need to be set so that energy losses in the engine are minimized. There are several constraints on the parameters which we discuss later. Each objective function evaluation takes a few hours on a high-end workstation.

In more detail this and other examples of application are described in [2].

The DFO algorithm was successfully used to solve problems of automatic calibration of a water distribution network by a group at the University of Colorado at Denver. These problems have no constraints and have up to 100 variables. Each function evaluation takes about 20 seconds on a high end PC.

The DFO algorithm which we consider here is described in detail in [5] and [6]. Its prototype is studied in [7] and it was inspired by [12] and [13]. A survey of most state-of-the-art techniques of optimization without derivatives and motivations for the DFO algorithm can be found in [6] (also see [14]).

In this paper we are mainly concerned with practical issues and the numerical performance of the DFO algorithm. The paper is organized as follows. In the next section we introduce the algorithm with some level of detail. We also discuss the main mechanisms used. In the section after that we discuss some properties of the DFO algorithm. Further we consider modification of the algorithm which allow us to solve constrained problems. We consider constraints of 3 kinds. "Easy" constraints are the constraints whose values and derivatives can be easily computed (with simple bounds as an almost trivial special case). Problems with such constraints can be solved by DFO with little modification. "Difficult" constraints are constraints whose derivatives are not available and whose values are at least as expensive to compute as that of the objective function. Solution of such problems require significant extension of the DFO algorithm which is out of scope of this paper. The third type of constraints, which we call "virtual" constraints, arise in the Boeing helicopter problem. We cover these constraints later in the paper. A modification of the DFO algorithm provides a heuristic solution for problems with "virtual" constraints, but an optimal solution is not guaranteed.

In the last section we present extensive computational results and compare DFO with other popular techniques, such as LANCELOT [4] using finite differences gradient approximation and PDS ([8], and [16]). We present results for a selection of nonlinear constrained and unconstrained problems from the CUTE library [1] and for the Boeing helicopter problem.

Description of general algorithm

The main ingredient of our algorithm is the approximation of the objective function by a quadratic model. The algorithm uses the properties of the model to predict the properties of the function itself. Naturally, to obtain such a model we need to sample the function. The points at which the function is sampled in our algorithm can be of two types - those which, hopefully, have a smaller function value, and those which, hopefully, provide a better approximation model. Another important ingredient of our algorithm is the trust-region framework. We assume that the quadratic model approximates the objective function well within a certain region of a given radius. (The radius can be increased or decreased, depending on whether this assumption about the model proves valid or not.) If a new point is computed to obtain a better objective function value, it is computed by minimizing the current model within the trust region. A point, which improves the model, is also computed within the trust-region, since that is where the quality of the model matters most. More formally the algorithmic framework can be described by the following steps:

DFO Algorithm

Step 0: Initializations.

Let x_s and $f(x_s)$ be given. Choose an initial interpolation set Y containing x_s. Choose an initial trust region radius $\Delta_0 > 0$ and $x_0 \in Y$. Set $k = 0$.

Step 1: Build the model.
Using the interpolation set Y, containing x_k, build a model $m_k(x)$.

Step 2: Minimize the model.
Compute the point x_k^+ such that

$$m_k(x_k^+) = \min_{x \in \mathcal{B}_k} m_k(x), \qquad (0.2)$$

where $\mathcal{B}_k = \{x : ||x - x_k||_\infty \leq \Delta\}$. If $x_k^+ = x_k$, then go to Step 4.

Step 3: Compute the function value.
Compute $f(x_k^+)$ and the ratio

$$\rho_k \stackrel{\text{def}}{=} \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}.$$

Note, ρ_k measures how much we have achieved in reducing the objective function, compared to how much we reduced the model.

Step 4: Update the interpolation.

- If we can add the new point to the interpolation set without dropping any other point from the set, then we do so.
- If the reduction of the objective function is good enough, compared to the reduction of model, then we include the point into the interpolation set, even if it requires removing another point from the set.
- If the reduction is not good (or not achieved at all) compared to the reduction of the model, then we may want to search for another point which, when added to interpolation set, will improve the approximation model.
- If some interpolation points are too far away from x_k, we take them out of the interpolation set.

Step 5: Update the trust-region radius.
If the ratio ρ_k is good - increase Δ.

If the ratio ρ_k is bad and the model is at least fully linear - decrease Δ.

If the trust region radius is too small, then Stop.

Step 6: Update the current iterate.
Increment k by one, choose x_k - the sample point with the smallest function value and go to Step 1.

End of algorithm

The critical step of the algorithm is Step 4 which handles the interpolation set. The better the interpolation set "spans" the neighborhood of the current iterate the more accurately the interpolation model approximate the true objective function. In this paper we consider interpolation by polynomials of at most second degree.

Properties of the interpolation

We now discuss the main ideas behind building an interpolation set. Given any number of points in $\mathbb{R}^n$ one can build some interpolation model which may or may not interpolate all given points. Naturally, given a set of points with known function values, we would like to interpolate as many of them as possible, since we would like to use as much of the provided information as possible. Since interpolation is at most quadratic we can at once interpolate at most $(n+1)(n+2)/2$ points. But not *any* $(n+1)(n+2)/2$ in $\mathbb{R}^n$ can be interpolated by a quadratic polynomial. For example, 6 points on a line in $\mathbb{R}^2$ with random function values associated with them can be interpolated by a quadratic polynomial with probability zero. Thus, some care should be taken when choosing an interpolation set. We, henceforth, use the following definition

Definition 1 *A set of points for which there exists a quadratic or linear polynomial interpolation for any set of function values is called "poised".*

Our purpose on every iteration of the algorithm is to select the largest possible poised interpolation set from the set of points with known function values (we will refer to such points as the "sample points").

A set of points which is nearly "not-poised" is also not desirable. Not only can it cause serious numerical problems but also it produces less accurate approximations of the true function. Hence, we need to choose "well-poised" sets. For detailed analysis of poisedness and multivariate polynomial interpolation see, for example, [15]. The measure of poisedness in the DFO algorithm is provided by a Newton Fundamental Polynomial mechanism (see [5] for details).

The concept of poisedness in combination with the Newton Fundamental Polynomials mechanism is very powerful in the DFO framework. It allows us not only to choose a good interpolation set from a given set of sample points but also to *find* a new sample point which improves the poisedness of the interpolation set. It is also possible to select a point which when replaced by another, preserves or improves the poisedness of the interpolation set. The core idea of Step 4 is based on these properties.

Sample points generated by Step 2 serve to produce smaller function values, but if appropriate they can be used to increase the size or to improve the poisedness of the interpolation set. On Step 4 the points are computed to increase the size or improve the poisedness of interpolation set, but they may also produce smaller function values. Both type of points are computed by minimizing a quadratic (possibly linear) function over a trust-region. For Step 4 the quadratic or linear function is a Newton Fundamental polynomial (see [5] for details).

Intuitively it can be said that the more poised the interpolation set is the better it "spans" the neighborhood. However, it is clear, that no matter how well-poised a set of 5 points is in a 6-dimensional space in general it can never produce a good approximation model. More formally we need at least a fully linear interpolation model to guarantee an acceptably accurate (at least in the neighborhood) approximation of the true function. This, in turn, means that we need a poised set of at least $n + 1$ points to insure progress of the algorithm in R^n. For that reason in this case the trust region radius should not be decreased on Step 5 until the interpolation becomes fully linear, since a smaller trust-region radius does not imply a more accurate model in general, if the model is sublinear.

It is also important that the interpolation set is sufficiently local. Therefore, it is sometimes preferable to have an interpolation set of a smaller size rather than include sample point that are very distant from the current iterate.

Properties of the DFO algorithm

Let us discuss the advantages of the algorithm. First, we note, that under reasonable assumptions this algorithm is globally convergent; i.e., it converges to a local stationary point starting from any initial solution (see [5]). To guarantee convergence the approximation model needs to be only fully linear. On every iteration only one or two function evaluations are needed, which means that the algorithm uses new information as soon as it becomes available. One can start the algorithm with only 2 interpolation points and proceed by expanding the model and by reducing the function at the same time. In the case of the 31 variable problem our implemented version of the algorithm produced a very good solution after about 180 function evaluations. To build even a single full quadratic model one needs at least 528 function evaluations. This confirms that, in practice, it it not necessary to have a full quadratic model for the algorithm to work acceptably.

Another advantage is that the points do not have to be on a certain lattice or any other predetermined structure, thus, if the method is combined with some prior analysis of the objective function, any information obtained by this analysis can be used in DFO. Finally, even though, theoretically, DFO is only guaranteed to converge to a local stationary point (just like most general nonlinear optimization algorithms), in practice, it seems to have a reasonable chance to find a global minimum or at least a good local minimum. This is supported by our computational results.

The objective function value is often computed in the simulation with a certain error. Hence, the algorithm should not break down in the presence of noise in the function values. Our algorithm proves to be very robust in case of noise. The reason is that the effect of noise becomes significant only when the trust region radius is small with respect to the noise level, which typically happens in the proximity of an approximate solution of the noisy problem. In contrast, a method based on gradient estimation through finite differences, may suffer dramatically from noise if its level is not known in advance, since the finite difference scheme may produce very bad estimates of the gradient.

Finally, the quadratic model, built at each iteration, is not defined uniquely. For a unique quadratic model in n variables one needs at least $(n+1)(n+2)/2$ points and their function values, which may be too expensive. As we often build models using many fewer points such models are not uniquely defined. There are several strategies for selecting a unique model from the pool of the possible interpolation models. The best strategy is to select such interpolation models which suit the particular objective function, based on the available information. For example, if due to previous experiment with the model, some variables appear less important than others, we can interpolate less important variables by linear terms only and have the quadratic terms interpolating only the important variables. On the other hand, if it is known that the objective function is separable with respect to some variables then the model can be made separable with respect to the same variables by setting certain coefficients to zero. This can reduce the number of points required for a more accurate interpolation, but it also requires additional information from the user. Currently, the best general strategy for the DFO algorithm appears to be selecting a model with the smallest Frobenius norm of the quadratic coefficient matrix.

Constraints

So far we considered unconstrained problems. Typically real-life problems have some kind of constraints. Here we assume that all constraints are defined by smooth functions, even if we we do not have complete informations about these functions. We consider 3 categories of constraints satisfying this requirement.

"Easy" constraints

These are general constraints of the form $g(x) \leq 0$, $h(x) = 0$, where g and h are continuously differentiable vector functions, whose values and first order partial derivatives can be computed relatively cheaply. The most trivial and the most frequently encountered example of such constraints are simple bounds on the variables; i.e., constraints of the type $a \leq x \leq b$. Another example is the linear inequality constraint of the Boeing helicopter problem. This constraint establishes an upper bound on the total mass of the helicopter blade.

Since the values and derivatives of the constraint functions are cheap to compute, solving a feasibility problem is also relatively easy. Recall that "cheap" or "easy" here means with respect to a single objective function evaluation, which is assumed to be very expensive. It, hence, seems appropriate to consider only feasible points and not to waste objective function evaluations on infeasible points. Let F denote a feasible region of a given optimization problem. If we substitute the trust-region minimization of Step 2 (and of Step 4, when applicable) by

$$m_k(x_k^+) = \min_{x \in \mathcal{B}_k \cap F} m_k(x). \tag{0.3}$$

then all points produced by our algorithm are feasible. On every iteration we solve at least one optimization problem with a quadratic objective function and the supplied general constraints, however the cost of solving such a problem is negligible when compared to a single true objective function evaluation.

With this modification to the trust-region subproblem our algorithm can be directly applied to problems with "easy" constraints.

It is worth noticing that our algorithm actually may benefit from the presence of equality constraints, since equality constraints reduce the degrees of freedom in the feasible set, which, in turn, reduces the number of points needed to build a quadratic interpolation. For example the set F satisfying the following two equations in $\mathbf{R}^4$

$$x_1 + x_3 - 1 = 0,$$
$$x_2 + x_4 = 0,$$

is a two-dimensional affine space. If we require the sample set to be contained in F then we effectively reduce our domain to a two-dimensional space and we consider interpolations of a function of 2 variables, instead of 4. Thus a full quadratic interpolation requires only 6 poised points. Naturally, 6 is the largest cardinality of a poised set that can be found in F.

The same mechanism of Newton Fundamental Polynomials can be used in the presence of equality constraints. As before, the method tries to produce the largest and best poised set in R^n with the only exceptions that all the sample points have to be contained in F. This way the largest and best poised set in F is generated automatically. This very useful feature of the Newton Fundamental Polynomials helps us to solve problems with "easy" constraints after very little modification to the DFO algorithm.

We considered the case when F is an affine space. Generally, a feasible set may have a complicated structure and may even be discontinuous. However, since we are only interested in local models (in theory, at least), we can assume that equality constraints are approximated by their first order Taylor series expansion at the current iterate. Then we can compute the dimension of the feasible set in the neighborhood of the iterate. Once we know the dimension of the feasible neighborhood we can efficiently compute the interpolation set in this neighborhood by the Newton Fundamental Polynomials mechanism.

We conclude that a problem with equality constraints may be solved with significantly fewer true function evaluations than another problem with the same number of variable but no equality constraints. Our computational results in Tables 1 and 3 support this conclusion.

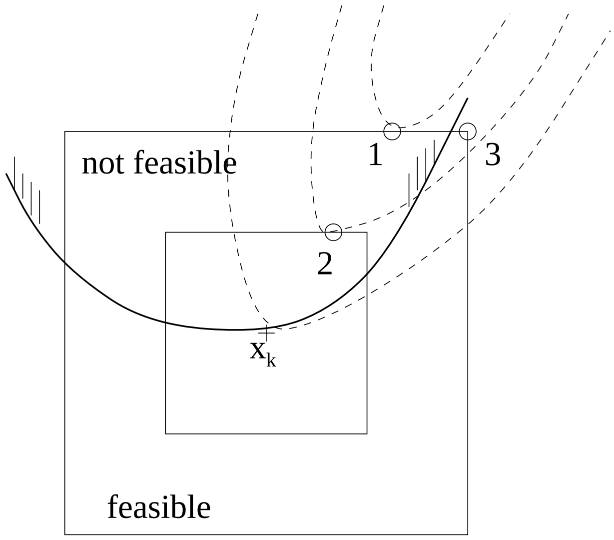

Figure 1: Early stopping in the presence of a virtual constraint

"Difficult" constraints

These are general nonlinear constraints which are, like the objective function, expensive to evaluate and their derivatives are not available. Some problems from Boeing are examples of such constraints. For example the engine nozzle design problem has a constraint that the air mass flow rate lies within the band specified by the engine manufacturer and this is not available as an analytic constraint. Other constraints include noise levels and structural weight. These constraints depend on the simulation code and their values are computed by the same analysis which is used for the objective function value computation. There are various possible approaches to problems with "difficult" constraints. Those approaches are based on using extention of the DFO algorithm. A study of these approaches is a subject to ongoing research and is outside the scope of this paper.

"Virtual" constraints

There are constraints that cannot be explicitly measured. The only thing known about them is whether or not such a constraint is violated at a given point. Moreover this information is costly to obtain. The concept of "virtual" constraint was inspired by the Boeing helicopter problem. In the case of this problem some function evaluations fail at certain points, which are feasible with respect to known constraints. The failure of the objective function computation takes about as much time as a successful computation. The sets of such points appear to be continuous but little else is known about them.

Our approach assumes that, in theory, the set of points at which function values can be successfully evaluated can be described exactly as a feasible set with respect to some constraints defined by smooth functions. We refer to these constraints as "virtual", since we do not know anything about them, neither can we compute their values. We can only check if a point is feasible with respect to the"virtual" constraints. To do so in the case of the helicopter problem one has to try to compute a function value at this point.

We use a heuristic approach to handle such constraints. Suppose that the point x_k^+ of Step 2 of the DFO algorithm is infeasible with respect to a "virtual" constraint. Then we reduce the radius of the trust-region (temporarily, until we complete Step 2) and repeat Step 2. Eventually either we obtain a new point, which is feasible and minimizes the model in some "smaller" trust-region, or the trust-region radius becomes too small. The latter case means that our current iterate lies on the border of the feasible region (with respect to the "virtual" constraints). We then assume that the model has a local minimum at the current iterate and we proceed to Step 4 as we usually do in such cases.

If an infeasible point is generated during Step 4, then a similar idea is applied, but with some modification. There is often more than one way to produce a point which improves a current interpolation set. Each of them involves a trust-region minimization. If one of these minimizations results in an infeasible point, we try other minimizations and if one of them produces a feasible point, we complete Step 4. If none produce a feasible point, then we reduce the trust-region radius (temporarily) and repeat Step 4. If the trust-region radius becomes too small we proceed to Step 5 assuming that the interpolation cannot be improved.

The flaw of this approach is clear: the model does not necessarily have a local minimum at x^k even if Step 2 fails to produce a new point. Figure 1 shows an example of such case. The dashed curves indicate the level sets of the model and the bold curve indicates the virtual constraint. The boxes are the trust-regions of different size. The trust-region minimization subproblem does not contain any information about the "virtual" constraint. As shown by Figure 1 it may be the case that all points produced by the trust-region minimization, e.g., points 1 and 2, are infeasible. However x_k is not a local minimum, since the value of the model at point 3 is smaller than that at x_k.

More sophisticate heuristics are the subject of future research. However, in many cases it may be too expensive to perform a more careful analysis of "virtual" constraints.

We would like to point out that at Step 4 the algorithm attempts to span the feasible set as accurately as possible, before cutting down the trust-region radius, so there is a good chance that points with smaller function values than the current iterate (if there are any) are eventually identified by Step 4, even if Step 2 fails to do so.

Computational results

First, we present the results of using DFO algorithm on problems from the CUTE [1] test set. We compare DFO with three other optimization methods that do not use first order derivative information: the parallel direct search method (PDS) [8], the nonlinear optimization package LANCELOT [4] with finite difference first order derivative approximations, and COBYLA [11], a derivative free method, which uses linear polynomial interpolation of the objective function. All algorithms are implemented in Fortran 77 in double precision, except COBYLA which is implemented in single precision. The trust-region minimization subproblem of the DFO algorithm is solved by NPSOL [9], a Fortran 77 nonlinear optimization package that uses an SQP approach.

Problem	#	# of function evaluations				final function value			
name	var.	DFO	PDS	LAN.	COB.	DFO	PDS	LAN.	COB.
BROWNAL	10	837	14603	66	103	9.2867D-07	1.3415D-03	1.1916D-11	9.3470D-09
HEART6LS	6	1350	37383	6652	8000	4.3167D-01	1.6566D+00	4.1859D-01	4.1839D+00*
SCHMVETT	3	53	2564	32	213	-3.0000D+00	-3.0000D+00	-3.0000D+00	-3.0000D+00
BROWNDEN	4	110	5381	281	540	8.5822D+04	8.5822D+04	8.5822D+04	8.5823D+04
ROSENBR	2	81	2307	94	8000	1.9716D-07	1.2265D-07	5.3797D-13	4.6102D-04*
DQDRTIC	10	403	74507	33	7223	1.6263D-20	1.1022D-04	1.6602D-23	3.8218D-06
GROWTHLS	3	243	2308	652	6529	1.2396D+01	1.2412D+01	1.0040D+00	1.2504D+01
SISSER	2	27	1795	33	46	1.2473D-06	9.3625D-20	1.3077D-08	1.1516D-20
CLIFF	2	75	3075	84	36	1.9979D-01	1.9979D-01	1.9979D-01	2.0099D-01
GULF	3	411	75780	148	8000	1.4075D-03	3.9483D-02	7.0987D-17	6.1563D+00*
SNAIL	2	246	2563	715	8000	1.2661D-08	2.6057D-10	4.8608D+00	7.2913D+00*
BIGGS6	6	1364	31239	802	8000	1.7195D-05	7.5488D-05	8.4384D-12	8.3687D-04*
CRAGGLVY	10	1026	13323	77	1663	1.8866D+00	1.8866D+00	1.8866D+00	1.8867D+00
EIGENALS	6	211	5895	35	1031	8.9164D-07	1.1905D-05	2.0612D-16	7.5428D-08
HAIRY	2	51	2563	357	3226	2.0000D+01	2.0000D+01	2.0000D+01	2.0000D+01
PFIT1LS	3	180	5124	216	8000	4.2637D-04	3.9728D-06	1.1969D+00	2.8891D-02*
VARDIM	10	2061	33035	165	4115	2.6730D-07	8.5690D-05	1.8092D-26	4.2233D-06
HART6	6	119	6151	57	124	-3.3229D+00	-3.3229D+00	-3.3229D+00	-3.3229D+00
MANCINO	10	276	11275	88	249	1.5268D-07	2.9906D-04	2.2874D-16	2.4312D-06
POWER	10	206	13067	187	368	2.6064D-06	1.6596D-13	8.0462D-09	6.8388D-18
HATFLDE	3	95	35844	66	8000	3.8660D-06	1.7398D-05	5.1207D-07	3.5686D-04*
WATSON	12	1919	76813	200	8000	4.3239D-05	2.5354D-05	2.0575D-07	7.3476D-04*
FMINSURF	16	1210	21265	224	654	1.0000D+00	1.0000D+00	1.0000D+00	1.0000D+00
DIXMAANK	15	1118	63504	2640	2006	1.0000D+00	1.0000D+00	1.0000D+00	1.0001D+00
MOREBV	10	476	75787	33	8000	6.0560D-07	1.0465D-05	1.9367D-13	2.2882D-06*
BRYBND	10	528	128011	133	8000	9.9818D-08	1.9679D-02	7.5942d-15	8.2470D-03*

Table 1: Results for unconstrained problems from CUTE

Problem	#	#	# of function evaluations			final function value		
name	var.	constr.	DFO	LAN.	COB.	DFO	LAN.	COB.
HS22	2	2	15	24	24	1.0000D+00	1.0000D+00	1.0000D+00
HS23	2	5	16	1456	28	2.0000D+00	2.0000D+00	2.0000D+00
HS26	3	1	49	1677	1001	1.9355D-09	6.1688D-09	1.0250D-07
HS34	3	2	22	405	41	-8.3403D-01	-8.3403D-01	-8.3403D-01
HS38	4	0	536	128	382	1.6583D-07	4.3607D-13	7.8770D+00
HS44	4	6	26	35	45	-1.5000D+01	-1.5000D+01	-1.5000D+01
HS65	3	1	35	1984	102	9.5353D-01	9.5353D-01	9.5353D-01
HS76	4	3	29	130	76	-4.6818D+00	-4.6818D+00	-4.6818D+00
HS100	7	4	127	6979	175	6.8063D+02	6.9138D+02*	6.8063D+02
HS106	8	6	63	196	93	7.0492D+03*	1.4995D+04	1.4944D+04
HS108	9	13	62	2416	177	-8.6603D-01	-8.6602D-01	-8.6603D-01
HS116	13	14	87	12168	119	9.7485D+01*	5.0000D+01	3.5911D+02
HS268	5	5	84	5286	676	-1.8190D-11	2.6576D+01	5.9238D+00

Table 2: Results for problems from CUTE with inequality constraints only

Problem	#	#	# of function eval.		final function value	
name	var.	constr.	DFO	LAN.	DFO	LAN.
HS32	3	2	15	24	1.0000D+00	1.0000D+00
HS48	5	2	31	24	2.1132D-20	2.1839D-16
HS49	5	2	85	132	2.3516D-06	1.2514D-07
HS54	6	1	27	38	-1.5391D-01*	-1.5403D-01
HS56	7	4	37	3701	-3.4560D+00	-3.4560D+00
HS59	2	3	32	2777	-6.7495D+00	2.9127D+01*
HS63	3	2	12	313	9.6172D+02	9.6172D+02
HS68	4	2	127	4761	-9.2042D-01	-9.2824D-01*
HS69	4	2	46	4965	-9.4134D+02*	-9.5694D+02*
HS74	4	5	71	386	5.1265D+03	5.1265D+03
HS79	5	3	25	220	7.8777D-02	7.8777D-02
HS107	9	6	29	270	5.0550D+03	5.0550D+03
HS111	10	3	227	10941	-4.7761D+01	-4.7744D+01*
HS112	10	3	337	366	-4.7761D+01	-4.7761D+01
HS114	10	11	8	9481	-9.1628D+02*	-1.8261D+03*
HS119	16	8	91	725	2.4490D+02	2.4490D+02

Table 3: Results for problems from CUTE with equality constraints

In Table 1 we list the results for the unconstrained problems from CUTE. In Tables 2 and 3 there are results for a selection of constrained problems from the test set of Hock and Schittkowski [10]. The problems in Table 2 do not have equality constraints, whereas the problems in Table 3 do. The implementation of PDS, which we used, handles only unconstrained problems. The implementation of COBYLA does not handle equality constraints. All constraints of problems from the Hock and Shittkowski set were treated as "easy" constraints.

We list the number of function evaluations that each algorithm took to solve the problem. We also list the final function values that each algorithm achieved. We do not list the CPU time, since it is not relevant in our context. The "*" symbol next to the function values in the DFO column indicates that the algorithm terminated early because the trust-region minimization algorithm failed to produce a feasible solution. This, clearly, only happens in constrained cases. Even when the algorithm terminates early, it often does so after finding a local optimal solution, as in the case of HS54 and HS106. In other columns the "*" indicates that an algorithm terminated early because the limit on the number of iterations was reached.

The stopping tolerance used by DFO was set to 10^{-4}, for the other algorithms the tolerance was set to appropriate comparable default values.

Next we present the results of using DFO to solve the Boeing helicopter problem with 11 and 31 variables. In Fig. 2 we show the results of 10 runs of the DFO algorithm on the 11-variable instance of the helicopter problem. The final values are quite different. The reason is that this problem appears to have a lot of local minima. All runs were initialized with the same solution, but the second point (at least two points are needed for the initial interpolation set) is chosen randomly. Hence, the choice of the second point determines the basin of a local minimum where DFO later converges. In Fig. 3 there are the results of 10 runs for the 31-variable instance.

The number of function evaluations includes both successful and unsuccessful function evaluations. The rate of failure in function evaluations (due to the "virtual" constraint) is about 15%-20% on the average.

Note that although we obtained different local solutions most of them were better than the solutions previously obtained by other means. Notice as well,

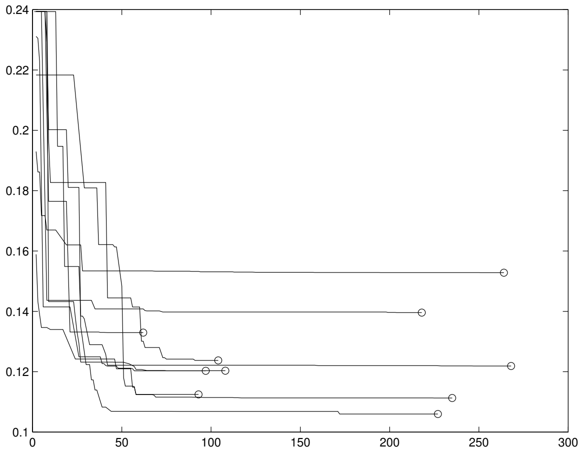

Figure 2: Ten runs of DFO on the helicopter problem with 11 variables

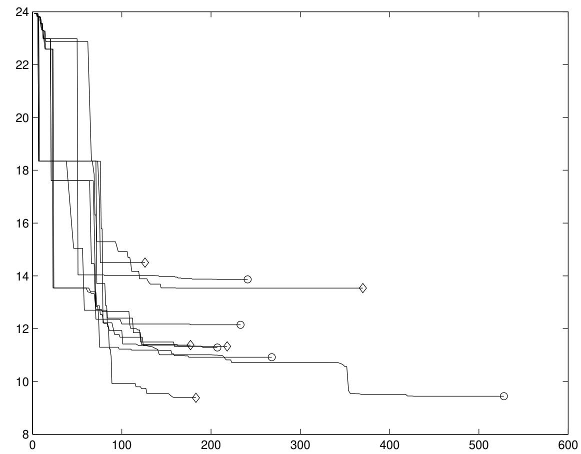

Figure 3: Ten runs of DFO on the helicopter problem with 31 variables

that even though the algorithm typically stops after about 200 function evaluations, it obtains very good solutions after about 60 function evaluations. For comparison with the result of using other methods for this problems see [3].

References

[1] I. Bongartz, A. R. Conn, N.I.M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[2] A. J. Booker, "Design and analysis of computer experiments", 1998. To be presented at the special session on Approximate Models for Multidisciplinary Design Optimization, Seventh AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization. St. Louis, MO.

[3] A. J. Booker, J. E. Dennis, P.D. Frank, David B. Serafini, V. Torczon and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. Technical Report SSGTECH-98-005, Boeing Shared Services Group, Applied Research and Technology, Seattle, WA, 1998.

[4] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[5] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108, Cambridge, UK, 1997. Cambridge University Press.

[6] A. R. Conn, K. Scheinberg and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79: 397–414, 1997.

[7] A. R. Conn and Ph. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In G. Di Pillo and F. Gianessi, editors, *Nonlinear Optimization and Applications*, pages 27–47, New York, 1996. Plenum Publishing.

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

[9] P. E. Gill and W. Murray and M. A. Saunders and M. H. Wright. User's Guide for NPSOL (Version 4.0): A Fortran package for nonlinear programming, Technical Report SOL 86-2 (1986), Department of Operations Research, Stanford University, Stanford, CA 94305, USA.

[10] W. Hock and K. Schittkowski, Test Examples for Nonlinear Programming Codes. *Lectures Notes in Economics and Mathematical Systems*, volume 187, Springer Verlag, Berlin, 1981.

[11] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275, pages 51–67, Dordrecht, NL, 1994. Kluwer Academic Publishers.

[12] M. J. D. Powell. A direct search optimization method that models the objective by quadratic interpolation. Presentation at the 5th Stockholm Optimization Days, 1994.

[13] M. J. D. Powell. Trust region methods that employ quadratic interpolation to the objective function. Presentation at the 5th SIAM Conference on Optimization, 1996.

[14] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, to appear, 1998.

[15] Th. Sauer and Yuan Xu. On multivariate Lagrange interpolation. *Mathematics of Computation*, 64:1147–1170, 1995.

[16] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997.