OPTIMIZING PARTIALLY SEPARABLE FUNCTIONS WITHOUT DERIVATIVES

by B. Colson[1] and Ph. L. Toint[1]

Report 2003/20 6th November 2003

[1] Department of Mathematics,
University of Namur,
61, rue de Bruxelles, B-5000 Namur, Belgium,
Email: benoit.colson@fundp.ac.be, philippe.toint@fundp.ac.be

Optimizing Partially Separable Functions without Derivatives

Benoît COLSON* and Philippe L. TOINT†

October 2003

Abstract. We present an algorithm for solving nonlinear programming problems involving a partially separable objective function whose derivatives are assumed to be unavailable. At each iteration we construct a quadratic interpolation model of the objective function around the current iterate and minimize this model to obtain a trial step. The whole process is embedded within a trust-region framework. We further propose to use ideas of Curtis, Powell and Reid to minimize the number of calls to the objective function in the part of the derivative-free algorithm that improves the geometry of the interpolation set. Numerical experiments tend to confirm the promising behaviour of the algorithm.
Keywords: partially separable functions, derivative-free optimization, multivariate interpolation, trust-region algorithms.

1 Introduction

Derivative-Free Optimization (DFO) is concerned with the design and the analysis of algorithms for solving mathematical programs involving functions whose derivatives are not available. This may be due to the fact that the evaluation of these derivatives is very difficult, unreliable or time-consuming, or their computation requires the solution of another problem, or even because the objective and/or constraint functions themselves are complex. The latter situation may occur when the computation of the accurate value of such a function at a given point requires calls to extremely expensive codes and solvers or because the related source codes may be unavailable or unmodifiable in which case they must be considered as black boxes. Also, such complex functions may be the output of a simulation or some physical, chemical or econometrical measurements or experiments, for instance.

DFO algorithms are designed in such a way that the number of function evaluations they require is minimized, which by the way makes this number the usual criterion for the assessment and comparison of algorithms. Existing methods may be regrouped in four broad categories. *Direct search and pattern search* methods (see *e.g.* Torczon [30], Lewis, Torczon and Trosset [18] and Audet and Dennis [1]) may be viewed as sampling methods in the sense that they are based on an exploration of the space region under consideration; they

*Department of Mathematics, FUNDP (The University of Namur), Rempart de la Vierge, 8, B-5000 Namur, Belgium. E-mail address: benoit.colson@fundp.ac.be

†Department of Mathematics, FUNDP (The University of Namur), Rempart de la Vierge, 8, B-5000 Namur, Belgium. E-mail address: philippe.toint@fundp.ac.be

perform a search by means of exploratory moves considering the behaviour of the objective function at a "pattern" of points that is independent of this function. A second class of methods was introduced by Powell [20], who described a *line search* method which proceeds in stages, each of them consisting of a sequence of $n+1$ one-dimensional searches, the latter being governed by the minimization of a quadratic interpolant computed for each direction. Lucidi and Sciandrone [19] reported some numerical experiments obtained with the algorithm of Grippo, Lampariello and Lucidi [17], who introduced a line search procedure where the usual conditions involving the gradient of the objective function are replaced by requirements on the values of f at points of the form $x \pm \alpha d$, which in turn allowed to design another algorithm for minimizing f using the set of orthonormal coordinate vectors as directions at each step. Powell [21, 22] also introduced another approach where finite-difference techniques are coupled with *quasi-Newton* algorithms, which constitute the third class of algorithms for DFO problems. The fourth and last class contains *trust-region* methods (see *e.g.* Powell [25] and Conn, Scheinberg and Toint [8]), in which all available function values are used to build a polynomial model interpolating the objective function at all those points at which its value is known. The model is then minimized within a trust region and a new point is obtained. The latter point is evaluated and while this enlarges the interpolation set this also allows to check whether the objective function is improved. The whole process is then repeated until convergence occurs.

Several recent contributions considered unconstrained DFO problems where the objective function is known to have some structure despite the fact that its derivatives are not available (see *e.g.* Colson and Toint [3, 4] and Powell [24]). The reason for studying algorithms adapted to these particular problems is that a suitable exploitation of the problem structure is expected to result in important savings in terms of computational effort. The research work we describe in this paper is in line with [3, 4] in which we considered algorithms for solving unconstrained problems exploiting the banded and sparse structure of the objective function respectively. In this article we introduce another trust-region algorithm for solving unconstrained problems where this time the objective function is *partially separable*, that is it may be expressed as

$$\min f(x) = \sum_{i=1}^{M} f_i(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and each *individual function* $f_i(x)$ is a function depending on some of the components $x_j, j \in I_i$ for some index sets $I_i \subseteq \{1, \ldots, n\}$ $(i = 1, \ldots, M)$. We assume that the cardinality of these M subsets is given by $|I_i| = n_i$ and that the n_i-dimensional subspace of $\mathbb{R}^n$ is denoted by R_i $(i = 1, \ldots, M)$.

The concept of partial separability was introduced by Griewank and Toint [14, 16] and it was also studied by Conn, Gould and Toint [6] in the framework of the development of the LANCELOT package. Many optimization problems involve partially separable functions (see *e.g.* the test problems collected by Toint [29]) and Griewank and Toint [15] show that every twice continuously differentiable function with a sparse Hessian is partially separable. It is remarkable that, although this is only an existence result and does not specify the particular decomposition (1), we are not aware of any practical problem having a sparse Hessian and whose partially separable decomposition is not explicitly known. In the context of problems where derivatives are unavailable, the partially separable structure often arises from the observation that the objective has a block structure involving different subsets of variables, even if each block is a black box in itself.

The next section provides some background material consisting of multivariate interpolation tools before describing the above mentioned algorithm of Conn, Scheinberg and Toint [8]. The reason for presenting it is that it served as a basis for the development of our algorithm for partially separable functions, which is described in Section 3. Section 4 details the numerical experience gained so far and it is followed by a short conclusion and several ideas for improving our algorithm further.

2 Trust regions and multivariate interpolation models

The new algorithm discussed in the present paper is a variation of that proposed by Conn, Scheinberg and Toint [8, 9] and is of trust-region type. At iteration k, this algorithm first constructs a quadratic model

$$m_k(x_k + s) = f(x_k) + \langle g_k, s\rangle + \frac{1}{2}\langle s, H_k s\rangle, \tag{2}$$

for some $g_k \in \mathbb{R}^n$ and some symmetric n-by-n matrix H_k, which is intended to be valid in a neighbourhood of the current iterate $x_k \in \mathbb{R}^n$ (the *trust region*) defined by

$$\mathcal{B}_k = \{x_k + s : s \in \mathbb{R}^n \text{ and } \|s\| \leq \Delta_k\}, \tag{3}$$

for some trust-region *radius* Δ_k.

In our approach, the model (2) is chosen to interpolate the value of a function f at a set $Y_k = \{y_i\}$ of points containing x_k, yielding

$$f(y_i) = m_k(y_i) \quad \text{for all } y_i \in Y. \tag{4}$$

We refer the interested reader to de Boor and Ron [12], Sauer [27], Sauer and Xu [28], Conn, Gould ans Toint [7] and Conn, Scheinberg and Vicente [10] for details of how the

$$p = n + \frac{1}{2}n(n+1) = \frac{1}{2}(n+1)(n+2) - 1 \tag{5}$$

entries of g_k and H_k can be computed given the set Y (of fixed cardinality p) and the collection of associated function values $F = \{f(y_i)\}_{y_i \in Y}$. It is however important to point out that the knowledge of Y and F is not sufficient to guarantee the existence and uniqueness of suitable g_k and H_k, but that a further geometric condition (known as *poisedness*) is also required. This condition ensures that the points of Y do not collapse in a lower dimensional space or do not lie on a quadratic curve (which would then span an infinite set of interpolating quadratic polynomials). If one chooses to build the interpolating quadratic using quadratic Newton fundamental polynomials as a basis (which is the approach taken by Conn, Scheinberg and Toint [9] and Conn,Gould and Toint [7]), poisedness is ensured if and only if the *pivots*, that is the values of the non-normalized Newton fundamental polynomials at their associated[1] interpolation point $|N_i^u(y_i)|$ are all different from zero (we denote

[1] We again refer the reader to de Boor and Ron [12], Sauer [27], Sauer and Xu [28], Conn, Gould ans Toint [7] and Conn, Scheinberg and Vicente [10] for details on how fundamental Newton polynomials are associated with interpolation points and normalized.

the non-normalized Newton fundamental polynomials by N_i^u and their normalized version by N_i). While from a theoretical point of view this may be sufficient to ensure poisedness, in practice we must verify that,

$$|N_i^u(y_i)| \geq \theta \quad \text{for all } y_i \in Y \tag{6}$$

for some *pivoting threshold* $\theta > 0$. If condition (6) is satisfied, then the interpolation problem (4) is said to be *well poised*, and the entries of g_k and H_k can be computed safely. If this condition fails, we may have to improve the geometry of our interpolation set to ensure it. Following Conn, Scheinberg and Toint [9] and Conn, Gould and Toint [7], a reasonable strategy for *improving the geometry* of this set might be to replace a point $y_- = y_i \neq x_k$ by another point y_+ such that $|N_i(y_+)|$ is larger, for instance by choosing

$$y_+ = \text{argmax}_{y \in \mathcal{B}_k} |N_i(y)| \tag{7}$$

provided $|N_i(y_+)| \geq 1+\theta$ since otherwise the geometry was already fine. Note that solving (7) reduces to solving two classical trust-region subproblem, for which good algorithms exists (see Chapter 7 of Conn, Gould and Toint [7], for instance).

Once we have computed the model $m_k(x_k + s)$ in the neighbourhood of the current iterate x_k, we follow the classical procedure of trust-region methods and compute the step $s_k \in \mathbb{R}^n$ minimizing the polynomial model m_k over the trust region defined by (3). Again, this can be done using any of the well-known algorithms described in Chapter 7 of [7]. If

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \tag{8}$$

is sufficiently positive, we say that iteration k is *successful*, define our next iterate x_{k+1} to be $x_k + s_k$ and possibly enlarge our trust-region radius Δ_k. If ρ_k is not sufficiently positive, our iteration is deemed *unsuccessful*, the current iterate is not altered and the trust-region radius is potentially reduced.

We now consider three particular issues arising in the particular framework of derivative-free optimization. The first point regards the convergence test at the end of each iteration. While in classical trust-region methods it may be sufficient to check that the gradient of f is small enough at the current point, our derivative-free framework makes things more difficult since we cannot compute the gradient $\nabla f(x)$. It is nevertheless the case that, if the model (2) is a sufficiently accurate representation of the objective function, we may assume that

$$\|g_k\| \approx \|\nabla f(x_k)\|.$$

Hence, as soon as the quadratic model m_k has been updated, we compute $\|g_k\|$ and check whether it is small enough. If this is the case, we ensure that the model is sufficiently accurate by making it *valid* in some ball centered at x_k, *i.e.* we verify that the points of the interpolation set are sufficiently close to x_k and that the associated Newton fundamental polynomials are bounded (see Theorem 9.4.4 of [7]). If $\|g_k\|$ remains small after this verification, convergence is declared and the algorithm terminates (see Lemma 9.1.2 of [7] for a proof that this process is well defined and finite whenever $\epsilon_c > 0$). Otherwise the optimization process is continued.

The second issue is the way to use new available function values in a suitable way. To see this, let us now assume that at some iteration k we get a point $x_k^+ = x_k + s_k$ of associated function value $f(x_k^+)$, one may consider the problem of finding the best way to make x_k^+ play a role in the next iterations when building the interpolating quadratic. Indeed, since in our derivative-free optimization framework the function evaluations are expensive, it is natural to take advantage of the fact that $f(x_k^+)$ is known to check whether x_k^+ may be added to the set of interpolation points or not. If $|Y| < p$, which generally occurs in the first few iterations of the process, we may simply add x_k^+ to Y, which allows to progressively complete the set of interpolation points. Otherwise, if $|Y| = p$, we try to find a point in Y, say y_-, which can advantageously be replaced by x_k^+. This replacement is performed in a way that makes the pivots as large as possible in order to obtain a well-poised interpolation problem. More precisely, we proceed as follows:

Let p_i ($i = 1, \ldots, M$) denote the size of a complete interpolation set for function f_i.
We first initialize the radius Δ_{geom} – as follows:

$$\Delta_{geom} = \begin{cases} \min(\|s_k\|, \Delta_k) & \textit{if the iteration is successful,} \\ \min(\|s_k\|, \Delta_k)/\gamma_{div} & \textit{if the iteration is unsuccessful} \\ & \textit{and the model is valid,} \\ \min(\|s_k\|, \Delta_{ref}/\gamma_{lim}) & \textit{otherwise,} \end{cases} \tag{9}$$

where $\gamma_{div} = 1.75$, Δ_{ref} is the trust-region radius of the most recent successful iteration and

$$\gamma_{lim} = \max(10, p_i)\gamma_{div}.$$

We then look for the point y_- whose distance to the base is the largest.

- *if $\|y_- - x_{k+1}\|$ is not too small (*e.g.* larger than $1.5\Delta_k$) and the value of the fundamental polynomial associated to y_- evaluated at x_k^+ is larger than*

$$2\left(\frac{\Delta_{geom}}{\|y_- - x_{k+1}\|}\right)^2,$$

then we replace y_- by x_k^+;

- *otherwise, we choose to select y_- to be the point associated to the fundamental polynomial whose absolute value is maximal at x_k^+, and we replace y_- by x_k^+ provided this absolute value is sufficiently large (*e.g.* larger than 1).*

Finally, a third issue is the management of the trust-region radius. In classical trust-region methods, the radius Δ_k is always decreased at unsuccessful iterations. In our framework, however, we must first verify that the interpolation set is poised before reducing Δ_k since a bad geometry might be the main reason for the iteration to be unsuccessful. If Y is not well poised, we thus have to improve its geometry (using – again – the procedure described above) possibly reducing Δ_k. It is particularly important not to modify Δ_k too early in the process as this would often impose too small steps and cause the algorithm to be excessively slow.

3 A trust-region algorithm for minimizing partially separable functions

We now turn back to the case of unconstrained problems involving a partially separable function (1). Intuitively, we may compute a solution to problem (1) by applying the interpolation ideas outlined above to each individual function $f_i(x)$ $(i = 1, \dots, M)$: we thus consider M interpolation sets Y_i whose points are used to construct M quadratic models approximating f_i around the current iterate x_k:

$$m_{i,k}(x_k + s) = f_i(x_k) + g_{i,k}^T s + \frac{1}{2} s^T H_{i,k} s, \qquad i = 1, \dots, M, \tag{10}$$

where each $g_{i,k}$ is a vector of $\mathbb{R}^{n_i}$ and $H_{i,k}$ is an n_i-by-n_i matrix with real coefficients for $i = 1, \dots, M$. Note that each model $m_{i,k}$ actually depends on the n_i-dimensional vectors of the subspace R_i $(i = 1, \dots, M)$ and approximate the f_i's around the *projection* of x_k onto R_i, even if, for the sake of simplicity, our notations in (10) keep this projection implicit.

Once the models (10) interpolating the individual functions are computed, a quadratic model of the global objective function f at x_k is trivially obtained as

$$\begin{aligned} m_k(x_k + s) &= \sum_{i=1}^{M} m_{i,k}(x_k + s), \\ &= f(x_k) + g_k^T s + \frac{1}{2} s^T H_k s, \end{aligned} \tag{11}$$

where the vector $g_k \in \mathbb{R}^n$ and the matrix $H_k \in \mathbb{R}^{n \times n}$ are built from "incomplete" – or "partial" – vectors $g_{i,k}$ and matrices $H_{i,k}$ $(i = 1, \dots, M)$. As usual in trust-region methods, the model (11) can then be minimized within the current trust region $\mathcal{B}_k$, which yields a new point $x_k + s_k$ at which the objective function f may be evaluated.

However, we now have to manage M interpolation sets Y_i $(i = 1, \dots, M)$ instead of a single one. We then have to consider two possible cases: either it is possible for the user to evaluate a single element function $f_i(x)$ independently of the others, or the objective function must always be evaluated as a whole (i.e. only the collection of values $\{f_i(x)\}_{i=1}^M$ can be computed for a given x). In the second case, a straightforward implementation of our algorithm could be very expensive in terms of function evaluations if we blindly apply procedures similar to those of the previous section to each Y_i for geometry improvement or for replacing an existing interpolation point with a new one. Indeed, computing a vector that improves the geometry of the j-th interpolation set yields a vector having only n_j "useful" components, and it is not clear which values should be assigned to the remaining $n - n_j$ components. Our strategy in this case is to group the necessary individual function evaluations by applying the CPR procedure of Curtis, Powell and Reid [11] for estimating sparse Jacobian matrices to the $n \times M$ occurence matrix of variables x_j into elements f_i defined by

$$d_{ji} = \begin{cases} 1 & \text{if function } f_i \text{ depends on } x_j \\ 0 & \text{otherwise,} \end{cases}$$

with $i \in \{1, \dots, M\}$ and $j \in \{1, \dots, n\}$.

Now, at some iteration k, we denote by L_k the set of indices corresponding to the interpolation sets for which a geometry improvement is requested. If $L_k = \{i_1, i_2, \dots, i_l\} \neq \emptyset$

(that is $|L_k| > 0$), we partition the set of columns of D in a number of groups, each of them containing columns corresponding to individual functions whose associated index sets have an empty intersection. Hence the individual functions whose associated columns belong to the same subset depend on strictly different components of a vector $x \in \mathbb{R}^n$. There are various ways to obtain such a grouping. For the experiments reported below, we have chosen the *greedy* approach originally proposed by Curtis, Powell and Reid, which is summarized as follows for $L_k = \{i_1, i_2, \ldots, i_L\}$:

- *a new group $L_{k,1}$ is first created that contains column i_1;*
- *the function i_2 is then considered and one checks whether f_{i_1} and f_{i_2} have common variables (that is whether $d_{j1} + d_{j2} > 1$ for all $j = 1, \ldots, n$): if this is not the case, then $L_{k,1}$ is replaced by $L_{k,1} \cup \{i_2\}$; otherwise, a new group $L_{k,2}$ is created containing column i_2;*
- *the process is repeated, considering the remaining elements of L_k in increasing order, and adding the corresponding columns to one of the existing subsets when possible, or creating a new group when this proves impossible.*

The authors are aware that it might be more efficient to use more sophisticate techniques, such as the graph coloring method of Coleman and Moré [2], to further improve the efficiency of this approach.

We now summarize the main steps of our algorithm. Note that at this point we must consider the two above mentioned possible frameworks: the first one corresponds to the situation where the functions f_i ($i = 1, \ldots, M$) are accessible individually while in the second one the value of an individual function f_i can only be obtained by calling the routine for evaluating the whole function f. We denote the corresponding versions of the algorithm by **I** and **G** respectively.

Partially Separable functions and Derivative-Free Optimization

Step 0. Initialization. *An initial point x_0 is given, as well as a trust-region radius $\Delta_0 > 0$, a user defined tolerance $\epsilon_c > 0$ and constants $\alpha \in (0, 1)$, $\mu > 0$, γ_1, γ_2, η_1 and η_2 satisfying $0 < \gamma_1 \leq \gamma_2 < 1$ and $0 < \eta_1 \leq \eta_2 < 1$. We also initialize that the parameters for geometry improvements γ_{div}, γ_{lim} and we choose $\Delta_{ref} = \Delta_0$. It is assumed that for each $i = 1, \ldots, M$ there exists a (possibly incomplete) set Y_i of interpolation points. The iteration counter k is set to 0.*

Step 1. Computing the models. *Compute quadratic interpolation polynomials $m_{i,k}$ (for $i = 1, \ldots, M$) approximating each f_i around the projection of x_k onto each R_i, as defined by (10).*
Form the complete model m_k defined by (11).

Step 2. Convergence test. *If $\|g_k\| \leq \epsilon_c$ and Y is well poised in a ball of radius $0 < \delta \leq \mu\|g_k\|$ centered at x_k stop. Otherwise* improve *the geometry until Y is well poised in a ball of radius $0 < \delta \leq \alpha\mu\|g_k\|$ centered at x_k and return to the beginning of Step 2.*

Step 3. Minimizing the models. *Solve*

$$\begin{array}{ll} \min_{s\in\mathbb{R}^n} & m_k(x_k+s) \\ \textit{s.t.} & \|s\| \leq \Delta_k, \end{array}$$

and obtain s_k.

Step 4. Step acceptance. *Compute* ρ_k *from (8). If* $\rho_k > \eta_1$ *then* $x_{k+1} := x_k + s_k$ *and* $x^+ := x_k$*, otherwise set* $x_{k+1} := x_k$ *and* $x^+ := x_k + s_k$.
Update Δ_k *as follows:*

$$\Delta_{k+1} \in \left\{ \begin{array}{ll} [\Delta_k, \infty) & \textit{if } \rho_k \geq \eta_2, \\ [\gamma_2\Delta_k, \Delta_k] & \textit{if } \rho_k \in [\eta_1, \eta_2), \\ [\gamma_1\Delta_k, \gamma_2\Delta_k] & \textit{if } \rho_k < \eta_1. \end{array} \right.$$

Step 5. Improve the geometry. *Update* Δ_{geom} *as in (9).*

Version I: *for* $i = 1, \ldots, M$*, compute the projection of* x^+ *on* R_i *and see if it can be advantageously added to* Y_i *(i.e. making its geometry better, as described in Section 2), possibly replacing an existing point. If this is the case, modify* Y_i *and update the fundamental polynomials related to it. Otherwise, try and compute an additional point that could improve the geometry* Y_i *(yielding larger pivots); if this is possible, compute this point as described in Section 2, update* Y_i *and the corresponding fundamental polynomials.*

Version G: *first set* $L_k := \emptyset$ *and then loop over the individual functions* f_i *and their interpolation sets* Y_i *(*$i = 1, \ldots, M$*) as above. If the insertion or replacement of a point in the interpolation set* Y_i *is possible, then proceed as before. Otherwise, add the index* i *to* L_k.
If $L_k \neq \emptyset$*, apply the above described CPR procedure to group the indices in* L_k*. The number of resulting groups gives the number of new points to compute for improving the geometry of the interpolation sets* Y_j *such that* $j \in L_k$.
If incidentally one of these new points has a better objective function value than x_{k+1}*, then replace* x_{k+1} *and update all fundamental polynomials.*

Go to Step 1.

Note that the trust-region radius Δ_k remains "global", *i.e.* it is related to f and not to the (possible) individual improvements in terms of the f_i's.

4 Numerical experience

The Fortran code resulting from the implementation of this algorithm is named PSDFO and the purpose of this section is to discuss preliminary numerical results obtained with it to solve unconstrained problems involving a partially separable objective function. The two above described versions of our algorithm were implemented and are denoted by PSDFO(I) and PSDFO(G) in the sequel. Table 1 provides the list of tested test problems. All these problems are part of the CUTEr (see [13]) collection, most of them being already described in Toint [29]. Note that since the primary goal of the CUTEr collection is to regroup test

Problem name	Dimension	Specifity	M
`ARWHEAD`	$n = 10, 15, 20, 25, 50, 100, 200$	sparse	$n-1$
`BDQRTIC`	$n = 10, 15, 20, 25, 50, 100, 200$	banded	$n-4$
`BDVALUE`	$n = 10, 20, 50, 100, 200$	banded	n
`BRYDN3D`	$n = 10, 20, 50, 100, 200$	banded	n
`CHROSEN`	$n = 10, 15, 20, 25, 50, 100, 200$	banded	$n-1$
`CRAGGLVY`	$n = 10, 20, 50, 100, 200$	banded	$(n-2)/2$
`DQDRTIC`	$n = 10, 20, 50, 100, 200$	banded	$n-2$
`EXTPOWELLSG`	$n = 10, 21, 49, 100, 201$	sparse	$(n-1)/3$
`EXTROSNB`	$n = 10, 20, 50, 100, 200$	sparse	$n-1$
`GENHUMPS`	$n = 10, 20, 50, 100, 200$	sparse	$n-1$
`LIARWHD`	$n = 10, 20, 50, 100, 200$	sparse	$n-1$
`MOREBV`	$n = 10, 20, 50, 100, 200$	banded	n
`ROSENBR`	$n = 10, 20, 50, 100, 200$	banded	$n-1$
`SCHMVETT`	$n = 10, 20, 50, 100, 200$	banded	$n-2$
`WOODS`	$n = 12, 20, 48, 100, 200$	sparse	$n/4$

Table 1: Problem specifications.

problems for general nonlinear programming software packages, most CUTEr problems are *not* derivative-free problems. Besides their availability in the public domain, the main advantage of using the CUTEr problems is that their solution is known, a very useful feature for testing.

Before describing the test results, we emphasize that the framework of partially separable functions makes it necessary to distinguish two different types of function evaluations: those related to the global objective function f and those related to the element functions f_i ($i = 1, \ldots, M$). The PSDFO package is implemented in such a way that a subroutine (named `UFN_I`) allows to evaluate *one* particular individual function at a time. As a result, when the value of the global objective function is required (*e.g.* after computation of an improvement step s_k), PSDFO calls the subroutine `UFN_I` M times (once for every element function). As we wish to compare the number of function evaluations required by our algorithm with that required by other approaches, we report below the equivalent number of objective function evaluations, which is obtained by dividing the total number of element functions evaluations by M. Since we did not observe major variations in the number of calls to `UFN_I` for evaluating the different element functions, we consider that this equivalent number of objective function evaluations is an appropriate measure for the assessment of the performance of PSDFO.

4.1 Comparison of the approaches

We first provide some numbers to compare PSDFO with two other packages: UDFO is the basic implementation of the algorithm of Conn, Scheinberg and Toint [8] and UDFO(S) is one of its two extensions considered in [3, 4] (for banded and sparse Hessian matrices); UOBYQA is Powell's package described in [25] while UOBSQA and UOBDQA are its extensions taking sparsity into account and constructing approximations having a diagonal

Hessian respectively (see [24]).

Table 2 summarizes the performance of these five packages for the three problems of the CUTE collection considered by Powell (see Table 1 in [24]). We also consider the same

Problem	Dimension	UDFO	UDFO(S)	UOBSQA	UOBDQA	PSDFO(I)	PSDFO(G)
ARWHEAD	$n = 10$	311	150	118	105	30	54
	$n = 15$	790	339	170	164	30	56
	$n = 20$	1268	333	225	260	35	161
	$n = 25$	1478	448	296	277	37	198
BDQRTIC	$n = 10$	519	327	350	288	348	358
	$n = 15$	1014	639	594	385	345	382
	$n = 20$	1610	893	855	534	509	596
	$n = 25$	2615	940	1016	705	360	542
CHROSEN	$n = 10$	2032	775	247	3652	85	123
	$n = 15$	4901	1660	431	4590	91	154
	$n = 20$	>10000	3268	553	5871	95	168
	$n = 25$	>10000	5015	736	5943	98	173

Table 2: Comparison of the number of function evaluations required by DFO solvers for solving medium-size instances of problems.

problem dimensions, that is $n = 10, 15, 20$ and 25 and as in [24] we focus on the number of function evaluations required for solving the problems. Figure 1 shows the evolution of the number of function evaluations required by the five solvers. As shown by this figure and the results in Table 2, both versions of PSDFO generally require much fewer function evaluations than the other packages. The reason for these savings in function evaluations may be explained by the fact that PSDFO obtains complete interpolation sets more rapidly since the dimensions n_i ($i = 1, \dots, M$) are (sometimes considerably) smaller than the dimension n of the problem. More precisely, the number of p function evaluations that is required by UDFO to obtain a full interpolation set is given by

$$\begin{array}{lll} \frac{1}{2}(n+1)(n+2)-1 & \text{for UDFO} & \text{(see (5))} \\ n + n_H & \text{for UDFO(S)} & \\ \frac{1}{2}(\bar{n}+1)(\bar{n}+2)-1 & \text{for PSDFO} & \end{array}$$

where n_H is the number of nonzero entries in the lower triangular part of the Hessian and where

$$\bar{n} = \max_{1 \le i \le M} n_i \ll n$$

which is typically independent of the actual size of the problem. (Due the the involved quasi-Newton mechanism, this comparison is not really relevant for UOBSQA and UOBDQA, except that in both cases p is bounded below by n.)

4.2 Performance of PSDFO

We now concentrate on PSDFO and analyze the complete results we obtained with this package. These results are reported in Table 3 and they were obtained with a PC Pentium 4 2.00GHz (900 MB Ram) running Linux. The three measures we collected with both PSDFO(I) and PSDFO(G) are the number of function evaluations, the number of iterations and the CPU time required for solving the problems of Table 1.

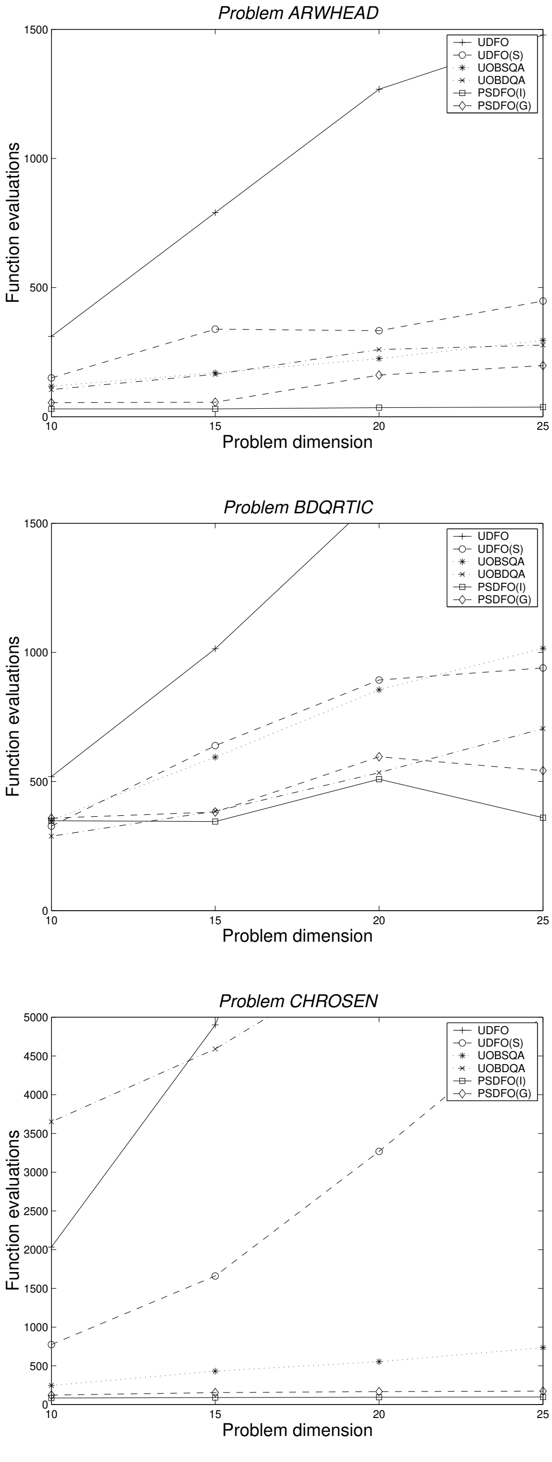

Figure 1: Number of function evaluations required by some DFO solvers for problems `ARWHEAD`, `BDQRTIC` and `CHROSEN`.

Problem	Measure	$n = 10$		$n = 20$		$n = 50$		$n = 100$		$n = 200$	
		I	G	I	G	I	G	I	G	I	G
ARWHEAD	evaluations	30	54	35	161	34	226	43	925	40	1624
	iterations	18	18	19	19	22	22	26	26	24	24
	CPU time	0.02	0.02	0.04	0.04	0.11	0.12	0.49	0.49	2.96	3.47
	groups	-	1.5	-	7.0	-	9.0	-	34.2	-	66.3
BDQRTIC	evaluations	348	358	509	596	627	1275	343	2263	336	5786
	iterations	214	214	408	408	567	561	295	295	279	279
	CPU time	1.68	1.75	8.33	4.68	33.58	33.60	37.74	39.23	118.80	117.50
	groups	-	0.05	-	0.2	-	1.2	-	6.5	-	19.6
BDVALUE	evaluations	46	57	45	73	44	72	42	79	27	42
	iterations	28	28	24	24	23	23	20	20	9	9
	CPU time	0.05	0.04	0.03	0.09	0.27	0.26	0.71	0.72	1.54	1.60
	groups	-	0.5	-	1.4	-	1.4	-	2.1	-	2.0
BROYDN3D	evaluations	53	70	56	84	68	144	153	342	123	278
	iterations	38	38	40	40	42	42	116	116	84	84
	CPU time	0.05	0.06	0.13	0.14	0.48	0.49	2.83	3.98	16.82	17.01
	groups	-	0.5	-	0.8	-	2.1	-	1.8	-	2.2
CHROSEN	evaluations	85	123	95	168	143	262	177	313	226	396
	iterations	63	63	67	67	99	99	111	111	141	141
	CPU time	0.03	0.05	0.11	0.11	0.57	0.58	2.50	2.54	22.67	21.58
	groups	-	0.8	-	1.4	-	1.6	-	1.7	-	1.7
CRAGGLVY	evaluations	392	455	390	530	525	878	713	1228	1040	1841
	iterations	327	327	334	334	424	424	562	562	799	799
	CPU time	0.55	0.70	1.73	1.73	6.13	6.14	23.58	23.10	160.43	159.58
	groups	-	0.3	-	0.5	-	1.0	-	1.1	-	1.3
DQDRTIC	evaluations	191	289	337	726	342	747	274	834	484	775
	iterations	151	162	233	275	212	212	170	226	283	204
	CPU time	0.21	0.26	0.93	1.18	2.20	2.59	6.59	6.78	49.27	43.98
	groups	-	0.6	-	2.4	-	2.4	-	2.6	-	2.7
EXTPOWELLSG	evaluations	270	276	256	308	401	525	504	772	844	1319
	iterations	225	225	213	213	354	354	441	441	746	746
	CPU time	0.37	0.41	0.85	0.75	3.70	3.71	14.42	14.38	132.52	132.30
	groups	-	0.05	-	0.3	-	0.4	-	0.6	-	0.7
EXTROSNB	evaluations	60	108	103	571	113	1553	205	6575	168	10864
	iterations	46	46	70	70	78	78	135	135	112	112
	CPU time	0.04	0.04	0.11	0.11	0.42	0.43	3.08	3.46	17.98	20.47
	groups	-	1.2	-	7.0	-	18.9	-	47.7	-	96.0
GENHUMPS	evaluations	124	(step)	212	228	185	(step)	880	(step)	198	338
	iterations	97		152	92	126		536		124	123
	CPU time	0.07		0.26	0.13	0.73		12.37		16.00	18.05
	groups	-		-	1.4	-		-		-	1.7
LIARWHD	evaluations	45	95	58	173	63	589	67	1345	93	(step)
	iterations	31	31	45	45	46	46	46	46	66	-
	CPU time	0.03	0.03	0.05	0.09	0.23	0.26	0.94	1.01	10.20	-
	groups	-	1.8	-	2.7	-	11.7	-	28.1	-	-
MOREBV	evaluations	32	40	32	41	34	49	37	65	23	34
	iterations	15	15	16	16	15	15	17	17	6	6
	CPU time	0.03	0.04	0.07	0.08	0.18	0.18	0.60	0.61	1.11	1.11
	groups	-	0.6	-	0.6	-	1.1	-	1.8	-	1.8
ROSENBR	evaluations	187	321	418	759	1158	2057	2502	4420	6728	11811
	iterations	136	136	276	276	738	738	1553	1553	4005	4070
	CPU time	0.13	0.13	0.53	0.56	4.78	4.94	40.32	33.3	707.91	731.69
	groups	-	1.3	-	1.7	-	1.8	-	1.8	-	1.9
SCHMVETT	evaluations	61	75	51	82	65	144	89	191	150	285
	iterations	44	44	33	33	41	41	62	62	97	97
	CPU time	0.09	0.10	0.12	0.13	0.46	0.47	0.78	0.87	21.37	21.49
	groups	-	0.4	-	1.1	-	2.2	-	1.9	-	1.8
WOODS	evaluations	963	1080	1046	1183	477	518	671	758	1366	1714
	iterations	840	840	933	933	416	416	585	585	1044	1044
	CPU time	1.53	1.56	2.54	2.91	3.46	3.49	18.05	18.24	192.22	192.72
	groups	-	0.2	-	0.2	-	0.2	-	0.3	-	0.6

Table 3: Further results for PSDFO.

We first examine the performance of PSDFO(I), that is the version for which it is assumed that the functions f_i ($i = 1, \ldots, M$) are accessible individually. As expected, the number of function evaluations does not increase much when higher dimensions are considered, except for some difficult problems like ROSENBR. This is illustrated on Figure 2, where the evolution of the number of function evaluations required by PSDFO(I) for the various sizes of the problems is displayed (note that these numbers are normalized in the sense that, for a given line of Table 3, the various values it contains are divided by the average of these values). For some problems (*e.g.* EXTROSNB and GENHUMPS), interme-

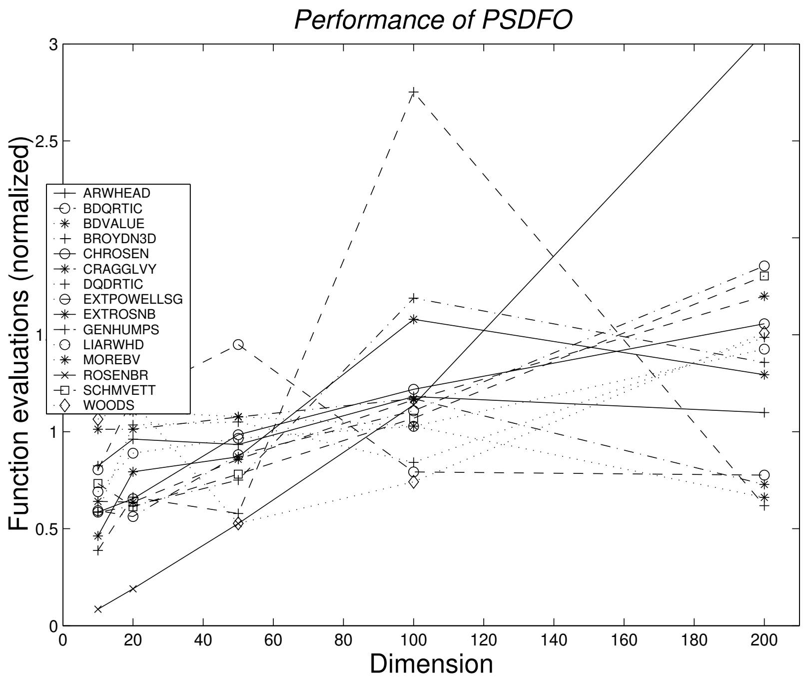

Figure 2: Performance of PSDFO(I) in terms of function evaluations when the size of problems increases.

diate dimension instances require more computational effort than for the larger instances we considered.

Let us now consider the results obtained with PSDFO(G). As could be expected, this version of the algorithm requires more function evaluations than PSDFO(I), which is a clear illustration of the advantages of the previous framework. However, it seems that the grouping strategy described in Section 3 allows to limit the increase in the computational cost. To show this, we reported a further measure regarding the results obtained with PSDFO(G), namely the average number of groups (per iteration) constructed by the algorithm when geometry improvements occur. For some problems, this number varies much from $n = 10$ to $n = 200$ while for others it remains between 0.5 and 2.0: this is due to the fact that some problems (like ARWHEAD or EXTROSNB) have individual functions that can *never* be grouped because they all depend on (at least) one common variable (for ARWHEAD, function f_i depends on x_i and x_{10} while for EXTROSNB it depends on x_1 and x_i) while other problems (like BDVALUE or WOODS) involve individual functions that can be grouped easily. As a consequence, solving the former problems with PSDFO(G) requires much more function evaluations (with the same number of iterations as that observed with PSDFO(I)) while the latter can be solved at a cost which is only slightly higher than with PSDFO(I). Note that for three problem instances PSDFO(G) terminated prematurely because the step s_k fell below the tolerance (1.0^{-12}), indicating that the algorithm was stalling.

Finally, we should mention that while PSDFO is clearly better than UDFO in terms of computational effort, the latter remains advantageous from the point of view of storage issues, should this become an issue. Indeed, since PSDFO constructs models for the M individual functions forming the objective, it must manage M interpolation sets, M bases of Newton fundamental polynomials and M models $m_{i,k}$. Despite the fact that each of these components is smaller than that corresponding to the single model m_k generated by UDFO (since $n_i < n$ for each $i \in \{1, \dots, M\}$), the need to consider M such spaces in PSDFO makes it more demanding in terms of storage.

5 Conclusions

We have presented a new algorithm for solving unconstrained derivative-free optimization problems whose objective function is partially separable. This algorithm proves to be very efficient in terms of function evaluations, the latter being the most important criterion for the assessment of derivative-free methods.

We believe that our algorithm can be extended further and we would like to mention here some possible refinements. The first one is related to Powell's methods and consists in using only the dominant parts of the Hessian matrix. Indeed, it seems that introducing "artificial sparsity" and ignoring some of the entries of the Hessian that may be considered as being negligible is likely to yield the same improvements as those reported by Powell [24]. A more promising approach would be to combine the least Frobenius norm update of Powell within our partially separable scheme and to apply Powell's technique at the level of individual functions and their models. Furthermore, we could use a criterion like that of Conn, Gould, Sartenaer and Toint [5] for selecting "negligible" individual functions whose model might be left unchanged in a larger region. Their criterion suggests to define the index set of negligible functions as follows:

$$\big\{i \in \{1, \dots, M\} : |m_{i,k}(x^{(k)}) - m_{i,k}(x^{(k)} + s^{(k)})| \leq \frac{\mu}{M} |m_k(x^{(k)}) - m_k(x^{(k)} + s^{(k)})| \big\},$$

where $m_{i,k}$ and m_k denote the model of the individual function f_i and that of the global objective f respectively while μ is a parameter such that $0 < \mu < 1$.

Also note that the current version of the algorithm assumes that we can only evaluate the f_i's at any point x but we might have a more complete knowledge concerning the gradient and/or the Hessian of some individual functions $f_i(\cdot)$. The use of all available derivatives is straightforward in (10). Combination with quasi-Newton methods is also possible for the element functions whose gradient but not the Hessian is available.

References

[1] C. Audet and J. E. Dennis Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[2] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20(1):187–209, 1983.

[3] B. Colson and Ph. L. Toint. Exploiting band structure in unconstrained optimization without derivatives. *Optimization and Engineering*, 2(4):399–412, 2001.

[4] B. Colson and Ph. L. Toint. A derivative-free algorithm for sparse unconstrained optimization problems. In A. H. Siddiqi and M. Kocvara, editors, *Trends in Industrial and Applied Mathematics*, volume 72 of *Applied Optimization*, pages 131–147. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[5] A. R. Conn, N. I. M. Gould, A. Sartenaer, and Ph. L. Toint. Convergence properties of minimization algorithms for convex constraints using a structured trust region. *SIAM Journal on Optimization*, 6(4):1059–1086, 1996.

[6] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Springer Verlag, Heidelberg, New York, USA, 1992.

[7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. SIAM Publications, Philadelphia, Pennsylvania, USA, 2000.

[8] 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 University Press, Cambridge, England, 1997.

[9] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming, Series B*, 79(3):397–414, 1997.

[10] A. R. Conn, K. Scheinberg, and L. N. Vicente. Error estimates and poisedness in multivariate polynomial interpolation. Pré-Publicações do Departamento de Matemática, Universidade de Coimbra, Preprint Number 03-09, 2003.

[11] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Maths Applics*, 13:117–119, 1974.

[12] C. de Boor and A. Ron. Computational aspects of polynomial interpolation in several variables. *Mathematics of Computation*, 58(198):705–727, 1992.

[13] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec), a Constrained and Unconstrained Testing Environment, revisited. *Transactions of the ACM on Mathematical Software*, 2003. To appear. See also `http://cuter.rl.ac.uk/cuter-www/`.

[14] A. Griewank and P. L. Toint. Local convergence analysis of partitioned quasi-Newton updates. *Numerische Mathematik*, 39:429–448, 1982.

[15] A. Griewank and P. L. Toint. On the unconstrained optimization of partially separable objective functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312, London, England, 1982. Academic Press.

[16] A. Griewank and P. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:119–137, 1982.

[17] L. Grippo, F. Lampariello, and S. Lucidi. Global convergence and stabilization of unconstrained minimization methods without derivatives. *Journal of Optimization Theory and Applications*, 56:385–406, 1988.

[18] R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methods: then and now. *Journal of Computational and Applied Mathematics*, 124(1–2):191–207, December 2000.

[19] S. Lucidi and M. Sciandrone. Numerical results for unconstrained optimization without derivatives. In F. Giannessi and G. Di Pillo, editors, *Nonlinear Optimization and Applications*, pages 261–269. Plenum Publishing, New York, NY, USA, 1995.

[20] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 17:155–162, 1964.

[21] M. J. D. Powell. A method for minimizing a sum of squares of nonlinear functions without calculating derivatives. *Computer Journal*, 7:303–307, 1965.

[22] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114. Gordon and Breach, London, England, 1970.

[23] M. J. D. Powell. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. Technical Report NA2002/08, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2002.

[24] M. J. D. Powell. On trust region methods for unconstrained minimization without derivatives. Technical Report NA2002/02, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2002.

[25] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic interpolation. *Mathematical Programming*, 92:555–582, 2002.

[26] M. J. D. Powell. On the use of quadratic models in unconstrained minimization without derivatives. Technical Report NA2003/03, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2003.

[27] T. Sauer. Computational aspects of multivariate polynomial interpolation. *Advances in Computational Mathematics*, 3:219–238, 1995.

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

[29] Ph. L. Toint. Test problems for partially separable optimization and results for the routine PSPMIN. Technical Report 83/4, Department of Mathematics, FUNDP, Namur, Belgium, 1983.

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