Using Partial Separability of Functions in Generating Set Search Methods for Unconstrained Optimisation*

Lennart Frimannslund† Trond Steihaug‡

March 21, 2006

Abstract

Generating set Search Methods (GSS), a class of derivative-free methods for unconstrained optimisation, are in general robust but converge slowly. It has been shown that the performance of these methods can be enhanced by utilising accumulated information about the objective function as well as a priori knowledge such as partial separability.

This paper introduces a notion of partial separability which is not dependent on differentiability. We present a provably convergent method which extends and enhances a previously published GSS method. Whereas the old method for two times continuously differentiable functions takes advantage of Hessian sparsity, the new method takes advantage of the separability properties of partially separable functions with full Hessians as well. If the Hessian is undefined we show a similar extension, compared with the old method. In addition, we introduce some new theoretical results and discuss variants of the method.

Keywords: Unconstrained optimisation, derivative-free optimisation, noisy optimisation, generating set search, pattern search, separability, sparsity.

1 Introduction

Direct search methods have been an active area of research in recent years. One class of direct search methods is Generating Set Search (GSS). For a comprehensive introduction to these methods, see [7], or papers among e.g. [8, 14, 2, 9]. GSS methods in their simplest form use only function values to determine the minimum

*This work was supported by the Norwegian Research Council (NFR).

†Department of Informatics, University of Bergen, Box 7800, N-5020 Bergen, Norway. E-mail: lennart.frimannslund@ii.uib.no

‡Department of Informatics, University of Bergen. E-mail: trond.steihaug@ii.uib.no

of a function. It has been shown that taking derivative information into account [1], and particularly curvature information [3, 6] can lead to more effective methods. In addition, a priori knowledge about separability of the objective function is also helpful in designing more effective methods, and methods that can be applied to problems of relatively large scale [12, 5]. This paper extends the work and method presented in [5], introducing a provably convergent variant of that method, generalises it to a wider class of partially separable functions and presents a more thorough theoretical foundation than what as been done before. This paper is organised as follows. In section 2 we present the algorithm of [6], and the extension outlined in [5], present modifications and extensions, and offer some new theoretical results. Section 3 presents results from numerical testing, and section 4 offers some concluding remarks.

2 A GSS Method using Curvature Information

A Basic GSS Algorithm GSS algorithms draw their name from the fact that they search along the members of a generating set. A generating set is a set of vectors

$$\mathcal{G} = \bigcup_{i=1}^{r} \{v_i\},$$

such that for any $x \in \mathbb{R}^n$ we have,

$$x = \sum_{i=1}^{r} c_i v_i, \quad \text{where } c_i \geq 0,\ i = 1, \ldots, r.$$

This can be achieved for $r \geq n+1$, depending on the vectors in $\mathcal{G}$. We will, for the most part be concerned with sets of the form

$$\mathcal{G} = \bigcup_{i=1}^{n} \{q_i, -q_i\}, \tag{1}$$

that is, the positive and negative of n vectors, which we in addition define to be orthonormal. We will call $\mathcal{G}$ our *search basis*. A simple method based on this set we will call compass search. The step length δ_i is associated with both q_i and $-q_i$. Pseudo code for compass search is listed in Figure 1. An example of how the method can work in $\mathbb{R}^2$ is given in Figure 2. In the figure, $\mathcal{G}$ consists of vectors at a 45 degree angle to the coordinate axes. The search starts at the black node/point, marked 0. First the method searches down and to the right, finds a better function value and steps. This is marked by a grey node. Then it searches down and to the left, and steps. It then searches up and to the left, but does not find a lower function value so it does not update x. This is indicated by a white node (marked

```
Compass Search
    Given $x,\ \mathcal{G},\ \delta_i \geq 0,\ i = 1, \ldots, n$.
    Repeat until convergence:
        Select next vector from $\mathcal{G}$, say, $q_i$.
        If $f(x + \delta_i q_i) < f(x)$,
            Take $x + \delta_i q_i$ as new point.
        Update $\delta_i$ according to some rule in accordance with convergence theory.
    end.
```

Figure 1: Compass search code.

Γ in the figure). Then it searches down and to the right again and steps, and so on. The step lengths remain the same throughout in this example. Compass search be modified in many ways. For instance, what is the "next" direction in $\mathcal{G}$ can be defined in more than one way, many rules of updating δ_i can be applied, and the set $\mathcal{G}$ can be updated, although convergence theory may place some restrictions on these updates (see e.g. [7]). The method of [6] uses a variant of compass search, and by adaptively choosing the order in which the search directions are selected, it is able to compute matrix of curvature information (which is an approximation to the Hessian if the function is two times continuously differentiable) and replaces $\mathcal{G}$ with the positive and negative of the eigenvectors of this matrix. In [6] numerical results show that updating $\mathcal{G}$ this way can reduce the number of function evaluations needed to converge dramatically, compared to compass search with the positive and negative of the coordinate vectors as its search basis $\mathcal{G}$. Curvature information can be obtained by the formula

$$\frac{f(x + hq_i + kq_j) - f(x + hq_i) - f(x + kq_j) + f(x)}{hk} = q_i^T \nabla^2 f(x{+}tq_i{+}sq_j) q_j, \quad s, t \in [0, 1], \tag{2}$$

where the equation holds for two times continuously differentiable functions. Let C_Q be a symmetric $n \times n$ matrix with the result of formula (2) as element (i, j), with x, h and k not necessarily being the same for each (i, j)-pair. To see that such information is obtainable in the context of compass search, consider again Figure 2. If we look at the figure, we see that we often have four points making up a rectangle, for instance the points marked 0, 1, 2 and Γ. Given four such points one can compute the appropriate element of C_Q with formula (2), since the numerator of the left hand-side of (2) is made up of the function values of four points in a rectangle. In higher dimension than 2 one can construct such rectangles by performing extra function evaluations if this is needed. The algorithm of [6] does this by shuffling the order in which the search directions are selected. Specifically,

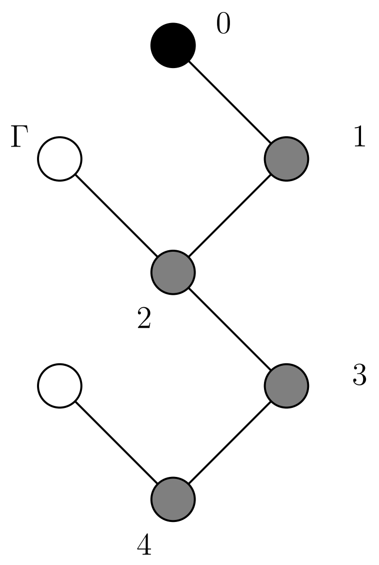

Figure 2: Compass search in $\mathbb{R}^2$ along vectors 45° to the coordinate axes.

it partitions the available directions into triples T of the form

$$T: \ \pm q_i \quad q_j \quad \mp q_i \ \text{ or } T: \ \pm q_i \quad -q_j \quad \mp q_i. \tag{3}$$

In Figure 2, if we define q_1 as down and to the right and q_2 as down and to the left, the directions are searched in the order

$$q_1 \quad q_2 \quad -q_1 \quad -q_2 \ ,$$

that is, one triplet of the form (3) and one leftover direction. If the search along the first two directions of such a triplet is successful, then one will obtain the required rectangle regardless of whether or not the search along the third direction is successful. In Figure 2 the search in the third direction is unsuccessful, but as mentioned the points marked 0, 1, 2 and Γ make up the required rectangle. If search fails along either of the first two directions then one needs to compute the fourth point of the rectangle separately. Three directions per element and $2n$ directions to choose from enables computation of the order of $2n/3$ elements of C_Q per iteration, although one at the most can obtain two elements in the same row or column per iteration. Diagonal C_Q elements can be computed from constellations like the points marked Γ, 2, 3, that is, three equally spaced points along a straight line.

Alternatively, one can arrange the search directions into pairs rather than triples, and compute an extra point for each rectangle. This is depicted in Figure 3. In Figure 3 the point marked by a cross needs to be computed separately in each case, and can be stepped to if it produces a lower function value, although this is not

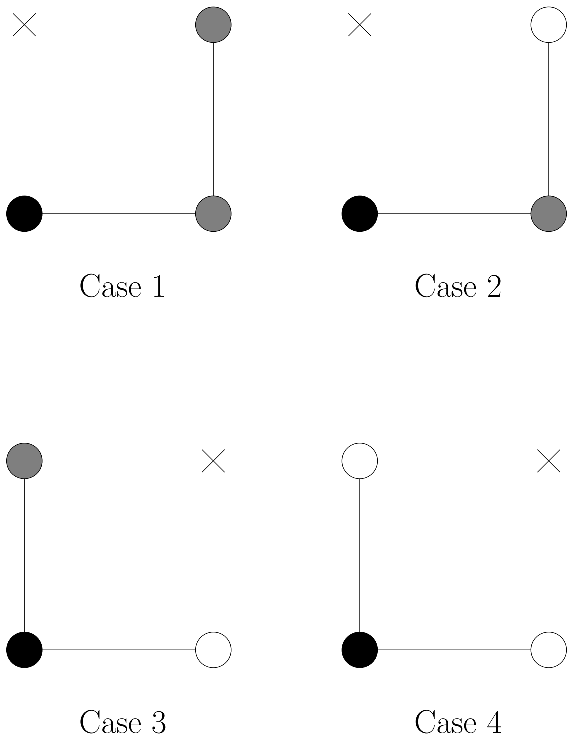

Figure 3: The four possible outcomes of successive searches to the east and north. A grey node signifies a step which has been taken, a white node signifies a step not taken. The search starts at the black node in each case.

done in [5, 6]. For instance, if $n = 4$ and one wants to compute $(C_Q)_{21}$, $(C_Q)_{31}$, $(C_Q)_{24}$, and $(C_Q)_{34}$, then one can order the directions

$$q_1 \quad q_2 \quad -q_1 \quad q_3 \quad -q_2 \quad q_4 \quad -q_3 \quad -q_4 \ .$$

Here the search along q_1 and q_2 provides us with $(C_Q)_{21}$, $-q_1$ and q_3 provides us with $(C_Q)_{31}$, and so on. This way we can compute the order of n elements per iteration. More complex schemes for obtaining curvature information can be devised, but we restrict ourselves to pairs and triples in this paper. Once the matrix C_Q is complete, the algorithm computes the matrix

$$C = QC_QQ^T, \tag{4}$$

which contains curvature information with respect to the standard coordinate system.

In general one can collect curvature information even if the members of $\mathcal{G}$ are not orthogonal. Let P be a matrix with linearly independent columns of unit length, not necessarily orthogonal, and denote its ith column by p_i. Let

$$f(x) = c + b^Tx + \frac{1}{2}x^THx,$$

and let the symmetric matrix C_P have its entry (i, j), $i > j$, computed by

$$(C_P)_{ij} = \frac{f(x + hp_i + kp_j) - f(x + hp_i) - f(x + kp_j) + f(x)}{hk} = p_i^T\nabla^2 f(x^{ij})p_j, \tag{5}$$

where the point x^{ij} varies with i and j. Then, since $\nabla^2 f$ is equal to H for all x, we have that

$$H = P^{-T}C_PP^{-1}.$$

To see this, observe that since $(C_P)_{ij}$ is the result of a difference computation along p_i and p_j, the construction of the matrix C_P is equivalent to finite difference computation of the Hessian, along the coordinate vectors, of the function

$$g(w) = c + b^TPw + \frac{1}{2}w^TP^THPw.$$

Here, we have

$$w = P^{-1}x,$$

so the Hessian of g satisfies

$$C_P = \nabla^2 g(w) = P^THP,$$

and the relation follows. For non-quadratic functions we cannot expect to recover an exact Hessian if the points x^{ij} in (5) are different for different i and j, but can recover the matrix

$$C = P^{-T}C_PP^{-1}. \tag{6}$$

The relationship between C and $\nabla^2 f$ is addressed in the following Lemma, which is a slight variation of Lemma 2 in [6]. Let C_P be the matrix with entry (i,j) as in (5), and let C be the matrix (6). Let

$$N = \bigcup_{\forall i,j} x^{ij},$$

where x^{ij} are the same as in equation (5). Let

$$\delta = \max_{x,y \in N} \|x - y\|,$$

and let

$$\mathcal{N} = \left\{ x \in \mathbb{R}^n \,\middle|\, \max_{y \in N} \|x - y\| \le \delta \right\}.$$

Lemma 1 *Assume $f : \mathbb{R}^n \mapsto \mathbb{R}$ is two times continuously differentiable, and that $\nabla^2 f$ satisfies*

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \le L\|x - y\|, \quad \forall x, y \in \mathcal{N}. \tag{7}$$

Then, C satisfies

$$\|C - \nabla^2 f(\widetilde{x})\| \le \|P^{-1}\|^2 nL\delta, \tag{8}$$

where n is the number of variables, and $\widetilde{x} \in \mathcal{N}$.

Proof. Consider the matrix

$$C_P - P^T \nabla^2 f(\widetilde{x}) P. \tag{9}$$

Element (i,j) of this matrix is equal to

$$p_i^T \big(\nabla^2 f(x^{ij}) - \nabla^2 f(\widetilde{x})\big) p_j,$$

so by the relation $\|AB\| \le \|A\|\,\|B\|$ and (7) we can write

$$|p_i^T \big(\nabla^2 f(x^{ij}) - \nabla^2 f(\widetilde{x})\big) p_j| \le \|p_i\| \; L \; \|x^{ij} - \widetilde{x}\| \; \|p_j\| \le L\delta,$$

where the last two inequalities hold since $\widetilde{x} \in \mathcal{N}$. The Frobenius norm of the matrix (9) is the square root of the sum of the squares of its n^2 elements, so we have

$$\|C_P - P^T \nabla^2 f(\widetilde{x}) P\|_F^2 \le \sum_{i=1}^{n} \sum_{j=1}^{n} L^2 \delta^2 = (nL\delta)^2.$$

By applying $\|AB\| \le \|A\|\,\|B\|$, we can now write

$$\begin{aligned} \|P^{-T}\big(C_P - P^T \nabla^2 f(\widetilde{x}) P\big) P^{-1}\| &= \|C - \nabla^2 f(\widetilde{x})\| \\ &\le \|P^{-T}\| \; \|C_P - P^T \nabla^2 f(\widetilde{x}) P\|_F \; \|P^{-1}\| \\ &\le \|P^{-1}\|^2 nL\delta. \end{aligned}$$

□

So, the error in the matrix C compared to the Hessian increases linearly with δ.

For the rest of the paper we assume, for simplicity, that $\mathcal{G}$ consists of the positive and negative of n mutually orthogonal vectors, $q_1, \ldots, q_n$ and that Q is the matrix with q_i as its ith column.

2.1 Extension to partially Separable Functions

An Extended Definition of Sparsity When a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is two times continuously differentiable, one can say that if its Hessian is sparse then the function partially separable. That is, it can be written as a sum of element functions,

$$f = \sum_{i=1}^{\nu} f_i \tag{10}$$

where, with the index sets χ_i, signifying the elements of x that f_i depends on,

$$\chi_i \subset \{1, 2, \dots, n\}, \; i = 1, \dots, \nu,$$

we have

$$f_i : \mathbb{R}^{|\chi_i|} \mapsto \mathbb{R}.$$

In other words, f is the sum of ν element functions, which each depends on less than n variables. (For a proof of this, see e.g. Theorem 9.3 of [11].) Let us extend the notion of a sparse Hessian and consequent separability to non-differentiable functions.

Given the standard coordinate vectors $e_1, \dots, e_n$, define the undirected *covariation graph* as the graph $G(V, E)$ with n nodes, one node for each of the n coordinate vectors. Furthermore, let there be an edge between node i and j if and only if there exist x, h and k such that

$$f(x + he_i + ke_j) - f(x + he_i) - f(x + ke_j) + f(x) \neq 0.$$

Define the adjacency matrix of a graph as the $|V| \times |V|$ matrix where element (i, j) is 1 if there is an edge from node i to j, and zero otherwise. Since the graph is undirected, this matrix is symmetric. Now, if the function f is two times continuously differentiable then the structure of the adjacency matrix corresponding to the graph G has the same structure as the Hessian matrix $\nabla^2 f$. If the Hessian is not defined, then the adjacency matrix has the same structure as the matrix C_P of (5), with $P = I$.

Lemma 2 *If the covariation graph corresponding to a function f is sparse, then f is partially separable.*

Proof. If G is not complete, then there exists i, j for which

$$f(x + he_i + ke_j) - f(x + he_i) - f(x + ke_j) + f(x) = 0.$$

Without loss of generality, assume $n = 2$. Then we have

$$f(x_1 + h, x_2 + k) - f(x_1 + h, x_2) - f(x_1, x_2 + k) + f(x_1, x_2) = 0.$$

Now, let $x_1 = x_2 = 0$ and let h and k be the independent variables. This gives us

$$f(h, k) - f(h, 0) - f(0, k) + f(0, 0) = 0,$$

which can be written

$$f(h,k) = f(h,0) + f(0,k) - f(0,0).$$

Now we can define, for instance $f_1(h) = f(h,0)$ and $f_2(k) = f(0,k) - f(0,0)$, and we have that f is if the form (10). $\square$
A graph $G(V,E)$ is sparse if there exist nodes i and j so that there is no edge from i to j in the edge set E.

Note that we do not require that f is differentiable here. If the objective function corresponds to a sparse covariation graph and sparse adjacency matrix with, say, ρ nonzero elements in the lower triangle, then we can obtain the matrix C more quickly than if the covariation graph is complete. In the following we for simplicity use the word "Hessian" when discussing the structure of the adjacency matrix, although that f is two times continuously differentiable is not a requirement for our method to be applicable.

Application to the Optimisation Method If the search directions of the method are ordered in doubles as outlined earlier, then computing a full Hessian of $n(n+1)/2$ elements at $O(n)$ elements per iteration takes $O(n)$ iterations, whereas computing ρ elements at $O(n)$ elements per iteration takes $O(\frac{\rho}{n})$ iterations, which in the case of for instance a tridiagonal Hessian, where $\rho = O(n)$ means that we can obtain C in a constant number of iterations, independent of n. Advantages of this are that:

- The method can update the search basis more often, which is important on functions like the Rosenbrock function.
- The method is effective for larger values of n than the non-sparse method since the latter for large n sometimes does not rotate at all before terminating, reducing it to compass search.
- The computed curvature information matrix will be more accurate compared to the Hessian (if it exists), since they share the same identical zeros, and since it is likely to have been computed over a smaller region.

In [5], the method of the previous section was extended to take advantage of the partial separability property of functions with sparse Hessians. Sparsity must be seen as relative to the coordinate system used, even if the second and cross derivatives vanish at many positions in the Hessian, this does not mean that the directional second and cross derivatives obtained when applying (2) along arbitrary directions will be zero as often, if at all. To overcome this difficulty, one can first reformulate the equation (4) to

$$Q^T C Q = C_Q. \tag{11}$$

Equation (11) can be reformulated using Kronecker products. Kronecker products are useful in light of the relation

$$AXB = C \Leftrightarrow (B^T \otimes A)\mathbf{vec}(X) = \mathbf{vec}(C). \tag{12}$$

The operation $\mathbf{vec}(X)$ stacks the columns of the matrix X on top of each other in a vector. By using (12), (11) can be rewritten as

$$(Q^T \otimes Q^T)\mathbf{vec}(C) = \mathbf{vec}(C_Q). \tag{13}$$

Both C and C_Q are symmetric, so we can eliminate all the strictly upper triangular elements of C from the equation system. (We can of course alternatively eliminate elements from the lower triangle of C instead.) This we do by first adding the columns in $(Q^T \otimes Q^T)$ corresponding to the upper-triangular elements of C to the columns corresponding to its lower-triangular elements, and then delete the former columns. After this operation we must delete the same number of rows, specifically the rows corresponding to the upper or lower triangle of C_Q. Otherwise, the coefficient matrix will be singular. To see this, consider a simple example with $n = 2$. Initially, we have equation (11), which looks like

$$\begin{bmatrix} Q_{11} & Q_{21} \\ Q_{12} & Q_{22} \end{bmatrix} \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} = \begin{bmatrix} (C_Q)_{11} & (C_Q)_{12} \\ (C_Q)_{21} & (C_Q)_{22} \end{bmatrix}. \tag{14}$$

Equation (13) becomes

$$\begin{bmatrix} Q_{11}Q_{11} & Q_{11}Q_{21} & Q_{21}Q_{11} & Q_{21}Q_{21} \\ Q_{11}Q_{12} & Q_{11}Q_{22} & Q_{12}Q_{21} & Q_{21}Q_{22} \\ Q_{11}Q_{12} & Q_{12}Q_{21} & Q_{11}Q_{22} & Q_{21}Q_{22} \\ Q_{12}Q_{12} & Q_{12}Q_{22} & Q_{12}Q_{22} & Q_{22}Q_{22} \end{bmatrix} \begin{bmatrix} C_{11} \\ C_{12} \\ C_{21} \\ C_{22} \end{bmatrix} = \begin{bmatrix} (C_Q)_{11} \\ (C_Q)_{12} \\ (C_Q)_{21} \\ (C_Q)_{22} \end{bmatrix}. \tag{15}$$

We now want to eliminate element C_{12}, so we add column two to column three in (15) and delete column two, giving us the overdetermined system

$$\begin{bmatrix} Q_{11}Q_{11} & Q_{11}Q_{21} + Q_{21}Q_{11} & Q_{21}Q_{21} \\ Q_{11}Q_{12} & Q_{11}Q_{22} + Q_{12}Q_{21} & Q_{21}Q_{22} \\ Q_{11}Q_{12} & Q_{12}Q_{21} + Q_{11}Q_{22} & Q_{21}Q_{22} \\ Q_{12}Q_{12} & Q_{12}Q_{22} + Q_{12}Q_{22} & Q_{22}Q_{22} \end{bmatrix} \begin{bmatrix} C_{11} \\ C_{21} \\ C_{22} \end{bmatrix} = \begin{bmatrix} (C_Q)_{11} \\ (C_Q)_{12} \\ (C_Q)_{21} \\ (C_Q)_{22} \end{bmatrix}. \tag{16}$$

In (16) we can see that rows two and three are the same, so we must eliminate one of them to obtain a square system with full rank. Let us eliminate the row corresponding to $(C_Q)_{12}$, and we finally get

$$\begin{bmatrix} Q_{11}Q_{11} & Q_{11}Q_{21} + Q_{21}Q_{11} & Q_{21}Q_{21} \\ Q_{11}Q_{12} & Q_{12}Q_{21} + Q_{11}Q_{22} & Q_{21}Q_{22} \\ Q_{12}Q_{12} & Q_{12}Q_{22} + Q_{12}Q_{22} & Q_{22}Q_{22} \end{bmatrix} \begin{bmatrix} C_{11} \\ C_{21} \\ C_{22} \end{bmatrix} = \begin{bmatrix} (C_Q)_{11} \\ (C_Q)_{21} \\ (C_Q)_{22} \end{bmatrix}. \tag{17}$$

A similar scheme for determining the elements of an unknown sparse matrix was used in [4]. When C is assumed to be sparse we can eliminate the columns in the equation system corresponding to elements of C which we know are zero. We then have the option of eliminating up to the same number of rows. The more rows we eliminate, the smaller the right-hand side, and the faster we can obtain the matrix C in the context of our optimisation method. The effect on the final solution C of removing rows depends on both the right-hand side we end up with, and on the conditioning of the resulting coefficient matrix.

Existence of Non-singular Coefficient Matrix There always exists a non-singular $\rho \times \rho$ coefficient matrix resulting from the process described above. Given the $n^2 \times n^2$ equation system

$$(Q^T \otimes Q^T)\mathbf{vec}(C) = \mathbf{vec}(C_Q), \tag{18}$$

we first add together the columns in the coefficient matrix corresponding to C_{ij} and C_{ji}, then delete columns corresponding C_{ij} where $i < j$ and $C_{ij} = 0$. This can be done by right-multiplying the matrix $(Q^T \otimes Q^T)$ with a matrix P_c. For instance, if C is a symmetric tridiagonal 3×3 matrix, that is

$$C = \begin{bmatrix} \times & \times & \\ \times & \times & \times \\ & \times & \times \end{bmatrix},$$

then the matrix which adds together the required columns of $(Q^T \otimes Q^T)$ and deletes the columns corresponding to upper triangular elements of C, as well as the zero element in the lower triangle of C is:

$$P_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{19}$$

Since columns in $(Q^T \otimes Q^T)$ are deleted, we must alter $\mathbf{vec}(C)$ in (18) as well for the equation to make sense. Define $\overline{\mathbf{vec}}()$ as the operator that stacks the nonzero elements of the lower triangle of C in a vector $\overline{\mathbf{vec}}(C)$. This compensates for the columns P_c removes. We can then write the reduced version of (18) as

$$(Q^T \otimes Q^T)P_c\overline{\mathbf{vec}}(C) = \mathbf{vec}(C_Q). \tag{20}$$

The equation system (20) is over-determined, with dimension $n^2 \times \rho$. Let P_r be an $\rho \times n^2$ matrix which selects ρ rows from an $n^2 \times \rho$ matrix, that is, let P_r consist of zeros except for one unity entry on each row. To reduce (20) to an $\rho \times \rho$ system, we left-multiply both sides of the equality sign with P_r, which finally gives us

$$P_r(Q^T \otimes Q^T)P_c\overline{\mathbf{vec}}(C) = P_r\mathbf{vec}(C_Q). \tag{21}$$

Lemma 3 *There exists at least one P_r such that the matrix $P_r(Q^T \otimes Q^T)P_c$ is invertible.*

Proof. The matrix $(Q^T \otimes Q^T)$ is orthogonal, and hence has full rank. Adding together columns like the matrix P_c does is an elementary column operation (which preserves rank), except for the fact that the columns corresponding to C_{ij} and C_{ji} are set equal. The matrix P_c deletes half of these columns, thus restoring full rank, as well as deleting additional columns corresponding to C_{ij} known to be zero. Consequently, the $n^2 \times \rho$ matrix

$$(Q^T \otimes Q^T)P_c$$

has rank ρ, and there is at least once selection of ρ rows which results in an $(\rho \times \rho)$ matrix of full rank. $\Box$

For simplicity, we introduce the notation

$$A = P_r(Q^T \otimes Q^T)P_c, \tag{22}$$

and

$$c_Q = P_r\mathbf{vec}(C_Q).$$

Construction of Nonsingular A It turns out not to be obvious which rows P_r should select when constructing A. For example, if C has a full diagonal, and P_r does not select rows in $(Q^T \otimes Q^T)$ corresponding to the diagonal of C_Q, then A is singular, and particularly,

$$A\overline{\mathbf{vec}}(I) = 0. \tag{23}$$

To see this, consider the left hand-side of (23), which using (22), can be written:

$$\begin{aligned} A\overline{\mathbf{vec}}(I) &= P_r(Q^T \otimes Q^T)P_c\overline{\mathbf{vec}}(I) \\ &= P_r(Q^T \otimes Q^T)\mathbf{vec}(I) \\ &= P_r\mathbf{vec}(Q^T I Q) \\ &= P_r\mathbf{vec}(I). \end{aligned}$$

Now the matrix corresponding to $\mathbf{vec}(I)$, the identity, is diagonal, but since P_r cuts all the rows corresponding to the diagonal, then

$$P_r\mathbf{vec}(I) = 0,$$

and we have shown that A is singular in this case. We can extend the result to:

Lemma 4 *Let E be a non-zero symmetric matrix with the same sparsity structure as C such that the vector $\mathbf{vec}(Q^T E Q)$ only has nonzero entries in positions which are cut by P_r. Then A is singular, and, specifically,*

$$A\overline{\mathbf{vec}}(E) = 0.$$

```
select_rows
Let $a_k$ be row $k$ of $(Q^T \otimes Q^T)P_c$.
Let $\mathcal{A} = \emptyset$.
For $i = 1$ to $\rho$,
    choose next $a_j \in \{a_1, a_2, \ldots, a_{n^2}\} \setminus \mathcal{A}$, so that the vectors in $\mathcal{A}$ and $a_j$ are linearly independent,
    set $\mathcal{A} = \mathcal{A} \bigcup \{a_j\}$ ,
end.
Set $A$ to be the matrix made up of the rows in $\mathcal{A}$, sorted appropriately.
```

Figure 4: Procedure for constructing nonsingular A.

We can always construct the matrix A in a brute-force fashion by starting with the matrix

$$(Q^T \otimes Q^T)P_c, \tag{24}$$

Which is an $n^2 \times \rho$ matrix of full rank. (It can easily be reduced to an $n(n+1)/2 \times \rho$ matrix of full rank, since we know that the rows corresponding to C_Q elements (i,j) and (j,i), say, are equal.) To reduce this to a $\rho \times \rho$ matrix of full rank we can use QR-factorisation (on the transpose of matrix (24)), which identifies which rows can be deleted. Alternatively, and without the need for storing a large matrix, one can build the matrix A row by row with a procedure like in Figure 4. The procedure constructs A one row at a time checking if the last added row is linearly independent from the previously added rows. This can be done by starting with A being zero and maintaining a QR-factorisation of A^T. That the procedure in Figure 4 always produces a non-singular matrix A regardless of how the for-loop is implemented is addressed in the following Lemma.

Lemma 5 *The procedure select_rows produces a matrix A which is nonsingular.*

Proof. Let $i < \rho$. Let A_i be an $i \times \rho$ matrix with i linearly independent rows picked from the rows of $(Q^T \otimes Q^T)P_c$, i.e. the result of select_rows after i iterations of the for-loop. Let a_j be row j of A_i, $j = 1, \ldots, i$. After iteration i we have

$$\operatorname{span}\{a_1, \ldots, a_i\} \subset \mathbb{R}^\rho.$$

Since the matrix $(Q^T \otimes Q^T)P_c$ has rank ρ, its rows span the entire space $\mathbb{R}^\rho$. Therefore, if $i \neq n$ there will always be a row in $(Q^T \otimes Q^T)P_c$ which, if projected into the space

$$\mathbb{R}^\rho \setminus \operatorname{span}\{a_1, \ldots, a_i\},$$

is nonzero. Consequently, given $i < \rho$ linearly independent rows, one can always find $i+1$ linearly independent rows, and repeat the process until A is complete. □

Whether or not select_rows will be effective depends on how the for-loop, and particularly the "choose next j" statement is implemented. The heuristic of [5] which often (but not always) produces nonsingular A-matrices, can be used for selecting the first candidate rows. The heuristic works by noting that including a specific row in A corresponds to computing a specific element of C_Q, say, $(C_Q)_{rs}$. The element $(C_Q)_{rs}$ is computed by searching along the search directions q_r and q_s, and the corresponding row in A is constructed from the same vectors. The heuristic suggests using the (q_r, q_s)-pairs which mimic the most the coordinate vectors which would have been used if $Q = I$, by the rule: For all $(i,j), i > j$, if $C_{ij} \neq 0$, then include the row in A constructed from q_r and q_s which satisfy

$$\arg\max_k (q_r)_k = i,$$

and

$$\arg\max_l (q_s)_l = j.$$

We now look at the effect of truncation errors in c_Q on C.

Square System We wish to solve

$$A\overline{\mathbf{vec}}(C) = c_Q. \tag{25}$$

The right-hand side c_Q is a vector whose entries are

$$c_Q = \begin{bmatrix} q_{\lambda_1}^T \nabla^2 f(x^{\lambda_1\sigma_1}) q_{\sigma_1} \\ \vdots \\ q_{\lambda_\rho}^T \nabla^2 f(x^{\lambda_\rho\sigma_\rho}) q_{\sigma_\rho} \end{bmatrix}.$$

Here $\lambda_1, \ldots, \lambda_\rho$ and $\sigma_1, \ldots, \sigma_\rho$ are the appropriate orderings of the numbers $1, \ldots, n$. Let

$$\eta_i = \overline{\mathbf{vec}}(\nabla^2 f(x^{\lambda_i\sigma_i})), \quad i = 1, \ldots, \rho,\ \eta_i \in \mathbb{R}^\rho.$$

Then we can write

$$c_Q = \begin{bmatrix} (A\eta_1)_1 \\ \vdots \\ (A\eta_\rho)_\rho \end{bmatrix},$$

where $(A\eta_i)_i$ is the ith element of the vector $A\eta_i$. This can be written

$$c_Q = \sum_{i=1}^{\rho} E_i A \eta_i,$$

where E_i is an $\rho \times \rho$ matrix with 1 in position (i,i) and zeros everywhere else. The solution $\overline{\mathbf{vec}}(C)$ becomes

$$\overline{\mathbf{vec}}(C) = A^{-1} \sum_{i=1}^{\rho} E_i A \eta_i.$$

If all the η_i are the same, say, η, we have

$$\overline{\mathbf{vec}}(C) = \eta,$$

regardless of which elements of C_Q we choose to compute. If the η_i are not equal the solution $\overline{\mathbf{vec}}(C)$ will *not* be independent of which C_Q elements are computed, but it can be shown that the resulting matrix C satisfies

$$\|C - \nabla^2 f(\widetilde{x})\|_F \leq \sqrt{2}\kappa(A)\rho L\delta,$$

for some $\widetilde{x} \in \mathcal{N}$. $\kappa(A)$ means the condition number of A. So, the error in C grows linearly with δ as before, and the condition number of A is a factor. The proof of this is given in [5], and is similar to the proof we give for least squares solution below.

Least Squares Solution It is possible to pick more than ρ rows from $(Q^T \otimes Q^T)P_c$ when reducing the size of the equation system. If we choose to compute m elements, $\rho < m \leq n(n+1)/2$, we get an over-determined equation system which can be written

$$\hat{P}_r(Q^T \otimes Q^T)P_c\mathbf{vec}(C) = \hat{P}_r\mathbf{vec}(C_Q), \tag{26}$$

where $\hat{P}_r$ is an $m \times n^2$ matrix which deletes the appropriate rows. (26) can be solved through least-squares solution. Define

$$\hat{A} = \hat{P}_r(Q^T \otimes Q^T)P_c,$$

and

$$\widehat{c_Q} = \hat{P}_r\mathbf{vec}(C_Q),$$

then (26) can be written

$$\hat{A}\overline{\mathbf{vec}}(C) = \widehat{c_Q}, \tag{27}$$

with least squares solution

$$\overline{\mathbf{vec}}(C) = \arg\min_{\overline{c}} \|\hat{A}\overline{c} - \widehat{c_Q}\|_2.$$

Here $\hat{A}$ is an $m \times \rho$ matrix. $\widehat{c_Q}$ is $m \times 1$ and can be written

$$\widehat{c_Q} = \sum_{i=1}^{m} E_i\hat{A}\eta_i,$$

where E_i this time is an $m \times m$ matrix with zeros everywhere except for a 1 in position (i,i), so that

$$\overline{\mathbf{vec}}(C) = (\hat{A}^T\hat{A})^{-1}\hat{A}^T\sum_{i=1}^{m} E_i\hat{A}\eta_i. \tag{28}$$

Here we assume that $(\hat{A}^T\hat{A})$ is invertible. If the η_i are all equal the solution is the same as for the square system. If the η_i are not equal then we can write (28) as

$$\overline{\mathbf{vec}}(C) = \eta + (\hat{A}^T\hat{A})^{-1}\hat{A}^T \sum_{i=1}^{m} E_i\hat{A}\epsilon_i$$

where

$$\eta = \overline{\mathbf{vec}}(\nabla^2 f(\widetilde{x})),$$

for some $\widetilde{x}$, and

$$\epsilon_i = \eta_i - \eta.$$

Let

$$c = \overline{\mathbf{vec}}(C).$$

We then have

$$\|c - \eta\| = \| \sum_{i=1}^{m} (\hat{A}^T\hat{A})^{-1}\hat{A}^T E_i\hat{A}\epsilon_i\|,$$

an upper bound on which is

$$\|c - \eta\| \leq m\|(\hat{A}^T\hat{A})^{-1}\| \; \|\hat{A}^T\| \; \|\hat{A}\| \; \max_i \|\epsilon_i\|.$$

If f satisfies (7), then it can be shown that

$$\max_i \|\epsilon_i\| \leq L\delta,$$

so that

$$\|c - \eta\| \leq \|(\hat{A}^T\hat{A})^{-1}\| \; \|\hat{A}^T\| \; \|\hat{A}\| \; mL\delta,$$

which implies

$$\|C - \nabla^2 f(\widetilde{x})\|_F \leq \|(\hat{A}^T\hat{A})^{-1}\| \; \|\hat{A}^T\| \; \|\hat{A}\| \; \sqrt{2}mL\delta,$$

since, for a symmetric matrix with ρ nonzero elements in the lower triangle, say, B, we have

$$\|B\|_F \leq \sqrt{2}\|\overline{\mathbf{vec}}(B)\|.$$

2.2 A Generalised Sparse Covariation Graph

Consider the function

$$f(x) = (x_1 + x_2)^2. \tag{29}$$

The covariation graph of this function is complete and its Hessian is full, namely

$$\nabla^2 f(x) = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}.$$

There are three elements in the lower triangle, but these all have the same value. As we will see, it is possible to compute the entire Hessian using only one finite difference computation in this case. To see how this can be the case, observe that for

$$U = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

we have

$$U^T \nabla^2 f(x) U = \begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix},$$

which has only one nonzero element. This gives rise to the following technique. Define the covariation graph with respect to the $n \times n$ nonsingular matrix U, $G_U(V, E)$, with n nodes, where each node corresponds one-to-one to a column in U. Let there be an edge between node i and j of G_U if and only if there exist x, h, k such that

$$f(x + hu_i + ku_j) - f(x + hu_i) - f(x + ku_j) + f(x) \neq 0.$$

Lemma 6 *If the covariation graph of f with respect to a nonsingular matrix U, $G_U(V, E)$ is not complete, then the function is partially separable, and subject to a change in variables can be written as a sum of element functions, each with an invariant subspace.*

Proof. We have

$$f(x + hu_i + ku_j) - f(x + hu_i) - f(x + ku_j) + f(x) = 0,$$

for all x, h and k. Without loss of generality assume that $n = 2$ and $x = 0$. For simplicity, let us write $u_1 = p$, $u_2 = q$, and let q_1 be element 1 of q, and so on. Then we can write

$$f(hp_1 + kq_1, hp_2 + kq_2) - f(hp_1, hp_2) - f(kq_1, kq_2) + f(0, 0) = 0,$$

which is the same as

$$f(hp_1 + kq_1, hp_2 + kq_2) = f(hp_1, hp_2) + f(kq_1, kq_2) - f(0, 0). \tag{30}$$

Let $g(h, k)$ be the function $f\big(y(h, k)\big)$, where

$$y(h, k) = \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix} \begin{bmatrix} h \\ k \end{bmatrix},$$

using that p and q are linearly independent. Then, (30) can be written

$$g(h, k) = g_1(h) + g_2(k),$$

where for instance

$$g_1(h) = f(hp_1, hp_2), \text{ and } g_2(k) = f(kq_1, kq_2) - f(0, 0),$$

and g is a sum of element functions, each of which are invariant to changes in one of the variables. □

Assume for simplicity that U is orthogonal. Now we can compute a full matrix C with only ρ finite difference computations along the directions of an orthogonal matrix Q, if there exists an orthogonal matrix U such that C_U is sparse, with ρ nonzero elements in the lower triangle. Usually, we compute ρ elements of the matrix C_Q and impose a sparsity structure on the matrix C in the equation

$$C = QC_QQ^T.$$

The matrix C is not sparse in the current context, but let us impose the appropriate sparsity structure on the matrix

$$Y = U^TCU.$$

Then, we get

$$Y = U^TCU = U^TQC_QQ^TU,$$

which leads to

$$Q^TUYU^TQ = C_Q,$$

which can be written

$$(Q^TU \otimes Q^TU)\mathbf{vec}(Y) = \mathbf{vec}(C_Q).$$

Using the same techniques as described before, one can reduce the size of this $n^2 \times n^2$ equation system to $\rho \times \rho$ and obtain the matrix Y, and finally obtain the matrix

$$C = UYU^T. \tag{31}$$

To give a concrete example, let us consider an extension of the function (29) to dimension n,

$$f(x) = (x_1 + \cdots + x_n)^2. \tag{32}$$

Here, let U be an orthogonal matrix with $e/\sqrt{n}$, e being a vector of all ones, as its first column. The structure of the matrix $U^T\nabla^2 f(x)U$ is all zero except for the element in position (1,1). Let Q be an arbitrary orthogonal matrix, and let us compute the finite difference

$$(C_Q)_{12} = f(x + q_1 + q_2) - f(x + q_1) - f(x + q_2) + f(x),$$

for some arbitrary x. With the appropriate matrices P_r and P_c we can now construct the 1×1 equation system

$$P_r(Q^TU \otimes Q^TU)P_c\overline{\mathbf{vec}}(Y) = c_Q,$$

which gives us

$$\overline{\mathbf{vec}}(Y) = Y_{11} = 2n.$$

From this we can compute C from (31), which becomes a matrix of all 2's, which is equal to the exact Hessian of (32).

2.3 Convergence Theory

A summary of convergence theory for various types of GSS methods on continuously differentiable functions is given in [7]. Since we update the search basis regularly, there are two ways of ensuring global convergence, according to the theory of [7]. One is placing restrictions on when the basis may be updated, and when step lengths may be reduced and increased. This is incorporated into the method of [6]. However, the aim of the current method is to update the basis as often as possible, and this conflicts with the restrictions on basis updates. Consequently, we must ensure global convergence in a different way.

The second option is to enforce sufficient, rather than simple decrease [9]. By simple decrease we mean that a step is accepted if

$$f(x + \delta_i q_i) < f(x). \tag{33}$$

By sufficient decrease we mean accepting a step if

$$f(x + \delta_i q_i) < f(x) - \rho(\delta_i), \tag{34}$$

where $\rho(t)$ is a nondecreasing continuous function, which in this context additionally is required to satisfy

$$\rho(t) = o(t), \text{ when } t \downarrow 0,$$

for the method to be provably convergent. A function which accomplishes this is

$$\rho(t) = \gamma_1 \cdot t^{\gamma_2}, \tag{35}$$

for $\gamma_1 < 0$ and $\gamma_2 > 1$. To allow for long steps γ_1 should be small, so we tentatively set

$$\gamma_1 = 10^{-4},$$

inspired by the Wolfe conditions for line search (see e.g. [11], chapter 3), and $\gamma_2 = 2$.

2.4 Algorithm Pseudo Code

Pseudocode for the algorithm is given in figures 5 and 6. For simplicity, Figure 5 and the following discussion does not cover the material of section 2.2, since this extension is relatively straightforward. Let us look at the main part, Figure 5, line by line. The first part is initialisation, which is determining the initial search vectors, the positive and negative of the unit coordinate vectors in our implementation, as well as the initial step lengths, which we in our numerical experiments choose by the rule:

- If $|x_i^0| > 0$, $\delta_i = 0.05|x_i^0|$.
- If $x_i^0 = 0$ and $\|x^0\| > 0$, $\delta_i = 0.05\|x^0\|$.
- If $\|x^0\| = 0$, $\delta = 0.05e$, where e is a vector of all ones.

It must be decided which elements of C_Q should be computed, which for $Q = I$ we set as the same elements as the nonzero elements of the lower triangle of C.

The main while loop of the algorithm first tests if the method is deemed to have converged. There exist several possible convergence criteria, and and replacing one with another in an algorithm is usually easy. We suggest either

$$\max_i \frac{\delta_i}{\|x\|} < \text{ tolerance},$$

or just

$$\max_i \delta_i < \text{ tolerance},$$

depending on whether or not the variable is assumed to approach zero over the course of the optimisation.

Next, the method orders the search direction for the given iteration. This is done the same way as outlined for the method not exploiting sparsity, by ordering the directions into pairs.

Searching along a direction is done as in Figure 6, where a point is taken if it gives sufficient decrease. If sufficient decrease is obtained, the method tries to double the step length.

Having searched along two directions, the method computes a fourth point as depicted in Figure 3 and computes the appropriate element of C_Q. For the remaining directions after all pairs of directions are have been searched along, the method searches along the remaining directions without considering off-diagonal C_Q elements, but still potentially computing diagonal C_Q elements.

When we are finished searching along all $2n$ directions, the method updates step lengths according to the rule:

- For those j where no step was taken along either q_j or $-q_j$, set $\delta_j \leftarrow \frac{1}{2}\delta_j$.

Once all required off-diagonal elements of C_Q have been computed, the search directions are updated. First we must compute all remaining required diagonal elements, if any. Then we solve the equation system (21) or (26) depending on whether we want to solve an over-determined equation system or not. From the solution we obtain the matrix C, and set the positive and negative of its eigenvectors as the new search directions. In addition we update step lengths. Let Q_{old} be the matrix with the n unique (apart from sign) old basis vectors, and Q_{new} the matrix containing the corresponding new basis vectors. We update step lengths by the rule

$$\delta_{\text{new}} = \left| \left(Q_{\text{new}}^T Q_{\text{old}} \delta_{\text{old}} \right) \right|,$$

the absolute value sign meaning absolute value of each element of the vector $Q_{\text{new}}^T Q_{\text{old}} \delta_{\text{old}}$, a relation deduced from wanting to maintain

$$Q_{\text{new}} \delta_{\text{new}} = Q_{\text{old}} \delta_{\text{old}},$$

and wanting to keep all step lengths nonnegative.

Next, since Q has changed we must determine which elements of C_Q are to be computed , which is equivalent to generating the matrix A discussed in section 2.1. This is done with the procedure select_rows of Figure 4. If we want to solve an overdetermined equation system then we in the experiments first select rows the using the method select_rows, then select the desired number of extra rows by picking the first available elements on the diagonal of C_Q, then on the first sub-diagonal, and so on. When the new basis is in place we perform four compass search iterations along the new search basis before starting to compute C_Q and C. The reason for this, based on initial numerical experience is that it is not helpful to update the basis too often (i.e. at every iteration, which can be done if C is e.g. tridiagonal), so we wait for a little while before pursuing a new search basis. It may of course well be that better rules exist than to just wait four iterations.

```
Initialise
While not converged
    Order directions
    For each direction pair
        Search along first direction
        Search along second direction
        Compute extra point and $C_Q$ element. Update iterate if extra point gives
        sufficient decrease.
    end.
    For each of remaining directions
        Search along direction
    Update step lengths
    If all desired off-diagonal $C_Q$ elements have been computed
        Compute remaining desired diagonal elements. Update iterate if a point
        giving sufficient decrease is found in the process.
        Change basis and update step lengths
        Determine which $C_Q$ elements are to be computed
        Perform ordinary compass search along new basis for 4 iterations.
```

Figure 5: Pseudocode for the algorithm.

```
Given current iterate $x$, search direction $q_i$, step length $\delta_i$, $\gamma_1$ and $\gamma_2$
If $f(x + \delta_i q_i) < f(x) - \gamma_1 \delta_i^{\gamma_2}$
    If $f(x + 2 \cdot \delta_i q_i) < f(x) - 2 \cdot \gamma_1 \delta_i^{\gamma_2}$,
        Take $x + 2 \cdot \delta_i q_i$ as new point.
        Set $\delta_i \leftarrow 2 \cdot \delta_i$.
    Else
        Take $x + \delta_i q_i$ as new point.
    Compute
        $$(C_Q)_{ii} = \frac{f(x + 2\delta_i q_i) - 2f(x + \delta_i q_i) + f(x)}{\delta_i^2}.$$
end.
```

Figure 6: Searching along a direction.

3 Numerical Results

The method was implemented in Matlab, and tested on functions from [10]. The functions chosen for testing are:

- The *extended Rosenbrock* function. This function, used in designing the Rosenbrock method [13], has a block-diagonal Hessian with block size 2. The function is designed to have search methods follow a curved valley, so we expect good results on this function.
- The *Broyden tridiagonal* function. This function has a tridiagonal Hessian.
- The *extended Powell singular* function. This function has a block diagonal Hessian matrix, with block size four. The Hessian is singular in certain subspaces of the domain of the objective function, most importantly at the optimal solution.
- The *discrete boundary value* function. This function has a Hessian with five diagonals.
- The *Broyden banded* function. This function has a Hessian with 13 diagonals.

The results for these smooth functions are given in Table 1. The column "Sparse" contains the number of function evaluations performed to reduce the function value to less than 10^{-5}, when starting from the recommended starting point and computing only the number of C_Q elements needed, that is, ρ elements. The column "LSQ" lists the number of function evaluations to reduce the function value below the same level, but computing 1.5 times the number of C_Q elements strictly needed. "Full" lists the results of computing the entire matrix C_Q. The reason we halt the methods rather then letting them run until they are deemed to have converged is that we are interested in studying the rate of decline produced by the methods, rather then the effects of a stopping criterion. To prevent stagnation, the methods were also halted if

$$\max_i \delta_i \leq 10^{-7}. \tag{36}$$

In general one can say that the "Sparse"-column contains the best results and the "Full"-column the worst, with the "LSQ"-column somewhere in between, and that the relative differences between the columns depend on the function. On the extended Powell Singular function, if we do not halt the methods after a value below 10^{-5} is obtained, then all of the methods perform a huge number of function evaluations without making much progress. The true Hessian becomes more and more ill-conditioned along the paths the methods follow, so it would seem that singular Hessians lead to bad search directions since zero eigenvalues can correspond to arbitrary eigenvectors.

We then add noise to the functions, by testing on the function

$$\widetilde{f}(x) = f(x) + \max\left\{10^{-4} \cdot f(x), 10^{-4}\right\} \cdot \mu,$$

where $-1 \leq \mu \leq 1$ has a uniform random distribution. We run each instance 10 times, and halt the methods when the objective function value drops below 10^{-2}. The results are given in Table 2, the average number of function evaluations being listed. In some instances the methods fail to produce a function value lower than 10^{-2} before they are halted by the criterion (36). For those instances where this happened all 10 times, the corresponding entry in the tables says "Fail". If a method failed on some, but not all 10 runs, then there is a number in parentheses after the average number of evaluations, the number is the number of *successful* runs, and the average number of function evaluations listed is over those successful runs only.

The picture is largely the same as for smooth functions. As n grows, it becomes beneficial to use the "Sparse" or "LSQ" approach. The "Full" approach fails frequently for large n.

4 Summary

We have presented a provably convergent GSS method which exploits average curvature information as well as partial separability. Numerical results have shown that taking separability into account gives a method which usually produces a faster rate of function value decline than not doing so, on smooth as well as noisy functions. In addition, the method exploiting separability succeeds on some noisy problems for large n where its counterpart fails.

Function	n	Sparse	LSQ	Full
Extended Rosenbrock	4	603	637	653
	8	1249	1346	1938
	16	2497	2693	6093
	32	4993	5514	18399
	64	10273	10538	50163
	128	20545	21941	184136
Ext. Powell Singular	4	237	-	204
	8	355	572	788
	16	936	961	1890
	32	1804	2351	5793
	64	4669	5915	21797
	128	9346	8777	77257
Broyden Tridiagonal	4	219	-	168
	8	390	376	449
	16	851	897	1003
	32	1791	1803	2377
	64	3563	3366	5779
	128	7611	8000	12035
Discrete boundary value	4	81	-	82
	8	191	195	237
	16	913	629	1028
	32	844	846	3522
Broyden banded	4	215	-	230
	8	499	-	500
	16	994	-	1156
	32	2240	2373	2342
	64	4735	4648	5081
	128	9242	10344	10647

Table 1: Number of function evaluations needed to reduce the objective function value to less than 10^{-5}, with $\gamma_1 = 10^{-4}$ and $\gamma_2 = 2$. In the experiments reported in the LSQ column 1.5 times the needed amount of C_Q elements were computed. A "-" entry signifies that 1.5 times the number of needed elements exceeds the total number of available elements in C_Q.

Function	n	Sparse	LSQ	Full
Extended Rosenbrock	4	496.8	528.3	528.7
	8	1022.0	1126.5	1753.2
	16	2069.3	2298.1	5604.5(8)
	32	4284.2	4771.4	Fail
	64	8919.4	9900.8	Fail
	128	18773.8	22542.2	Fail
Ext. Powell Singular	4	128.8	-	115.5
	8	268.5	262.9	515.0
	16	578.4	561.5	1441.7
	32	1448.1	1485.4	2428.5(2)
	64	3519.4	3452.7	Fail
	128	7306.3	8745.3(9)	Fail
Broyden Tridiagonal	4	135.9	-	82.4
	8	223.6	223.2	231.9
	16	428.4	443.0	608.4
	32	862.9	868.5	1515.3
	64	1804.8	1809.9	3098.1
	128	3947.6	4060.1	6207.5
Broyden banded	4	143.2	-	145.8
	8	319.6	-	310.5
	16	713.0	-	691.3
	32	1493.8	1596.3	1594.9
	64	3144.4	3197.4	3534.8
	128	6810.8	6690.0	7592.0

Table 2: Number of function evaluations needed to reduce the objective function value with noise added to less than 10^{-2}, with $\gamma_1 = 10^{-4}$ and $\gamma_2 = 2$. Average over 10 runs listed.

References

[1] M. A. Abramson, C. Audet, and J. E. Dennis, Jr. Generalized pattern searches with derivative information. *Mathematical Programming, Series B*, 100:3–25, 2004.

[2] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *Les Journées de l'Optimisation 2004*, 2004.

[3] I. D. Coope and C. J. Price. A direct search conjugate directions algorithm for unconstrained minimization. *ANZIAM Journal*, 42(E):C478–C498, 2000.

[4] R. Fletcher, A. Grothey, and S. Leyffer. Computing sparse hessian and jacobian approximations with optimal hereditary properties. In L. Biegler, T. Coleman, A. Conn, and F. Santosa, editors, *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*. 1997.

[5] L. Frimannslund and T. Steihaug. A generating set search method exploiting curvature and sparsity. In *Proceedings of the Ninth Meeting of the Nordic Section of the Mathematical Programming Society*, pages 57–71, Linköping, Sweden, 2004. Linköping University Electronic Press.

[6] L. Frimannslund and T. Steihaug. A generating set search method using curvature information. To appear in Computational Optimization and Applications, 2006.

[7] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[8] 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.

[9] S. Lucidi and M. Sciandrone. On the global convergence of derivative-free methods for unconstrained optimization. *SIAM Journal on Optimization*, 13(1):97–116, 2002.

[10] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer–Verlag, 1999. ISBN 0–387–98793–2.

[12] C. P. Price and P. Toint. Exploiting problem structure in pattern search methods for unconstrained optimization. *Optimization Methods and Software*, 21(3):479–491, 2006.

[13] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, Oct. 1960.

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