Paper I

Chapter 1

A GENERATING SET SEARCH METHOD USING CURVATURE INFORMATION*

Lennart Frimannslund
Department of Informatics, University of Bergen, Norway
lennart.frimannslund@ii.uib.no

Trond Steihaug
Department of Informatics, University of Bergen, Norway
trond.steihaug@ii.uib.no

Abstract Direct search methods have been an area of active research in recent years. On many real-world problems involving computationally expensive and often noisy functions, they are one of the few applicable alternatives. However, although these methods are usually easy to implement, robust and provably convergent in many cases, they suffer from a slow rate of convergence.

Usually these methods do not take the local topography of the objective function into account. We present a new algorithm for unconstrained optimisation which is a modification to a basic generating set search method. The new algorithm tries to adapt its search directions to the local topography by accumulating curvature information about the objective function as the search progresses.

The curvature information is accumulated over a region thus smoothing out noise and minor discontinuities. We present some theory regarding its properties, as well as numerical results. Preliminary numerical testing shows that the new algorithm outperforms the basic method most of the time, sometimes by significant relative margins, on noisy as well as smooth problems.

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

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

1. INTRODUCTION

When choosing an optimisation method for unconstrained optimisation the choice is often Newton's method or quasi-Newton methods. If the function is well-defined through computer code, one can obtain the required derivatives to machine precision by the use of automatic differentiation techniques (AD), with little extra cost in the case of the gradient. A different alternative is to use finite differences (FD), which produces approximations to the desired derivatives.

Consider now less ideal scenarios than the one just described. AD methods may be inapplicable if the source code is not available, or, say, if the computer code representing the function is written in several languages. Furthermore, the nature of the function might make AD techniques inappropriate. In [19], function evaluations that involve integrating a function backwards in time as well as poor portability make AD an undesirable option. As for FD, in some instances the approximate derivatives obtained are not helpful either. Optimization problems in computational fluid dynamics where the objective function includes integrals [5, 6] are examples of this. The reason FD derivatives can fail here is that the discretisation involved in solving the integral introduces noise. This noise is numerical in nature, in the sense that the same input always gives the same output, but gives inaccurate FD derivatives, and in [6] spurious solutions which are not present in the underlying objective function. Plots of the objective functions in [5] look like the function in Figure 1.1. Although there is an underlying, smooth function, it is obscured by noise.

Generating set search (GSS) methods (see e.g. [2, 3, 15, 18]) are methods which try to overcome the difficulties mentioned. They do this by not using derivative information, and not allowing the topography of the function to degenerate the set of search directions they consider. GSS methods have been applied to a wide range of real-world problems, both computationally expensive and inexpensive, among others design of thermal insulation systems [1], shape optimization in aeroacoustics [19], and helicopter rotor blade design [4]. GSS methods have also been implemented on parallel machines [10].

These methods are however, usually slow when it comes to convergence. If the function to be minimised is expensive, as is the case in [4] where a single evaluation takes minutes, it might be impossible to perform, say, 10,000 function evaluations to reach the optimum, even in a high-performance computing environment.

We will show an approach to how to utilise curvature information, which on several examples improves a basic GSS method on both smooth

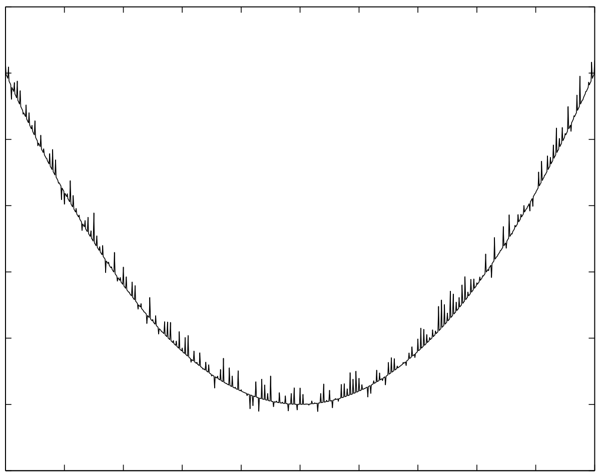

Figure 1.1. Quadratic function with noise.

and noisy problems *when such information is helpful*, without deteriorating performance-wise if curvature information is not helpful. Coope and Price [7] have developed a generating set search method for smooth problems using conjugate directions. This method also builds up and uses second order information, but the two methods differ on several issues.

We consider the unconstrained optimisation problem

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$. Extensions toward linearly constrained optimisation [16] can readily be incorporated into our method. In addition, there exist strategies [3, 9, 17] dealing with nonlinear constraints which are applicable as well. A strategy to tailor these methods toward the large-scale case has been studied in [21]. An extension of this paper to separable functions, also allowing for larger n appeared as [13].

This paper is organised as follows. In the rest of section 1 we outline two basic, existing GSS methods. In section 2 we present our new method, and in section 3 sketch some of its theoretical properties. Section 4 presents numerical experiments, while section 5 offers some concluding remarks.

1.1. BASIC GSS ALGORITHMS

A basic variant of GSS is what we will call Coordinate Search. Let $q_i,\ i = 1 \dots n$ be an orthonormal basis, and $\mathcal{G}$ be

$$\mathcal{G} = \bigcup_{i=1}^{n} \{q_i, -q_i\}, \tag{2}$$

the set of search directions. The standard choice giving justification to the name of the algorithm is

$$q_i = e_i,\ i = 1 \dots n,$$

where e_i is the vector with zeros everywhere except for 1 in position i.

A pseudo-code for coordinate search on the unconstrained problem (1) is given below. The algorithm evaluates the function along each search direction, and steps to the point which reduces the function value the most. See Figure 1.2.

```
Coordinate Search
  Given $\delta_{tol} > 0$,   $\alpha \geq 1 > \beta > 0$ and $x \in \mathbb{R}^n$.

  While $\delta > \delta_{tol}$,
    Compute $v : \min_{v \in \mathcal{G}} f(x + \delta v)$,
    If $f(x + \delta v) < f(x)$,
      Set $x \leftarrow x + \delta v$.
      Set $\delta \leftarrow \alpha\delta$.
    Else, set $\delta \leftarrow \beta\delta$.
  end.
```

A typical choice is $\beta = \frac{1}{2}$. Under reasonable assumptions on f, this algorithm can be shown [15, 24] to be globally convergent in the sense that

$$\lim_{k \to \infty} \inf \|\nabla f(x^k)\| = 0.$$

1.2. COMMON GSS VARIANTS

There exist several modifications to this basic algorithm. One modification is to introduce individual step lengths for each search direction. This tactic is a possible remedy for variable scaling issues. Furthermore, the algorithm needs not consider all coordinate directions before accepting the step. Simply stepping to a point as soon as a smaller function value is identified, gives an algorithm we will call *Compass Search*. See Figure 1.3. In the figure, the search starts at the black node/point.

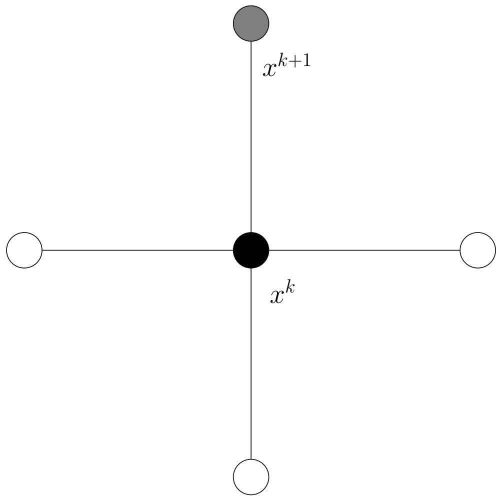

Figure 1.2. Coordinate search in $\mathbb{R}^2$.

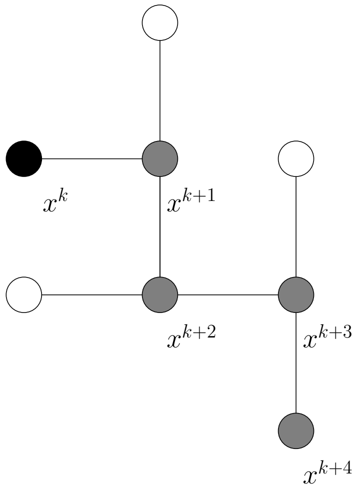

Figure 1.3. Compass Search in $\mathbb{R}^2$: Search along positive and negative coordinate directions, and step immediately if function reduction found.

First, we search rightwards, and step. Then we search upwards, but do not step. Then downwards, step, then leftwards, but do not step, etc.

The number of search directions in both methods when using coordinate directions is $2n$, but this number can be reduced. A set of vectors $v_i, i = 1, \ldots, r$ constitutes a positive basis or generating set if for every

$x \in \mathbb{R}^n$,

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

It can be shown [8] that $n + 1 \leq r \leq 2n$, so in theory a GSS method only needs $n + 1$ directions. In this paper we will use a positive basis of $2n$ directions, the positive and negative of n orthogonal directions.

Comprehensive convergence theory for GSS methods can be found in [15]. In general, no more than linear convergence can be expected.

2. THE BASIC IDEA

Our basic idea is to rotate the search basis based on curvature information. A similar scheme, which aligns the basis to the average direction the search progresses, appeared as early as 1960 in [22]. To illustrate the idea we use Compass Search, although it can be applied to other algorithms as well. Thus, the new method is a Compass Search, but with a dynamic search basis. It implicitly uses a quadratic model function by assuming we are minimising, say,

$$g(y) = \phi + b^T(y - x) + \frac{1}{2}(y - x)^T C(y - x),$$

where C is a symmetric matrix, although we in reality are minimising a general function f. Let the search directions (2) of our search be positive and negative of the column vectors of the orthogonal matrix Q, that is,

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}.$$

Since g is a quadratic function, we have

$$q_i^T C q_j = \frac{g(x + \delta_i q_i + \delta_j q_j) - g(x + \delta_i q_i) - g(x + \delta_j q_j) + g(x)}{\delta_i \delta_j}.$$

For a general function function f define the matrix C_Q with element (i, j) by

$$(C_Q)_{ij} = \frac{f(x + \delta_i q_i + \delta_j q_j) - f(x + \delta_i q_i) - f(x + \delta_j q_j) + f(x)}{\delta_i \delta_j}. \quad (3)$$

In addition, let

$$C = Q C_Q Q^T, \quad (4)$$

and C will be an approximation to the Hessian matrix (and be exact for a quadratic function).

The four points in (3) make up the corner points of a rectangle in the (q_i, q_j)-plane. If we take $q_i = q_j$ we get a formula for the second derivative along this vector, consisting of four or three points, depending on whether or not $\delta_i = \delta_j$. As the compass-type search progresses, the algorithm performs exploratory function evaluations, and as can be gathered from Figure 1.3 some of these points will lie in the constellations required by (3). For $n > 2$, the necessary rectangles can be constructed with little extra effort. In this way the matrix C_Q can be built up in a predictable and systematic fashion. The algorithm's key ingredients are:

- Compass Search along the columns of Q and $-Q$.
- Computation of the terms $(C_Q)_{ij}$ by (3) as the method updates x, with adaptive shuffling of search directions to facilitate all combinations of i and j, $i \geq j$.
- Application of formula (4) to obtain C.
- Computation of all the eigenvectors q_i of C, and setting these eigenvectors as the new search basis Q.

The initial choice of Q can be, for instance, $Q = I$.

Although eigenvector computation is considered expensive, the cost must be seen relative to that of a function evaluation. If a function evaluation takes many seconds, not to say minutes, an eigenvalue factorisation is inexpensive by comparison.

When f is Noisy — Average Curvature Information. Consider again the function in Figure 1.1. As mentioned in the introduction, a finite difference-based method using small differences will run into problems on this function since local rate of change and local curvature may differ very much from the *average* rate of change and curvature in the region covered by the figure. However, a finite difference-scheme with sufficiently *large* differences will capture these average quantities. (See for example [14] for a discussion.) In addition, average curvature can be estimated from a wide range of sample points, as long as they are sufficiently far apart.

We suggest using relatively large step sizes and thereby gather information about average second derivatives. Hopefully, this information will provide us with eigenvectors that make good search directions in the sense that they allow for long steps even if, for instance, we are in a narrow valley. Once the algorithm nears the optimal solution step lengths become smaller, and we then obtain local curvature information, which we want when close to the optimum.

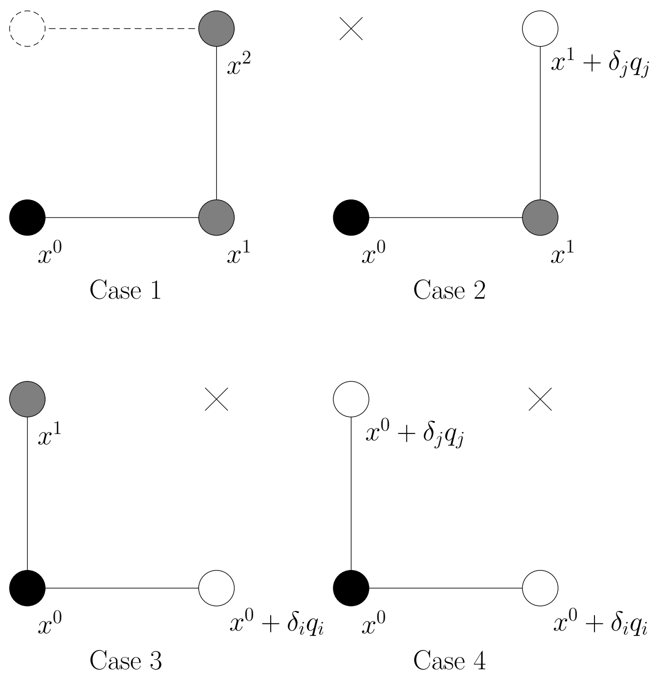

Figure 1.4. The four possible outcomes of successive searches along q_i and q_j. A grey node signifies a step which has been taken, a white node signifies a step not taken. The search starts at the black node in each case.

2.1. COMPUTING THE MATRIX C_Q SYSTEMATICALLY

If the search directions $\pm q_i$ and $\pm q_j$ are ordered

$$\left[\begin{array}{ccccc} q_i & q_j & -q_i & \cdots & -q_j \end{array}\right],$$

then the first three directions may enable the algorithm to compute $(C_Q)_{ij}$. The reason for this is that a successful search and subsequent step along the two first directions and a successful or unsuccessful step along the third direction provides the rectangle of points needed by (3). However, this is not always the case, and we consider now the four possible cases that can occur in the first two steps. In the following paragraphs, we consider the ordering

$$\left[\begin{array}{ccc} q_i & q_j & -q_i \end{array}\right], \tag{5}$$

and look at the situation after two trial steps. All four cases refer to Figure 1.4.

Case 1 — Success along both directions. Two successful steps indicated by grey nodes, and the point along $-q_i$, indicated by the dashed line and circle have been computed. An approximation to $(C_Q)_{ij}$ by equation (3) can be computed regardless of the success or failure of the third step.

Case 2 — Success along first direction only. Three points have been evaluated, but the algorithm has not stepped to the second point since it does not provide a lower function value. This is indicated by a white node. An extra function evaluation at the point marked by a cross must be computed in order to obtain $(C_Q)_{ij}$. Note that the next point scheduled for evaluation (along $-q_i$) is the point x^0, and there is no need for this evaluation.

Case 3 — Success along second direction only. As in case 2, the algorithm has to perform one extra function evaluation, marked by a cross. The next point scheduled for evaluation falls outside the square, and will be evaluated and accepted if leading to a reduction in function value.

Case 4 — No Success along either direction. In this case the algorithm also needs to perform one extra evaluation at the point marked by a cross, and consider the next point scheduled for evaluation as in case 3.

Choosing the Ordering of Directions. How to best order the search directions can be illustrated by an example. Consider the case with $n = 4$, and order the eight directions in two groups each consisting of three directions (like the ordering (5)) and let the remaining (two) directions be columns in the matrix B.

$$\left[\begin{array}{c|c|c} T_1 & T_2 & B \end{array}\right] = \left[\begin{array}{ccc|ccc|cc} q_1 & q_2 & -q_1 & q_3 & -q_2 & -q_3 & q_4 & -q_4 \end{array}\right].$$

The first group, T_1, enables computation of $(C_Q)_{21}$ and T_2 gives us $(C_Q)_{32}$. The last group, B, consists of left over vectors, and can be ordered arbitrarily. After searching along all directions, it is time to reshuffle. This time, we can order the directions

$$\left[\begin{array}{ccc|ccc|cc} q_1 & q_4 & -q_1 & q_2 & -q_4 & -q_2 & q_3 & -q_3 \end{array}\right],$$

to compute $(C_Q)_{41}$ and $(C_Q)_{42}$. The remaining elements can be obtained through successive, appropriate orderings. This way, dynamically ordering directions to help computing C_Q elements, we obtain all its off-diagonal elements in a systematic fashion. The important observation is that the first and third members of a triplet are the negative of each other.

When computing matrix elements this way we can predict how quickly we obtain all off-diagonal elements of C_Q. Let us call the search process between two shuffles a *sweep*. We obtain approximately $\frac{2n}{3}$ off-diagonal elements per sweep, since there are $2n$ search directions and we need three directions to compute an element. Since we need to estimate $\frac{n^2-n}{2}$ off-diagonal elements, this can be done in approximately

$$\frac{\frac{n^2-n}{2}}{\frac{2n}{3}} = \frac{3}{4}(n-1)$$

sweeps, which depends on n. This is not a problem in our experience for n smaller than about 30.

Diagonal Elements. The computation of diagonal elements requires three or four points along a line depending on whether or not $\delta_i = \delta_j$ in (3). A constellation of three points we can achieve when adjusting step lengths. If we immediately try to double the step length when encountering a function value reduction, we have the three points we need. However, the rate at which we will obtain diagonal elements this way is difficult to predict. Potentially, we can obtain all n elements in the first sweep, or we might not obtain any elements if we have step lengths which are too large. In any event, since there are only n diagonal elements, we can afford to compute these separately if needed.

Off-diagonal element Computation using Pairs or Triplets. Upon studying Figure 1.4 one might wonder if the grouping of search directions into triplets is necessary, since only the case of success along the first two directions makes use of the fact that the third direction is the negative of the first direction. Indeed, it is possible to group the search directions into pairs and estimate the objective function value at extra points as marked by crosses in Figure 1.4, but our preliminary numerical experience suggests that this does not lead to a more effective algorithm. On the contrary, the resulting algorithm on average performs slightly poorer than the algorithm employing triplets, the reason for this seemingly being that the extra cost of always having to compute an extra point outweighs the advantages of obtaining a full matrix at an earlier time. This matter could however potentially benefit from further study, especially when n is relatively large.

2.2. THE ALGORITHM

We now present the new algorithm in pseudo-code format, listed in Figure 1.5. It requires an initial guess, x^0, an $n \times n$ orthogonal basis matrix Q with column i being q_i, as well as a vector of step lengths δ, where each component δ_i corresponds to the vectors q_i and $-q_i$. In the algorithm we use the term "successful step" if the variable is updated. Note also that the variable x^{k+1} can be overwritten several times by candidate variable values before k is increased, thereby accepting the new variable value. The helper function *exploratory_moves* is listed in Figure 1.6. The update of δ in step 5 (one of many possible updates) seeks to preserve the properties of step lengths in the old basis to the new basis. It is based on the same change that applies to the basis matrix Q, that is, since Q is replaced by X, which can be accomplished by multiplying it with XQ^T, the method does the same with δ. The factor 2 is to undo the step length reduction in step 4.

The function *exploratory_moves* takes as input x_{in}, q, δ_{in} and k_{in}, and gives as output x_{out}, δ_{out} and k_{out}. The convergence criterion mentioned in step 6 can be set as

$$\max_i \delta_i < \text{tolerance}. \tag{6}$$

This criterion gives good results in practice, as discussed in [11]. Since we obtain an approximation to the Hessian and the algorithm in theory can gather gradient information as well, it would be possible to add a Newton step to the algorithm, as is done in [7] to speed up convergence on smooth functions.

3. THEORETICAL ASPECTS

In this section we investigate the relationship between C and $\nabla^2 f(x)$, as well as address the relationship between our new method and existing convergence theory.

The following lemma can be found in textbooks (e.g. lemma 3.5 in [12]).

Lemma 1 *Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be two times continuously differentiable. Let $\|p\| = 1$ and $\|q\| = 1$. Given $h, k \in \mathbb{R}$, then there exists a $t \in (0, h)$ and an $s \in (0, k)$ such that*

$$\frac{f(x + hp + kq) - f(x + hp) - f(x + kq) + f(x)}{hk} = p^T \nabla^2 f(x + tp + sq)q.$$

Now we can turn our attention to the effect of the rotation (4) in the algorithm. We first show a result for non-orthogonal search directions,

```
Step 1
Order directions $\left[\ T_1 \ \middle|\ T_2 \ \middle|\ \cdots \ \middle|\ B\ \right]$, where $T_i = \left[\ q_r \quad q_s \quad -q_r\ \right]$,
for columns $q_r$ and $q_s$ in $Q$, where $(C_Q)_{rs}$ is not yet computed,
and $B$ are leftover columns from triplet partitioning.
Step 2
For each $T_i$:
    $(x^{k+1}, \delta_r, k_{\text{out}}) \leftarrow \textit{exploratory\_moves}(x^k, q_r, \delta_r, k)$. $k \leftarrow k_{\text{out}}$.
    $(x^{k+1}, \delta_s, k_{\text{out}}) \leftarrow \textit{exploratory\_moves}(x^k, q_s, \delta_s, k)$. $k \leftarrow k_{\text{out}}$.
    If the search along both $q_r$ and $q_s$ was successful,
        $(x^{k+1}, \delta_r, k_{\text{out}}) \leftarrow \textit{exploratory\_moves}(x^k, -q_r, \delta_r, k)$ $k \leftarrow k_{\text{out}}$,
        (Figure 1.4, case 1)
    end.
    Compute $(C_Q)_{rs}$ by (3), evaluating extra point if needed
    (Figure 1.4, case 2, 3 and 4).
    If in case 3 or 4,
        $(x^{k+1}, \delta_r, k_{\text{out}}) \leftarrow \textit{exploratory\_moves}(x^k, -q_r, \delta_r, k)$. $k \leftarrow k_{\text{out}}$,
    end.
end.
Step 3
For each direction $q_i$ in $B$,
    $(x^{k+1}, \delta_i, k_{\text{out}}) \leftarrow \textit{exploratory\_moves}(x^k, q_i, \delta_i, k)$. $k \leftarrow k_{\text{out}}$,
end.
Step 4
For all $j$, if no step was made along $\pm q_j$ where $\delta_j$ has been
increased by exploratory_moves, set $\delta_j \leftarrow \frac{1}{2}\delta_j$.
If no step lengths have been increased by exploratory_moves or
such increase has been undone, and if no step was made along any
direction, set $\delta \leftarrow \frac{1}{2}\delta$.
Step 5
If all off-diagonal elements of $C_Q$ have been computed and no step
was made along any direction:
    Compute remaining diagonal $C_Q$ elements by (3).
    Set $C \leftarrow QC_QQ^T$.
    Eigenvalue-factorise $C$: $C = X\Lambda X^T$.
    Set $\delta \leftarrow 2 \cdot XQ^T\delta$.
    Set $Q \leftarrow X$.
Step 6
If convergence criterion satisfied, terminate, otherwise go to step 1.
```

Figure 1.5. Pseudocode for the algorithm.

```
**exploratory_moves**$(x_{\text{in}}, q, \delta_{\text{in}}, k_{\text{in}})$

    If $f(x_{\text{in}} + \delta_{\text{in}} q) < f(x_{\text{in}})$,
        If $f(x_{\text{in}} + 2\delta_{\text{in}} q) < f(x_{\text{in}} + \delta_{\text{in}} q)$,
            $x_{\text{out}} \leftarrow x_{\text{in}} + 2\delta_{\text{in}} q$,
            $\delta_{\text{out}} \leftarrow 2\delta_{\text{in}}$,
        else
            $x_{\text{out}} \leftarrow x_{\text{in}} + \delta_{\text{in}} q$,
            $\delta_{\text{out}} \leftarrow \delta_{\text{in}}$,
        end.
        Compute diagonal $C_Q$ element corresponding to $q$ by (3),
        $k_{\text{out}} \leftarrow k_{\text{in}} + 1$,
    else
        $x_{\text{out}} \leftarrow x_{\text{in}}$,
        $\delta_{\text{out}} \leftarrow \delta_{\text{in}}$,
        $k_{\text{out}} \leftarrow k_{\text{in}}$,
    end.
```

Figure 1.6. The helper function *exploratory_moves*.

$p_1, \ldots, p_m$. Let

$$p_1, \ldots p_m \in \mathbb{R}^n, \ \|p_k\| = 1, k = 1, 2 \ldots m \leq n, \tag{7}$$

and assume that the elements $(C_P)_{ij}$ of the symmetric $m \times m$ matrix C_P have been computed using formula (3) at the points

$$\left\{x^{ij}, \ x^{ij} + h^{ij} p_i, \ x^{ij} + k^{ij} p_j, \ x^{ij} + h^{ij} p_i + k^{ij} p_j\right\}, \tag{8}$$

for all $(i, j), \ i \geq j$ and $(C_P)_{ji}$ set to be equal to $(C_P)_{ij}$. Let N be the union of all such points and let

$$\delta = \max_{z, y \in N} \|z - y\|, \tag{9}$$

and

$$\mathcal{N} = \left\{ x \in \mathbb{R}^n \,\middle|\, \max_{y \in N} \|x - y\| \leq \delta \right\}. \tag{10}$$

Lemma 2 *Assume that f is twice continuously differentiable and $\nabla^2 f$ is Lipschitz-continuous in $\mathcal{N}$, that is*

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq L\|x - y\|, \ \textit{for all } x, y \in \mathcal{N},$$

where $\mathcal{N}$ is defined by (10). Then the $m \times m$ symmetric matrix C_P with entry $(i, j), i \geq j$ computed by (3) at the points (8), and for any $x \in \mathcal{N}$

satisfies

$$\|C_P - P^T \nabla^2 f(x) P\| \leq mL\delta, \tag{11}$$

where

$$P = \left[\ p_1 \mid p_2 \mid \cdots \mid p_m\ \right]$$

is the $(n \times m)$-matrix where column k is p_k in (7).

Proof. From Lemma 1 we have for each pair (i,j), a point $\widetilde{x}^{ij}$ such that

$$(C_P)_{ij} = p_i^T \nabla^2 f(\widetilde{x}^{ij}) p_j,$$

where $\widetilde{x}^{ij} \in \mathcal{N}$. Then,

$$|(C_P)_{ij} - p_i^T \nabla^2 f(x) p_j| \leq L\delta,$$

and

$$\|C_P - P^T \nabla^2 f(x) P\| \leq mL\delta.$$

□

The result can be stated for an orthogonal matrix Q.

Corollary 3 *Given the quantities (7)–(10), but replacing (7) with n orthogonal vectors q_i, $i = 1 \ldots n$, and updating the other quantities accordingly, we have*

$$\|QC_QQ^T - \nabla^2 f(x)\| \leq nL\delta. \tag{12}$$

The dissimilar placement of the matrices Q and P in (11) and (12) respectively owes to the fact that multiplication by an orthogonal matrix Q does not affect norms.

Convergence Theory. In [15] several requirements are listed for a GSS method to be convergent. Given a GSS algorithm that enforces simple decrease (a step is accepted only if it produces a smaller function value, but there is no requirement of sufficient decrease), searches along the elements of a generating set $\mathcal{G}$ and an additional set of directions $\mathcal{H}$, restricted to be integer combinations of the directions in $\mathcal{G}$, the requirements are: Such a method is convergent if it reduces its step lengths or updates $\mathcal{G}$ only if the search fails along all directions in $\mathcal{G}$, and never increases step lengths.

The new method meets these requirements. It searches along the positive and negative of the column vectors of an orthogonal matrix Q, which, when multiplied with the corresponding step lengths in δ make up a generating set $\mathcal{G}$. (The members of a generating set need not have unit length.) Although step lengths are allowed to be increased, this amounts to searching along directions in $\mathcal{H}$. Since step lengths are reduced only if step length increase has been undone by subsequent reductions and search fails along all directions, this is the same as the search failing along all directions of $\mathcal{G}$ as required. Additionally, basis rotation, which is an update of $\mathcal{G}$, only takes place when search has failed along all directions in $\mathcal{G}$. The subsequent multiplication

$$\delta \leftarrow 2XQ^T\delta$$

can be seen as part of the update of $\mathcal{G}$.

For Compass Search, which does not update its search basis, the requirements are not as strict. It may increase and decrease step lengths freely as long as it enforces simple decrease and all iterates lie on a rational lattice. See [15] for details.

4. NUMERICAL RESULTS

We tested the algorithm on several functions from [20], and compared it with our version of Compass Search which we get if we suspend direction shuffling and C_Q element computation at all times as well as relax the requirements on step length updating corresponding to the requirements of the convergence theory, by allowing step lengths to be decreased even if search has not failed along all directions of $\mathcal{G}$. The reason we compare with Compass Search, which is arguably a slow method, is that this will effectively illustrate the benefits of the idea of basis rotation based on curvature information.

The test set consists of functions from [20] of the form

$$F : \mathbb{R}^n \mapsto \mathbb{R}^m,$$

the test functions being

$$f(x) = F(x)^T F(x), \tag{13}$$

following the recommendations in [20]. All the test functions have an optimal value of zero. The initial step lengths where chosen as follows:

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

On some functions this choice of step length lead to the methods finding the optimal solution almost immediately (e.g. if $x^0 = (\ 1 \quad 1\ )^T$ and $x^* = (\ 0 \quad 0\ )^T$), so in such cases custom initial step lengths were used.

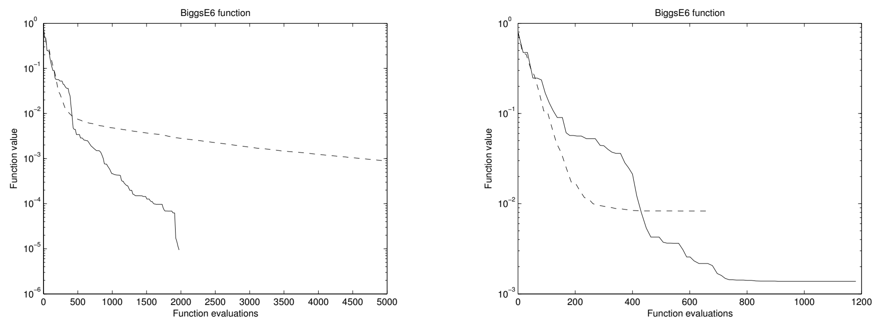

Figure 1.7. Logarithmic plot of function values vs. the number of function evaluations on function #18, without noise added in the left plot, and with noise in the right plot. The solid curve corresponds to the new algorithm, the dashed curve to Compass Search.

On smooth functions, the methods were halted once a function value less than 10^{-5} was obtained, if the maximum step length was less than 10^{-12} or if the number of sweeps exceeded 10000. The results are presented in Table 1.1. The the first column lists the function number, corresponding to the function names in Table 1.3. The next three columns are, for the new algorithm, the number of function evaluations performed, the lowest function value obtained, and the number of basis changes, which is equal to the number of C-matrices computed. The last two columns are, for Compass Search, the number of function evaluations performed, and the lowest function value found.

As can be seen, the new algorithm outperforms Compass Search most of the time, sometimes by significant margins, with but a few exceptions. The most notable case where the new algorithm performs worse than Compass Search is function number 4. This is a function which is badly scaled and is very well suited to methods that search along the coordinate directions. On this function the new method initially steers away from the search directions based on the identity matrix, and also suffers from the requirements on step length increase set by the convergence theory.

To make the test examples more realistic we added noise to the functions. We adopt the noise scheme of [23], which is to test on the function

$$\widetilde{f}(x) = f(x) + \max(10^{-4} \cdot |f(x)|, 10^{-4}) \cdot \mu, \tag{14}$$

where μ is uniformly distributed in the interval $[-1, 1]$. The methods were halted once a function value less than 10^{-2} was obtained, or, as before, the maximum step length was less than 10^{-12} or the number of sweeps more than 10000. The results, which are median value over

Table 1.1 . Numerical results on smooth functions. Methods halted when $f \leq$ 1e-5.

Function	New algorithm			Compass Search	
#	#feval	f^*	#Basis	#feval	f^*
1	461	2.59e-07	10	11118	9.99e-06
3	134	1.19e-07	5	118	1.87e-07
4	1659	2.70e-06	33	222	4.58e-13
5	200	6.39e-06	7	186	9.86e-06
7	340	2.13e-06	6	5978	9.93e-06
14	617	5.49e-07	7	6328	9.91e-06
18	1973	9.41e-06	10	37499	9.99e-06
21	11705	6.73e-06	11	65456	9.96e-06
22	1637	1.44e-06	6	224381	4.93e-04
25	312	7.11e-06	3	532	9.36e-06
28	215	1.31e-06	2	839	9.63e-06

100 runs, are shown in Table 1.2. Function #28 is not included, since its initial value is lower than 10^{-2} with the starting point used. Both methods fail on function #4, which is badly scaled, the new method fails on function #3, which is also badly scaled.

On the noisy functions the methods resemble each other more than on smooth functions, although the new method still outperforms Compass Search by significant margins in some cases. To understand why the picture is different than on smooth functions, one can look at Figure 1.7. In the left plot, we see that the two methods start with a similar rate of decline in function values, but that the curve corresponding to Compass Search levels off after a while. This phenomenon is quite typical for Compass search, and indeed it has been pointed out (e.g. [25]) that although pattern search methods are slow when it comes to convergence, they are good at finding approximate solutions. On noisy functions, however, the methods are halted before Compass Search starts leveling off, and hence the results are more similar. This can be observed in the right plot of Figure 1.7. One function where a good approximation is not obtained quickly is the Rosenbrock function, which is designed to make algorithms search along a curved valley. On this function (#1, and #21 for the extended Rosenbrock function) we observe similar behaviour as on smooth functions, in the sense that the new algorithm reduces the amount of function evaluations significantly.

Table 1.2 . Numerical results on noisy functions. Median over 100 runs shown. Methods halted when $f \leq$ 1e-2.

Problem	New algorithm			Compass Search	
#	#feval	f^*	#Basis	#feval	f^*
1	445.5	7.20e-03	12	2424.5	2.30e-02
3	59	1.00e+00	2	75.5	7.19e-03
4	77	1.00e+12	3	58.5	1.00e+12
5	94	6.12e-03	3	50	9.65e-03
7	172	9.81e-04	3	2719	2.37e-02
14	344	7.71e-03	4	345	8.89e-03
18	434	9.01e-03	2	227	9.66e-03
21	7421	9.37e-03	7	12419	1.13e-01
22	301.5	6.44e-03	1	792.5	1.19e-02
25	180	5.90e-03	1	127	9.08e-03

Table 1.3 . Test function names.

#	n	m	Function name
1	2	2	Rosenbrock
3	2	2	Powell badly scaled
4	2	3	Brown badly scaled
5	2	3	Beale
7	3	3	Helical Valley
14	4	6	Wood
18	6	13	Biggs EXP6
21	10	10	Extended Rosenbrock
22	8	8	Extended Powell Singular
25	4	6	Variably dimensioned
28	5	5	Discrete boundary value

5. CONCLUDING REMARKS

In this paper we suggested a modification of the algorithm Compass Search, to make it more aware of the local topography of the objective function. On smooth functions we have had good results when it comes to reducing the number of function evaluations, where reduction of well over 50% is not uncommon. On noisy functions, when approximate solutions are obtained after few function evaluations, the two methods resemble each other in performance. When approximate solutions cannot be obtained quickly, we again get good results for the new method. Only rarely does it perform significantly worse than Compass Search.

References

[1] M. A. Abramson. Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. *Optimization and Engineering*, 5:157–177, 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] C. Audet and J. E. Dennis, Jr. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14(4):980–1010, 2004.

[4] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, and V. Torczon. Optimization using surrogate objectives on a helicopter test example. In J. T. Borggaard, J. Burns, E. Cliff and S. Sherk, eds., Computational Methods for Optimal Design and Control, Birkhauser, Boston, 1998.

[5] J. Borggaard, D. Pelletier, and K. Vugrin. On sensitivity analysis for problems with numerical noise. AIAA Paper 2002–5553, 2002. Presented at the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, Georgia.

[6] J. Burkardt, M. Gunzburger, and J. Peterson. Insensitive functionals, inconsistent gradients, spurious minima, and regularized functionals in flow optimization problems. *International Journal of Computational Fluid Dynamics*, 16(3):171–185, 2002.

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

[8] C. Davis. Theory of positive linear dependence. *American Journal of Mathematics*, 76:733–746, 1954.

[9] J. E. Dennis, Jr., C. J. Price, and I. D. Coope. Direct search methods for nonlinearly constrained optimization using filters and frames. *Optimization and Engineering*, 5:123–144, 2004.

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

[11] E. D. Dolan, R. M. Lewis, and V. Torczon. On the local convergence of pattern search. *SIAM Journal on Optimization*, 14(2):567–583, 2003.

[12] C. H. Edwards. *Advanced Calculus of Several Variables.* Academic Press, 1973. ISBN 0–12–232550–8.

[13] 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.

[14] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981. ISBN 0-12-283950-1.

[15] 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.

[16] R. M. Lewis and V. Torczon. Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941, 2000.

[17] R. M. Lewis and V. Torczon. A globally convergent augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4):1075–1089, 2002.

[18] 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.

[19] A. L. Marsden, M. Wang, J. E. Dennis, Jr., and P. Moin. Optimal aeroacoustic shape design using the surrogate management framework. *Optimization and Engineering*, 5:235–263, 2004.

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

[21] 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.

[22] 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.

[23] V. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Department of Mathematical Sciences, Rice University, Houston, Texas, 1989. Available as Tech. Rep. 90-07, Department of Computational and Applied Mathematics, Rice University, Houston, Texas 77005-1892.

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

[25] V. Torczon and M. W. Trosset. From evolutionary operation to parallel direct search: Pattern search algorithms for numerical optimization. *Computing Science and Statistics*, 29:396–401, 1998.