A Generating Set Search Method Exploiting Curvature and Sparsity*

Lennart Frimannslund† Trond Steihaug‡

Abstract

Generating Set Search methods are one of the few alternatives for optimising high fidelity functions with numerical noise. These methods are usually only efficient when the number of variables is relatively small. This paper presents a modification to an existing Generating Set Search method, which makes it aware of the sparsity structure of the Hessian. The aim is to enable the efficient optimisation of functions with a relatively large number of variables. Numerical results show a decrease in the number of function evaluation it takes to reach the optimal solution, sometimes by significant margins, on noisy as well as smooth problems, for a modest as well as a relatively large number of variables.

Keywords: Nonlinear programming, derivative–free optimization, pattern search, generating set search, sparsity.

1 Introduction

We consider the unconstrained optimisation problem

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

where $f : \mathbb{R}^n \mapsto \mathbb{R}$. Suppose that f is only available as

$$\widetilde{f}(x) = f(x) + \epsilon, \tag{2}$$

where the error term ϵ is either stochastic or numerical in nature. By numerical noise we mean the noise which can arise from, for instance, the discretisation involved if evaluating

*This work was supported by the Norwegian Research Council.

†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

f requires computing an integral, solving a differential equation or any other subproblem which is solved inexactly. The same input will always give the same output, but the function will not be smooth. An example of such a function occurs in [1], where the objective function contains an integral. The truncation error stemming from the computation of the integral makes the function look like the one in figure 1. There is an underlying smooth function, but it is obscured by noise. On such methods derivative-based methods can easily run into trouble, since finite difference-based derivatives may be very inaccurate and automatic differentiation often is unhelpful as well. Generating Set Search (GSS) Methods are a good alternative in this case. GSS methods are comprehensively reviewed in [12]. Although usually easy to implement, GSS methods in their most basic form often converge slowly. Modifications to speed up convergence were suggested as early as in 1960 by Rosenbrock [17]. Two recent approaches using curvature information have been suggested [2, 7]. The main modification to basic GSS in these papers is that the search directions the methods consider are dynamic. The introduction of a dynamic search basis is shown to significantly reduce the number of function evaluations required to reach the optimiser, in most cases.

Apart from slow convergence, GSS methods are often unsuitable for problems where the number of variables n is large. In [16], one proposes a method effective for the optimisation of smooth functions which can be decomposed into *element functions*. Let $\chi_k \subseteq \{1, 2, \ldots, n\}$, $k = 1, \ldots, n$ and let $|\chi_k|$ be the cardinality of the set χ_k. Let $f_k : \mathbb{R}^{|\chi_k|} \mapsto \mathbb{R}$, $k = 1, \ldots, n$, where χ_k are the indices of x on which f_k depend. If f is of the form

$$f(x) = \sum_{k=1}^{n} f_k(x), \tag{3}$$

then f is said to be *partially separable*, or *totally separable* depending on the cardinality of the sets χ_k. Separability of f is closely related to the sparsity structure of the derivatives, but we make the distinction because separability structure is defined even if the function is not differentiable. Theory on separability of functions can be found in [11].

Given a totally separable function one can obtain the value of f at as many as $3^n - 1$ points at the cost of only 2 f-evaluations, as long as the points in question are aligned with the coordinate axes. The optimisation algorithm in [16] exploits this fact to solve smooth problems of the form (1) with f of the form (3) for up to more than 5000 variables. We wish to exploit separability of f, on noisy functions.

In [7] an algorithm which solves (1) where the function is of the form (2) using average curvature information to speed up convergence was developed. However, as n grows, the algorithm becomes increasingly unable to exploit this information. In this paper we present an extension to the algorithm of [7], which utilises the sparsity pattern of the Hessian of f in (2). Although noise can potentially eliminate any sparsity pattern from $\nabla^2 f$ in $\nabla^2 \widetilde{f}$, a priori knowledge about $\nabla^2 f$ through knowledge about the separability structure (3) or known Hessian sparsity structure is assumed to be valid for $\nabla^2 \widetilde{f}$ as well.

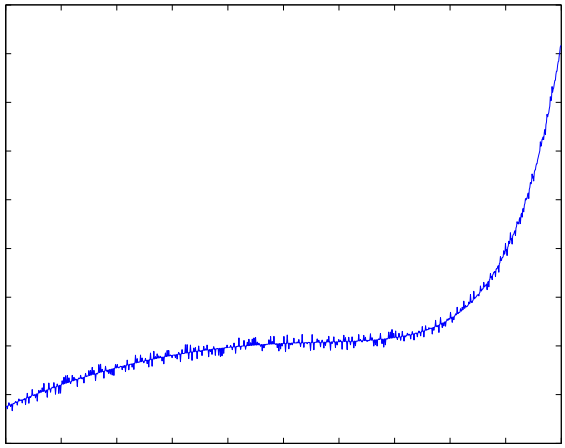

Figure 1: $e^x - x^2$ with noise.

This paper focuses on unconstrained optimisation, but extensions toward constrained optimisation discussed in [4, 13, 14] are applicable.

2 Generating Set Search

GSS methods are a class of methods which search along the vectors of a *generating set* or *positive basis*. A generating set consists of vectors $v_i,\ i = 1, \ldots, r$ such that for any $x \in \mathbb{R}^n$,

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

In words, the vectors in the set *positively span* $\mathbb{R}^n$. It is shown in [3] that to positively span $\mathbb{R}^n$, $n + 1 \leq r \leq 2n$, depending on the vectors. The positive and negative of the Cartesian coordinate vectors, say $e_i,\ i = 1, \ldots, n$ are an example of a generating set with $2n$ vectors. These methods are also known as *pattern search*, the name Generating Set Search was coined in [12].

Let the set of search directions $\mathcal{D}$ be defined as

$$\mathcal{D} = \bigcup_{i=1}^{r} \{p_i\}.$$

Associate with each p_i a step length δ_i. Then, a pseudo code for a method we will call *Compass Search* is:

Compass Search

Given $x,\ \delta_{tol},\ \alpha \geq 1 > \beta > 0,$

Repeat until convergence,

For each $p_i \in \mathcal{D}$,

If $f(x + \delta_i p_i) < f(x)$,

$x \leftarrow x + \delta_i p_i$

$\delta_i \leftarrow \alpha \delta_i$

else,

$\delta_i \leftarrow \beta \delta_i$

end.

end.

end.

α and β need not be constant throughout. We will call one run of the repeat-loop a *sweep*. For this and other GSS methods one can expect linear convergence, see [12] and the references therein.

Rosenbrock's method [17] is based on Compass Search with $2n$ search directions. It regularly rotates the search vectors in $\mathcal{D}$ by aligning the principal search direction to an average gradient and generates $(n-1)$ additional directions through the Gram-Schmidt process. It uses the positive and negative of the resulting vectors as its new search directions.

2.1 GSS Methods Using Curvature Information

We look at two different methods employing curvature information.

The Method of Coope and Price This method for unconstrained optimisation of smooth functions, is described fully in [2]. It minimises the function on successively finer grids which are defined by the search directions $v_i, i = 1, \ldots, n$ and the step lengths associated with each direction. The method searches along both the positive and negative of these directions, and hence has $2n$ search directions. In the process of searching along the current direction, say, v_i, the method obtains the function values at three points along this line. From these three points it creates an interpolating quadratic function. The step length δ_i corresponding to v_i is then based on the distance from the current iterate to the minimiser of the interpolating function.

Using the parallel subspace theorem (see, e.g. theorem 4.2.1 of [6]) the method generates conjugate search directions, one direction at a time from the n initially non-conjugate search directions. Once a conjugate direction has been found, the algorithm deletes a non-conjugate direction, to maintain the number of search directions. The generated conjugate directions are stored in a matrix V_c, which becomes an indirect approximation to $(\nabla^2 f)^{-1}$ once n conjugate directions have been found, by the relation

$$V_c V_c^T \approx (\nabla^2 f)^{-1}.$$

The method is able to perform a finite difference Newton step from time to time. Once the entire inverse Hessian approximation is in place, the algorithm starts building up a new approximation. The algorithm terminates exactly on quadratic functions.

A Method Exploiting Average Curvature Information This method is described fully in [7]. Let the search basis $\mathcal{D}$ consist of the positive and negative of the column vectors of the orthogonal matrix

$$Q = \left[\begin{array}{cccc} q_1 & q_2 & \cdots & q_n \end{array} \right],$$

where q_i is column i. By adaptively shuffling the order of the directions in $\mathcal{D}$ once per sweep, the algorithm is able to gather average curvature information from the history of function evaluations. The algorithm builds up what in [7] is called a *curvature information matrix*, C_Q, one element at the time, by the formula

$$(C_Q)_{ij} = \frac{f(x^{ij} + \delta_i q_i + \delta_j q_j) - f(x^{ij} + \delta_i q_i) - f(x^{ij} + \delta_j q_j) + f(x^{ij})}{\delta_i \delta_j}. \tag{4}$$

where δ_i and δ_j are the step lengths along the search directions q_i and q_j respectively, at any given time. The point x^{ij} is usually different for each $(C_Q)_{ij}$. C_Q is required to be symmetric, so only the lower triangle of C_Q is computed. The expression (4) equals a directional second derivative,

$$(C_Q)_{ij} = q_i^T \nabla^2 f(\widetilde{x}^{ij}) q_j \tag{5}$$

for some $\widetilde{x}^{ij}$ in the rectangle with the four points $x^{ij} + \delta_i q_i + \delta_j q_j$, $x^{ij} + \delta_i q_i$, $x^{ij} + \delta_j q_j$ and x^{ij} as corner points. (See e.g. lemma 3.5 in [5].) If the step lengths are sufficiently large then average curvature information is obtained, thus smoothing out the effects of noise. The method is able to obtain $O(n)$ C_Q-elements per sweep, so the entire matrix C_Q consisting of $\frac{n^2+n}{n}$ unique elements is computed in $O(n)$ sweeps. When C_Q is determined, the matrix C, given by the formula

$$C = Q C_Q Q^T, \tag{6}$$

is computed. The positive and negative of the eigenvectors of C are taken as the new search basis, and Q is updated accordingly.

3 A Scheme for Exploiting Sparsity

We now propose an extension to the algorithm of [7]. Assume f is separable. The individual f_k and χ_k define $|\chi_k| \times |\chi_k|$ Hessian structural information, and by assembling all the individual matrices, we have a sparsity structure for the entire Hessian. If sparsity structure is not known a priori, it can be detected by the technique of [10], or it is possible to obtain the information from computational graphs, which are used in Automatic Differentiation (AD). (See, e. g. [9] for more on AD.)

However, sparsity is relative to the coordinate system. C_Q will not be sparse if $Q \neq I$, and neither will the matrix C from (6) be unless the function is quadratic, due to truncation error in (4). Therefore, we impose the restriction that C have the same sparsity structure as the Hessian.

When $\nabla^2 f$ is full, we need to compute $\frac{n^2+n}{2}$ C_Q-elements by (4). If the Hessian is sparse with, for instance, $O(n)$ unique elements, we would like to compute no more elements in C_Q than there are unique elements in the Hessian itself. $O(n)$ elements can be computed in $O(1)$ sweeps.

We do this by writing (6) as the equation

$$Q^T C Q = C_Q, \tag{7}$$

where the unknown is the matrix C. Let D and B be $n \times n$-matrices. The *Kronecker product* $(D \otimes B)$ is an $n^2 \times n^2$-matrix

$$(D \otimes B) = \begin{bmatrix} D_{11}B & \cdots & D_{1n}B \\ \vdots & & \vdots \\ D_{n1}B & \cdots & D_{nn}B \end{bmatrix}. \tag{8}$$

See e.g. [8]. Useful identities are

$$(D \otimes B)^{-1} = (D^{-1} \otimes B^{-1}), \tag{9}$$

and

$$(D \otimes B)^T = (D^T \otimes B^T), \tag{10}$$

Using the Kronecker product, (7) can be rewritten as

$$(Q^T \otimes Q^T)\mathbf{vec}(C) = \mathbf{vec}(C_Q), \tag{11}$$

where **vec** is an operator $\mathbf{vec} : \mathbb{R}^{n \times n} \mapsto \mathbb{R}^{n^2}$ which stacks the entries of a matrix in a vector such that the equivalence between (7) and (11) holds. Denote the columns of the matrix C by $c_i,\ i = 1, \ldots, n$, that is,

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}.$$

Then

$$\mathbf{vec}(C) = (\ c_1^T \quad c_2^T \quad \cdots \quad c_n^T\ )^T. \tag{12}$$

If we examine the matrix $(Q^T \otimes Q^T)$ it reads

$$(Q^T \otimes Q^T) = \begin{bmatrix} Q_{11}Q^T & \cdots & Q_{n1}Q^T \\ \vdots & & \vdots \\ Q_{1n}Q^T & \cdots & Q_{nn}Q^T \end{bmatrix}. \tag{13}$$

The first row consists of products involving only the elements of q_1. The second row consists of products involving only the elements of q_1 and q_2. Similarly, each of the remaining rows contain products involving elements of only two q-vectors. Since the **vec**-operator is also applied to C_Q in the right-hand side of (11), the row made up of the vectors q_i and q_j corresponds to the element $(C_Q)_{ij}$ in $\mathbf{vec}(C_Q)$. We now want to reduce the number of variables in (11) based on our knowledge of symmetry and sparsity structure. Since we require C to be symmetric we can, for all $r > s$, add the columns corresponding to C_{sr} to the columns corresponding to C_{rs} and delete the former columns. This means we only consider the elements in the lower triangle of C. Accordingly, we delete all the rows which do not correspond to computation of elements in the lower triangle of C_Q. Furthermore, since C has a certain sparsity structure, we can delete all columns which correspond to elements C_{rs} we know are to be zero.

Having removed the columns corresponding to zero elements, we must also remove the same number of rows. We have some freedom when it comes to which rows are to be removed. We want the resulting coefficient matrix after row removal to be well conditioned. If we were working in a Cartesian coordinate system, then the two vectors used to compute C_{rs} by a difference formula like the one in (4) would be the coordinate vectors e_r and e_s, and any nonsingular submatrix of (13) would be well conditioned. Since we are working in the coordinate system defined by the vectors $q_i,\ i = 1, \ldots, n$, the closest we can get to e_r and e_s are the vectors with their maximum absolute elements in position r and s, that is, vectors q_i and q_j such that

$$\max_k |(q_i)_k| = |(q_i)_r|,$$

and

$$\max_k |(q_j)_k| = |(q_j)_s|.$$

So, for each nonzero C_{rs} we pick the vectors q_i and q_j and keep the corresponding row. Let ρ be the number of unique nonzero elements in the Hessian. Since we want an equation system with ρ equations an unknowns, we need to modify the **vec** to take this into account. Let $\overline{\mathbf{vec}}$ be the operator which stacks the nonzero elements of the lower triangle of a matrix in a vector. Let c_Q signify the ρ-vector of C_Q-entries that we compute. The resulting $\rho \times \rho$ equation system becomes

$$A\overline{\mathbf{vec}}(C) = c_Q, \tag{14}$$

where A is the resulting matrix from modifying $(Q^T \otimes Q^T)$. In our experiments, using the heuristic just described, A was usually very well conditioned.

Since we need to compute ρ c_Q-elements and can compute $O(n)$ elements per sweep, the right-hand side c_Q will be available in $O(\frac{\rho}{n})$ sweeps. Then we solve (14) and construct C with the inverse of the operator $\overline{\mathbf{vec}}$.

3.1 The Relationship between C and the Hessian

In this section we examine the error

$$\|C - \nabla^2 f\|.$$

First we need a technical result. Define

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

Then we have

$$\|c\| \leq \|C\|_F \leq \sqrt{2}\|c\|. \tag{15}$$

Too see this, suppose that C has n diagonal and γ off-diagonal nonzero elements. We then have

$$\|c\| = \left(\sum_{i=1}^{n+\gamma} c_i^2 \right)^{\frac{1}{2}}, \tag{16}$$

and

$$\|C\|_F = \left(\sum_{\forall(r,s)} C_{rs}^2 \right)^{\frac{1}{2}}. \tag{17}$$

Not counting terms C_{rs}^2 where C_{rs} is known to be zero, the sum in (17) contains $n + 2\gamma$ nonnegative elements. All of the terms in the sum in (16) are present in (17), so clearly $\|c\| \leq \|C\|_F$. As for the second inequality, we have

$$\sqrt{2}\|c\| = \|\sqrt{2}c\| = \left(\sum_{i=1}^{n+\gamma} (\sqrt{2}c_i)^2 \right)^{\frac{1}{2}}. \tag{18}$$

This can be written

$$\left(2\sum_{i=1}^{n+\gamma} c_i^2 \right)^{\frac{1}{2}} = \left(\sum_{i=1}^{n+\gamma} c_i^2 + \sum_{i=1}^{n+\gamma} c_i^2 \right)^{\frac{1}{2}}. \tag{19}$$

The final sum of (19) contains a sum of $2n + 2\gamma$ nonnegative elements. All the $n + 2\gamma$ elements in (17), (still not counting terms C_{rs}^2 where C_{rs} is known to be zero) are present in (19), so the second inequality of (15) holds as well.

Now we can turn our attention to the relationship between C and the Hessian.

Lemma 1 *Let f be twice continuously differentiable. Assume A in (14) is invertible and let c be the solution to (14). Let element l, $l = 1, \ldots, \rho$ of c_Q in (14) be computed by (4) and be equal to $q_i^T \nabla^2 f(\widetilde{x}^l) q_j$ for the appropriate vectors q_i and q_j by (5). Define*

$$N = \bigcup_{l=1}^{\rho} \left\{ \widetilde{x}^l \right\}, \tag{20}$$

and let

$$\delta = \max_{x,y \in N} \|x - y\|, \tag{21}$$

and

$$\mathcal{N} = \left\{ x \in \mathbb{R}^n \,\middle|\, \max_{y \in N} \|x - y\| \leq \delta \right\}. \tag{22}$$

Let f be Lipschitz-continuous in $\mathcal{N}$ with Lipschitz-constant L. Then, the matrix C obtained by applying the inverse of the operator $\overline{\mathbf{vec}}$ on c, satisfies

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

where $x \in \mathcal{N}$ and $\kappa(A)$ is the condition number of A.

Proof. Let $h_l = \overline{\mathbf{vec}}(\nabla^2 f(\widetilde{x}^l)),\ l = 1, \ldots, \rho$. The Hessian has the same sparsity structure as C, so c_Q can be written

$$c_Q = \begin{bmatrix} (Ah_1)_1 \\ (Ah_2)_2 \\ \vdots \\ (Ah_\rho)_\rho \end{bmatrix},$$

where $(Ah_l)_l$ is the lth element of the vector Ah_l. If we now let E_l be the matrix with 1 in position (l, l) and zero everywhere else, we have

$$c = A^{-1} \sum_{l=1}^{\rho} (E_l A h_l).$$

The Hessian mapping $\nabla^2 f : \mathbb{R}^n \mapsto \mathbb{R}^{n \times n}$ is assumed to be Lipschitz-continuous in $\mathcal{N}$, that is,

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq L \|x - y\| \quad \text{for all } x, y \in \mathcal{N}. \tag{23}$$

Let $x \in \mathcal{N}$. Define

$$\overline{\mathbf{vec}}(\nabla^2 f(x)) = h.$$

Then we have

$$c = A^{-1} \sum_{l=1}^{\rho} (E_l A (h + \epsilon_l)),$$

where

$$\epsilon_l = h_l - h.$$

This expands to

$$c = A^{-1}(E_1 + \cdots + E_\rho)Ah + \sum_{l=1}^{\rho} A^{-1}E_l A\epsilon_l.$$

The first part of the expression reduces to just h, since the sum of the E_l becomes the identity matrix. The second term becomes an error term, whose norm is bounded by

$$\|c - h\| = \|\sum_{l=1}^{\rho} A^{-1}E_l A\epsilon_l\| \leq \rho\|A^{-1}\| \left(\max_l \|E_l\|\right) \|A\| \left(\max_l \|\epsilon_l\|\right). \tag{24}$$

All the E_l have unit norm, and the norms $\|A\|$ and $\|A^{-1}\|$ together make up the condition number of the matrix A, $\kappa(A)$. We now need a bound on $\max_l \|\epsilon_l\|$. We have

$$\max_l \|\widetilde{x}^l - x\| \leq \delta,$$

since x and all the $\widetilde{x}^l$ are in $\mathcal{N}$. Thus, by (23):

$$\max_l \|\widetilde{x}^l - x\| \leq \delta \Rightarrow \max_l \|\nabla^2 f(\widetilde{x}^l) - \nabla^2 f(x)\| \leq L\delta.$$

By (15) we have

$$\max_l \|\epsilon_l\| = \max_l \|h - h_l\| \leq \max_l \|\nabla^2 f(\widetilde{x}^l) - \nabla^2 f(x)\|_F \leq L\delta.$$

This turns (24) into

$$\|c - h\| \leq \rho\kappa(A)L\delta,$$

and finally, by (15),

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

□

4 Preliminary Numerical Results

Numerical test were performed on three functions from [15], for various sizes of n. All the functions have a minimum value of zero. The results on smooth functions are listed in table 1. The columns contain, from left to right, the number of variables, the number of unique nonzero elements to be determined ρ, the number of function evaluations performed to reach the solution, the number of C-matrices computed and hence the number of times the positive basis $\mathcal{D}$ is updated, and the final function value obtained, for the method

using sparsity and the method of [7] (marked "regular" in the table), respectively. The convergence criterion used in the experiments on smooth functions was

$$\max_i \delta_i < 10^{-7}.$$

The results on the extended Rosenbrock function agree very well with our expectations. The Hessian of the extended Rosenbrock function has $O(n)$ elements, so as expected the number of C-matrices and hence $\mathcal{D}$-updates is relatively constant for the sparse method, consistent with the bound $O(\frac{p}{n})$ for obtaining the desired C_Q-elements. In the case of the regular method, $\mathcal{D}$-updates become fewer as n grows, consistent with the bound $O(n)$ on the computation of C_Q in this case. In addition, the sparse method uses fewer function evaluations to reach the optimum, apparently since it is able to change search basis and hence adapt to the landscape of the function more often than the regular method.

On the Broyden tridiagonal function we see a similar picture, although the savings in function evaluations are not as apparent here as on the extended Rosenbrock function. The reason this seems to be that frequent basis updates is not crucial on this function. The same can be said about the results on the Broyden banded function. Note that on the two Broyden functions, when $n = 64$ and $n = 128$, no basis change takes place in the case of the regular method, which then in reality becomes Compass Search.

We also tested on the functions with noise, specifically

$$\widetilde{f}(x) = f(x) + \max(10^{-4} \cdot |f(x)|, 10^{-4}) \cdot \mu, \tag{25}$$

where μ is uniformly distributed in the interval $[-1, 1]$. This noise scheme is adopted from [18]. On these problems, the convergence criterion used was

$$\max_i \delta_i < 10^{-4}.$$

The results are listed in table 2. Since we add noise to the problems by (25) we cannot expect to find a lower function value than 10^{-4}. On the extended Rosenbrock function the picture is very much the same as with no noise. However, the regular method terminates prematurely for n equal to 32, 64, and 128. The sparse method terminates prematurely for $n = 128$. On the Broyden functions we also have the same picture as when no noise is added.

5 Concluding Remarks

We have proposed and extension to the algorithm of [7] to make it aware of sparsity, and thereby enable solution of problems with n relatively large. We have managed to reduce the number of function evaluations it takes to reach a minimum on all three test functions as n grows. The results hold promise, and much can be done to improve the results still, for

Extended Rosenbrock Function

	Sparse				Regular		
n	ρ	#feval	#Basis	f^*	#feval	#Basis	f^*
4	6	893	16	1.53e-15	1051	14	3.52e-13
8	12	1972	18	5.89e-16	2870	11	3.26e-16
16	24	3669	17	1.99e-15	8128	8	9.62e-16
32	48	7368	17	3.65e-15	20632	6	2.77e-15
64	96	14849	17	1.63e-15	65284	4	1.18e-14
128	192	29781	17	3.26e-15	190884	3	2.13e-13

Broyden Tridiagonal Function

	Sparse				Regular		
n	ρ	#feval	#Basis	f^*	#feval	#Basis	f^*
4	7	355	6	1.53e-13	365	5	4.62e-13
8	15	826	7	2.59e-13	781	3	1.20e-13
16	31	1556	6	8.25e-13	1672	2	7.97e-13
32	63	3384	7	4.09e-13	4153	1	7.52e-14
64	127	6440	7	1.70e-12	9186	0	1.42e-12
128	255	14997	8	1.41e-12	18879	0	8.61e-12

Broyden Banded Function

	Sparse				Regular		
n	ρ	#feval	#Basis	f^*	#feval	#Basis	f^*
4	10	457	6	5.08e-15	382	5	2.56e-13
8	35	824	3	1.36e-14	804	3	4.28e-13
16	91	1667	3	5.05e-14	1682	2	2.34e-13
32	203	3439	2	6.90e-13	3437	1	9.31e-13
64	427	6709	2	1.45e-12	7524	0	1.76e-13
128	875	13450	2	2.24e-12	15070	0	3.96e-13

Table 1: Numerical results, smooth functions.

Extended Rosenbrock Function

	Sparse				Regular		
n	ρ	#feval	#Basis	f^*	#feval	#Basis	f^*
4	6	808	15	3.63e-5	874	11	3.52e-4
8	12	1635	15	2.67e-3	2251	9	1.31e-4
16	24	3113	14	2.36e-2	7556	8	4.35e-3
32	48	7014	14	2.36e-2	10623	3	3.06e1
64	96	14085	16	1.38e-1	5236	0	1.22e2
128	192	29321	17	1.86e1	6629	0	2.49e2

Broyden Tridiagonal Function

	Sparse				Regular		
n	ρ	#feval	#Basis	f^*	#feval	#Basis	f^*
4	7	182	3	5.25e-5	220	3	6.71e-5
8	15	383	3	3.66e-5	400	2	9.09e-5
16	31	855	4	1.86e-4	923	1	1.98e-4
32	63	1710	4	6.69e-4	1955	0	9.15e-4
64	127	3436	4	1.03e-4	4460	0	2.03e-3
128	255	6834	4	1.70e-3	8146	0	4.81e-3

Broyden Banded Function

	Sparse				Regular		
n	ρ	#feval	#Basis	f^*	#feval	#Basis	f^*
4	10	205	3	1.73e-5	264	4	2.70e-5
8	35	460	2	5.76e-5	434	2	7.89e-5
16	91	893	1	1.13e-4	925	1	1.50e-4
32	203	1687	1	1.93e-4	1885	0	2.53e-4
64	427	3734	1	2.81e-4	3791	0	7.56e-4
128	875	6799	1	8.60e-4	7504	0	1.33e-3

Table 2: Numerical results, noisy functions.

instance incorporating ideas like the one in [16] mentioned in the introduction, and dealing with the great number of technical issues which arise when converting the algorithm of [7] to handle sparse Hessians.

References

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

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

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

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

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

[6] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons Ltd., 1987. Second Edition, ISBN 0–471–91547–5.

[7] Lennart Frimannslund and Trond Steihaug. A generating set search method using curvature information. To appear, 2004.

[8] Alexader Graham. *Kronecker Products and Matrix Calculations with Applications*. Halsted Press, John Wiley and Sons, New York, 1981. ISBN 0470273003.

[9] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000. ISBN 0–89871–451–6.

[10] Andreas Griewank and Christo Mitev. Detecting jacobian sparsity patterns by bayesian probing. *Mathematical Programming*, 93(1):1–25, 2002.

[11] Andreas Griewank and Philippe L. Toint. On the unconstrained optimization of partially separable functions. In Michael J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312. Academic Press, New York, NY, 1982.

[12] Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[13] Robert Michael Lewis and Virginia Torczon. Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941, 2000.

[14] Robert Michael Lewis and Virginia 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.

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

[16] C. P. Price and P. Toint. Exploiting problem structure in pattern search methods for unconstrained optimization. Technical Report 2004/3, Mathematics and Statistics department, Canterbury University, Christchurch, New Zealand, 2004.

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

[18] Virginia 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.