On Curvature and Separability in Unconstrained Optimisation

Lennart Frimannslund

Thesis for the degree of Philosophiae Doctor (Ph.D.)
Department of Informatics
University of Bergen
Norway

March 2006

Acknowledgments

I would like to thank my principal advisor over the past years, professor Trond Steihaug, without whose blend of patience, impatience, competence and professionalism this thesis would never have been completed. Thank you! Thanks also to my second advisor, professor Hans Julius Skaug, for being patient and treating me more like a peer than a student.

On the personal side, thanks to my girlfriend, family and friends, for believing in me and supporting me, not least at the times when I didn't deserve it.

Notation

In this thesis we use superscripts to denote the iteration a variable corresponds to. We use subscripts to denote variable names and indices. If a variable has more than one item of information which is to appear in the same position, we use parentheses. Some examples are given below:

x^k	The variable x, at iteration k.
x_*	The optimal value of x.
$(C_Q)_{ij}$	Element ij of the matrix C_Q.
$(y^k)^T$	The transpose of the variable y at iteration k.
e_1	The first unit coordinate vector.
$(q_r)_i$	Element i of the vector q_r.

In addition, we sometimes use the notation x^{ij} in conjunction with a matrix element $(C_Q)_{ij}$, which means x^{ij} is a vector, which has a connection with element ij of the matrix C_Q.

Contents

Chapter 1

Introduction

Optimisation is a diverse field, where the only common denominator of all its incarnations can be said to find the best choice in a situation where one has several possibilities to choose from. For instance, one might be adjusting the ratio of components to make a chemical process run as smoothly as possible, or the properties of physical object such as its shape, mass distribution and stiffness. One might be trying to make a production line run more efficiently, increase the revenue of a factory, or try to produce the largest number of clothes from a given amount of fabric or any other similar situation. To give an example, consider the problem of a farmer looking for a site to build a farm. The farm must have irrigation, with water supplied from a nearby reservoir, electricity from a local power station, and road access to a processing plant, where the products of the farm will be delivered. There is a cost associated with building a water pipeline, installing cables for electricity and paving a road to the processing plant. The farmer wants to build the farm in the site which corresponds to the lowest cost, so the optimisation problem can be formulated as

$$\min_{\text{Farm position}} \text{Pipeline cost} + \text{cable cost} + \text{road cost}. \tag{1.1}$$

As one learns from calculus, extreme values are found where the gradient vanishes. Apart from cases involving very simple objective functions one needs iterative solution methods. Many methods are implicitly based on Taylor's theorem by constructing polynomial model functions. Newton's method assumes a quadratic model function, and steps to the minimum of that function. A third-order tensor method assumes a cubic model function, and similarly steps to the minimum of that function. Since Taylor's theorem tells us that the higher the degree of the polynomial, the better we can approximate an arbitrary function, one might expect that methods of this kind would become more and more powerful as their order increases. However, trying to build the highest-order method possible would present new problems along the way, for instance that:

1. The desired derivatives may be expensive, or difficult to compute.

2. The objective function might not be differentiable, or yield unhelpful derivatives.

3. The desired derivatives may be expensive to store.

The fact that derivatives may be expensive to compute is addressed by quasi-newton methods like the BFGS method (see e.g. [50] and the references therein), which make use of gradient information from past iterations to produce an approximation to the Hessian. Compared to Hessian matrices, gradients can be computed relatively cheaply using Automatic Differentiation (AD) techniques, see e.g. [36]. However, if gradients are not directly available, for example if the function is written in several programming languages, precluding the use of AD, or is for some other reason only available as a "black box"-procedure, one might have to turn to other alternatives.

If the objective function is the result of a simulation on a computer, then the resulting function might well not be smooth, although the same input will always give the same output except in the case of random effects. For example, determining the time of day it is fastest or slowest for a cargo train to get from one city to another, given ordinary traffic that uses the same network of rails, is a problem which can be cast as a continuous optimisation problem, since the input variable (start time) and output variable (travel time) are continuous. Since two trains cannot occupy the same piece of rail at the same time, there could be situations where the cargo train would have to wait for oncoming trains, or wait until a track is cleared for some other reason. This could result in the objective function resembling a the profile of a staircase, like in Figure 1.1, which would make it a non-differentiable function in some parts of its domain and the gradient zero wherever it is defined, which would certainly be a challenging function for a gradient-based method. A different example — simulations might be used in weather forecasting. One could if one were to calibrate a model for weather forecasting based on historical weather data, encounter an objective function which would probably involve solving a differential equation. Differential equations are solved using iterative methods, such as for example Runge-Kutta methods. These methods inevitably involve some sort of discretisation, which can result in that even if there exists an underlying smooth function in the mathematical sense, the numerical function the optimisation method works with appears as non-smooth. This can also happen in the context of flow optimisation, e.g. [8]. To generalise this kind of functions, suppose we are dealing with a function that is very expensive to compute accurately, but which can be computed approximately at a lower cost. Such a function could, apart from differential equations for instance contain integrals. High-dimensional integrals can be expensive to compute, especially if they cover a broad region. One might solve the integral using a coarse discretisation, which would make the integral – and in turn the function – relatively cheap to compute, although one would then only obtain an approximation to the underlying function value. A third example, the objective function itself could contain an "inner" optimisation problem, so that evaluation of the objective function would mean solving an optimisation problem in the process. The inner problem could be solved to varying degrees of accuracy,

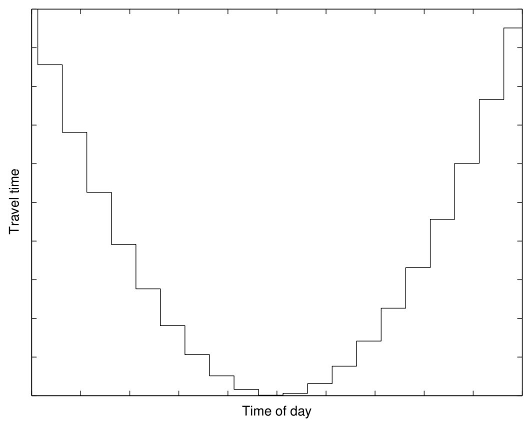

Figure 1.1: Cost function where the derivative is not always defined.

which would again influence computation time and accuracy of the final numerical objective function value. In chapter 6 we will encounter a function of this type, where the inner optimisation is part of an approximation to an integral.

If the function values are expensive to obtain accurately, then it is not hard to imagine that the same is the case for the derivatives. If the function when computed to limited accuracy is not smooth, it is not necessarily clear what an inaccurately computed derivative represents or if it contains useful information about the underlying smooth objective function at all. In an example in flow optimisation [9], a discretisation in the computed objective function causes the corresponding computed gradient to be inconsistent with the underlying mathematical function, and causes gradient-based optimisation methods to terminate at non-stationary points.

In all of the cases mentioned one can turn to derivative-free optimisation methods. These methods in their most basic form use only function values. This is a strength, since it makes these methods robust in the sense that they in practice can perform very well on functions which are not differentiable, even though for many of these methods there is often no theoretical guarantee for convergence on such functions. The fact that these methods only use function values is also a source of potential weakness, since these methods in their simplest form typically converge very slowly, unless the objective function by chance is particularly well suited to the method at hand.

When we talk about expensive functions, the case might also be that the objective function is so expensive to evaluate that one may only perform a limited number of evaluations, since evaluating the function more times is impractical, or even impossible. If this is the case one often works with a surrogate model function instead, which is in part based on the values of the actual objective function. In surrogate modeling derivative-free methods which we will be concerned with can play a part (see e.g. [28, 6, 7]), although we will not investigate this kind of modeling directly.

That derivatives are expensive to store is an obstacle that *limited memory methods* attempt to deal with. The gradient of the objective function has n elements, which allows for quite large n on today's computers, although concepts like small and large will of course always be relative. A Hessian matrix has $n(n+1)/2$ unique elements, and the quadratic growth of the storage requirements as n grows puts a limit on the size of variables it is practical to work with. Even more so in case of the third derivatives, which have $n(n+1)(n+2)/6$ unique elements. In this case one often tries to exploit sparsity if it is present, which can reduce the need for storage. Even if the derivatives are not sparse, if the function is separable then one it might still be possible to cut back on storage requirements, for example in the case of a Hessian matrix where all the diagonal elements are always equal to each other, and all the off-diagonal elements are equal to each other as well. In this case, even if the Hessian were of infinite size, it would only contain two unique elements and could be stored using only two floating point numbers. If one cannot cut storage requirements through separability, one may use a limited number of lower-order derivative values to approximate higher-order derivatives. Two important methods which do this are the limited memory BFGS (L-BFGS) method [54, 48] and the discrete Newton method [27, 56]. Both these methods use gradients to approximate Hessian information. The L-BFGS method is a variant of the BFGS method. Whereas the storage requirement for BFGS is $O(n^2)$, the L-BFGS method uses only $O(mn)$ space, where m is a variable which can be controlled by the user. The discrete truncated Newton uses the property of most iterative methods for solving linear equations that they do not need an explicit representation of the coefficient matrix, in this case the Hessian. Instead, a iterative method usually only needs the ability to compute the product of the Hessian with an arbitrary vector. Such a product one can approximate using two values of the gradient. Newton methods which use iterative equation solvers usually do not solve the Newton step equations to full precision at each iteration, and hence are called (discrete) truncated Newton methods.

What we wish to investigate in this thesis is how we can speed up methods that deal with the three points listed. To see how this can be done, let us put ourself in the place of an optimisation method for a moment. If one as a human being were to determine what is the best choice in a situation where one has several or infinite options to choose from, one would probably do some research before one starts experimenting, and memorise one's experience during the testing phase. Optimisation methods are little different from humans in this respect, they can make use of information about the problem provided to them before they start evaluating the function. We will make use of the separability properties of the objective function, which are sometimes available before the optimisation process starts. Similarly, optimisation methods can remember what has happened in earlier iterations, and use for instance previous function values and gradients to make an informed decision about where to search for the next solution. We intend to use curvature information to this end. Thus, we will try to improve existing methods and we will keep in mind that objective functions are not always smooth, even if the underlying mathematical representation might be. Specifically, we will deal with generating set search methods, which deal with potentially non-smooth functions and functions where the derivatives are difficult to obtain, and hence address points 1 and 2, and limited memory methods which address point 3. If the function is not smooth, we will use average curvature information to try to capture the curvature properties of the underlying function. For example, in the case of the function in Figure 1.1, this would mean capturing the curvature properties of a quadratic function. As for limited-memory methods, although the L-BFGS method approximates curvature over time, previous research has shown that its initial curvature approximation is very important when it comes to ensuring quick convergence. We will try to enhance this method by trying to make use of second derivative information, that is, given that we can compute the product of the Hessian with an arbitrary vector, we will try to make L-BFGS exploit this information in its initial curvature approximation.

The thesis is organised as follows. In chapter 2 we introduce unconstrained optimisation, briefly review derivative free methods, introduce generating set search methods, and reiterate which types of functions we will be concerned with. In chapter 3 we show how a generating set search method can be made to reach the optimal solution faster, by making it detect and exploit average curvature information. In chapter 4 we introduce the concept of separability, and show how it is not necessarily related to differentiability. We extend the improved generating set search method of chapter 3 to take separability into account. In chapter 5 we introduce quasi-Newton methods and limited memory methods, particularly L-BFGS and the (discrete) truncated Newton method. We present a new class of hybrid methods which encompasses L-BFGS and truncated Newton. The new class can be viewed either as a class of L-BFGS methods enhanced by curvature information, or as a class of truncated Newton methods enhanced by using previously acquired gradient values. Chapter 6 contains an application of the generating set search method of chapter 3 to a concrete problem from statistics, and chapter 7 offers some concluding remarks and proposes some possible avenues of further research. The five papers themselves appear at the end of this thesis.

Chapter 2

Generating Set Search

In this thesis we will address problems of the form

$$\min_x f(x), \tag{2.1}$$

where

$$f : \mathbb{R}^n \mapsto \mathbb{R}.$$

The function may or may not be differentiable, and may or may not be continuous. We assume that the function is not stochastic, that is, the same input always gives the same output. We will try to obtain *local* minima of the objective function f from an arbitrary starting point, we will not try to predictably obtain global minima.

We can define a concrete version the farm problem (1.1) to obtain a problem of the form (2.1). Let the processing plant be at the origin, let the water reservoir be at position $x_w = (100, 0)$ and the power plant be at position $x_{pow} = (150, 50)$, where the units along the axes are kilometers. Let the cost of of cable to the power plant be €7000 per kilometer, the cost of a pipeline from the water reservoir be €8000 per kilometer, and the cost to build a road to the processing plant be €9000 per kilometer. Let the position of the farm be x_f. Again, where is the cheapest site to build the farm? We can model this as

$$\min_{x_f} \quad 9000 \cdot \|x_f\|_2 + 8000 \cdot \|x_f - x_w\|_2 + 7000 \cdot \|x_f - x_{pow}\|_2. \tag{2.2}$$

This is a smooth optimisation problem, with well-defined derivatives except at the origin, x_w and x_{pow}. In real life on the other hand, the costs involved can depend on things like the nature of the terrain, cost of subcontracting, available manpower, overtime payment — simply put any number of problems one can think of. In other words, although one would like the cost function for for example the water pipeline to look like the lowermost curve in Figure (2.1), it might well look like the staircase-curve in the middle, or even the uppermost noisy curve, in complicated cases. In all three curves of the figure we have an

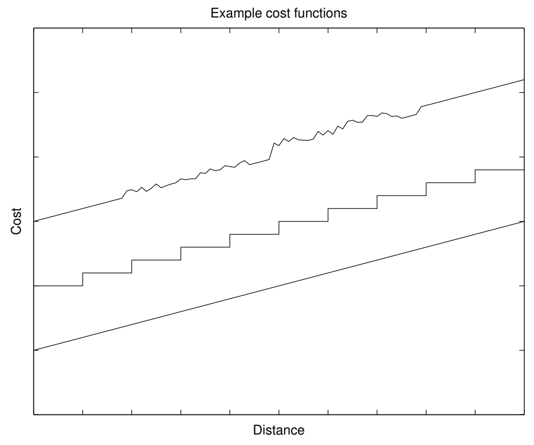

Figure 2.1: Example cost functions.

underlying, simple function, but the two uppermost curves present potential problems for gradient-based methods.

If the function is challenging for gradient-based methods, there are many derivative-free method classes to choose from. One may for instance sample the function at particular points, and construct an interpolating model function. Then, one could minimise the interpolating function, and evaluate the objective function at the optimum of the model function, construct a new model function, and repeat the process. Polynomials, either linear or quadratic are often the interpolating functions of choice in this context. Using this method one might get an interpolating function which is a poor approximation to the objective function, especially far from the current best point. An extension to the above strategy is to consider the model function to be valid only within a specific radius of the current best point, and to search for the minimum of the model function within this region only. This is called a trust region approach. Some of these methods, including linear model functions are reviewed in [16, 59, 61]. Methods that use quadratic approximations include those in [18, 60, 62, 14], a recent method with both a linear and a quadratic variant is given in [51]. A recent extension towards separable function is given in [15].

A different approach entirely is that of simplex methods. One of the most widely cited and popular simplex methods is the method of Nelder and Mead [52]. The Nelder-Mead method in n dimensions keeps a simplex of $n+1$ points, and changes one of these points per iteration. We will not give a full description of the method, but highlight one of its features, which is of interest to us. In Figure 2.2 we see what is called an outside contraction, in $\mathbb{R}^2$. The three points of the triangle with dashed sides are the points corresponding to, say, iteration k, and the three points of the triangle with solid sides correspond to iteration

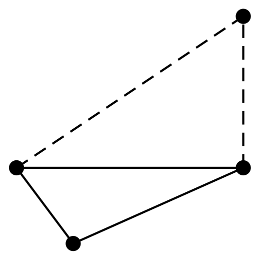

Figure 2.2: An outside contraction in the Nelder-Mead method.

$k+1$. In other words, the uppermost point is substituted with the lowermost point in the figure. Similarly, the algorithm can also perform what is called a reflection and expansion step, which in the figure corresponds to the solid triangle belonging to iteration k and the dashed triangle to iteration $k+1$. In any event, what we wish to highlight is that the shape of the simplex is not constant, since the angles between the sides of the simplex can vary. This can cause the shape of the simplex to be for instance very long and narrow, which in turn can cause the method to stagnate. Torczon [68] demonstrated that the direction of progress of the Nelder-Mead method can become nearly orthogonal to the negative gradient for even the simplest functions. Torczon introduced a new method, the multi-directional search method, which just like the Nelder-Mead method keeps a simplex of $n+1$ points in n dimensions, but which does not allow the angles between its sides to change. The numerical results in [68] show the multi-directional search method to perform better than the Nelder-Mead method.

That the function may cause the points the Nelder-Mead method keeps to lie close to each other is a phenomenon which can also affect trust region interpolation methods in the form we have described them above. The reason for this is that we have not imposed any restrictions on the geometry of the points we include in our model, so for a given function, the equations that define the interpolating function might be singular, or near-singular, as pointed out in [17]. Interpolation methods can be made to enforce a certain geometry, which eliminates this particular problem.

As far as derivative-free methods are concerned, we wish to develop methods that enforce a certain geometry on its points and search directions, but do not wish to do this within the a trust-region framework, that is, we do not want an explicit model function. It is then natural to turn to methods like the multi-directional search algorithm.

This method is in the class of what is called generating set search (GSS) methods. Until the review article [42] coined this name (GSS) there seems to have been some overlapping use of terminology in this field. In the review article [73], both the Nelder-Mead method and the multi-directional search algorithm are called "direct search methods", Torczon et al use the name "pattern search", in e.g. [70, 71, 30, 47] to signify (a subset of) GSS methods, whereas Conn and Toint in [18] use the name "pattern search" for both the Nelder-Mead method and the multi-directional method. Exactly how to define these methods has not been an easy task either, leading to headings like *"I know it when I see it: Toward a definition of direct search methods"* [72] and indirect definitions like that a direct search method only uses function values and does not "in its heart" develop an approximate gradient [73].

A framework encompassing all the GSS algorithms we will be concerned with was given in [42], and we will stick with the name and definitions of that review. GSS methods are in their simplest form very intuitive methods. As stated in the preface of [25], one such method was used when one first started to optimise functions on computers. Two issues however, that for many years made direct search methods an unattractive option for many users were as pointed out in for instance [67, 42], the lack of convergence theory, and the fact that the methods often converge slowly in practice. The multi-directional search algorithm was presented along with a proof of convergence [68, 69], and this convergence theory was generalised in [70], which also presented a unifying framework for a particular class of GSS algorithms. This class is called generalised pattern search (GPS) algorithms in [4], where the convergence theory of [44, 45, 70] was unified. GPS with derivative information was studied in [1], and GPS with inexact derivative information in [23].

Convergence theory for GSS methods we will call moving grid methods was presented in [21] and for what we will call sufficient decrease methods in [49]. Extensions toward constrained optimisation have been discussed in [29, 44, 45, 46], and an algorithm designed specifically for noisy problems appeared in [2].

The methods for which [70, 4] provides convergence theory are methods which restrict all the iterates to lie on a rational lattice, which essentially is a regular grid in n dimensions. Mathematically, this can be stated as that given an initial iterate x^0, search directions $d_i,\ i = 1, \ldots, r$ and initial corresponding step lengths δ_i, then any iterate on x^k on a given lattice defined by d and δ, satisfies

$$x^k = x^0 + \sum_{i=1}^{r} \zeta_i \delta_i d_i, \tag{2.3}$$

where the coefficients ζ_i are all nonnegative integers. The lattice can be successively refined according to rules imposed by the convergence theory, for simplicity one can imagine the refinement as restricted to halving the step lengths δ. Since the resulting lattice is rational, the convergence result becomes that

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

This result cannot always be strengthened, as discussed in [3].

Moving grid methods can be viewed as an extension to rational lattice methods. Moving grid methods allow the grid of points to be reconfigured once a grid minimum is found. A grid minimum is a point on a fixed grid which corresponds to a lower objective function value than all of its neighbouring points. This reconfiguration can be for instance a rotation, scaling, a shear transformation or a combination of some or all of these transformations. (For a definition of shear, see e.g. [32] section 5.2.)

Sufficient decrease methods do not impose restrictions on grids, as long as a new point is only accepted if it produces sufficient decrease, that is,

$$f(x_{\text{new}}) \leq f(x_{\text{old}}) - \rho(\delta),$$

where

$$\delta = \|x_{\text{new}} - x_{\text{old}}\|,$$

and

$$\rho(\delta) = o(\delta), \text{ as } t \downarrow 0.$$

All of these results where summarised in [42], which also credits [74] and [43] for the convergence results. Central to all GSS methods is the presence of a generating set, or positive basis as it is also called. A generating set is a set of vectors $v_i,\ i = 1, \ldots, r$ such that for each x in $\mathbb{R}^n$, we have

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

This requires [26] that $r \geq n + 1$. A GSS method uses these directions as its search directions, or search basis. We will, for the rest of this thesis only consider methods where $r = 2n$, where the search directions are the positive and negative of the columns of an orthogonal matrix Q, unless otherwise stated. We will occasionally state results for methods which search along the positive and negative of the columns of an $m \times n$ matrix P.

A very simple example of a GSS method which we will call compass search, can be written as in Figure 2.3. From the code one can gather what is meant by an iterative method. The method takes an approximation to, or guess of the optimal solution x^k, and produces a better approximation x^{k+1}. Then the process repeats, and can be repeated until no better function value is found, or one is satisfied with the current approximation. There are several aspects of this method which can be decided upon by the user, for instance which directions $\mathcal{G}$ should contain, in which order the method should search along the directions, as well as when step lengths are increased and decreased. It is common to halve the step lengths if no search progress is made, either by halving all the step lengths if no progress is made along any direction, or treating the directions individually and halving the step lengths corresponding to directions with no progress. One can also increase the step lengths, for instance by doubling them, if for instance the method has stepped along a certain direction or certain directions many times in a row. There are a few technical requirements for the method to be

```
Given $f$, $k = 0$, $x^k$, search directions $\mathcal{G}$, step lengths $\delta_i$,
While $\max_i \delta_i >$ threshold
    Set $x_+ \leftarrow x^k$,
    For each direction $d_i$ in $\mathcal{G}$, with associated step length $\delta_i$
        If $f(x_+ + \delta_i d_i) < f(x_+)$
            Set $x_+ \leftarrow x_+ + \delta_i d_i$.
    end.
    Set $x^{k+1} \leftarrow x_+$.
    Update step lengths $\delta_i$
    Set $k \leftarrow k + 1$.
end.
```

Figure 2.3: Pseudocode for Compass Search.

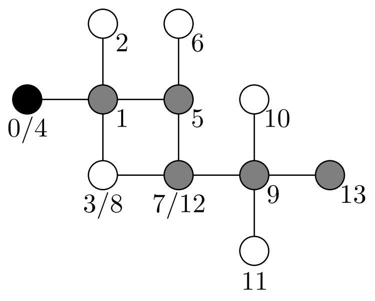

Figure 2.4: Progress of Compass Search.

convergent, most importantly that the set of search directions multiplied by all possible step lengths spans the entire space of interest.

An illustration of how the method searches in $\mathbb{R}^2$ when $\mathcal{G}$ consists of the positive and negative of the unit vectors is given in Figure 2.4. In the figure each point is marked by a node. The directions in $\mathcal{G}$ are searched in the order east, north, south, west. In addition, step lengths are unchanged as long as there is progress along one of the four directions. The search starts at the black node, and the points are evaluated in the order they are numbered. First, the method searches to the right/east, finds a better function value, and steps to the grey node. Then, it searches upward/north, but does not find a better value, which is marked by a white node. Then it searches to the south/down, but does not find a better value. Then it searches to the left/west, and tests the original black point again, but does not step since the starting point does not give a lower function value. In the form given in Figure 2.3 the method would perform this evaluation twice, hence it is marked 0/4. Then the cycle repeats, the method tests in the order east, north, south, west. Note that the points marked 3/8 and 7/12 are also visited twice.

We can apply compass search to the farm problem (2.2). Let us take the point $x_f = (50, 50)$ as the initial guess. We use the positive and negative of the unit coordinate vectors as the set $\mathcal{G}$, the same step length δ for all directions in $\mathcal{G}$, and halve δ if we cannot find reduction along any direction. Initially $\delta = 10$. The result of the first 25 iterations are given in Table 2.1. As one can see in the table,

Iteration	f	δ	$(x_f)_1$	$(x_f)_2$
0	1.90208e+06	1.00000e+01	5.00000e+01	5.00000e+01
1	1.85853e+06	1.00000e+01	4.00000e+01	5.00000e+01
2	1.83126e+06	1.00000e+01	3.00000e+01	5.00000e+01
3	1.82114e+06	1.00000e+01	2.00000e+01	4.00000e+01
4	1.82114e+06	5.00000e+00	2.00000e+01	4.00000e+01
5	1.82114e+06	2.50000e+00	2.00000e+01	4.00000e+01
6	1.82093e+06	2.50000e+00	2.25000e+01	4.00000e+01
7	1.82080e+06	2.50000e+00	2.25000e+01	4.25000e+01
8	1.82080e+06	1.25000e+00	2.25000e+01	4.25000e+01
9	1.82074e+06	1.25000e+00	2.12500e+01	4.12500e+01
10	1.82074e+06	6.25000e-01	2.12500e+01	4.12500e+01
11	1.82071e+06	6.25000e-01	2.18750e+01	4.12500e+01
12	1.82071e+06	3.12500e-01	2.18750e+01	4.12500e+01
13	1.82071e+06	3.12500e-01	2.18750e+01	4.15625e+01
14	1.82071e+06	1.56250e-01	2.18750e+01	4.15625e+01
15	1.82071e+06	1.56250e-01	2.18750e+01	4.14062e+01
16	1.82071e+06	7.81250e-02	2.18750e+01	4.14062e+01
17	1.82071e+06	7.81250e-02	2.17969e+01	4.14844e+01
18	1.82071e+06	7.81250e-02	2.17969e+01	4.14062e+01
19	1.82071e+06	3.90625e-02	2.17969e+01	4.14062e+01
20	1.82071e+06	3.90625e-02	2.17969e+01	4.14453e+01
21	1.82071e+06	1.95312e-02	2.17969e+01	4.14453e+01
22	1.82071e+06	1.95312e-02	2.18164e+01	4.14258e+01
23	1.82071e+06	9.76562e-03	2.18164e+01	4.14258e+01
24	1.82071e+06	9.76562e-03	2.18164e+01	4.14355e+01
25	1.82071e+06	4.88281e-03	2.18164e+01	4.14355e+01

Table 2.1: First 25 compass search iterations on problem (2.2).

the sequence of iterates tends towards the point $x_f = (21.81\ldots, 41.43\ldots)$, and δ declines as the method closes in on this point. The corresponding placement of the farm is visualised in Figure 2.5.

Informally, one can say that GSS methods converge linearly. If we call the white nodes in Figure 2.4 $x^l_{\text{unsuccessful}}$, where l is an index which is updated as the

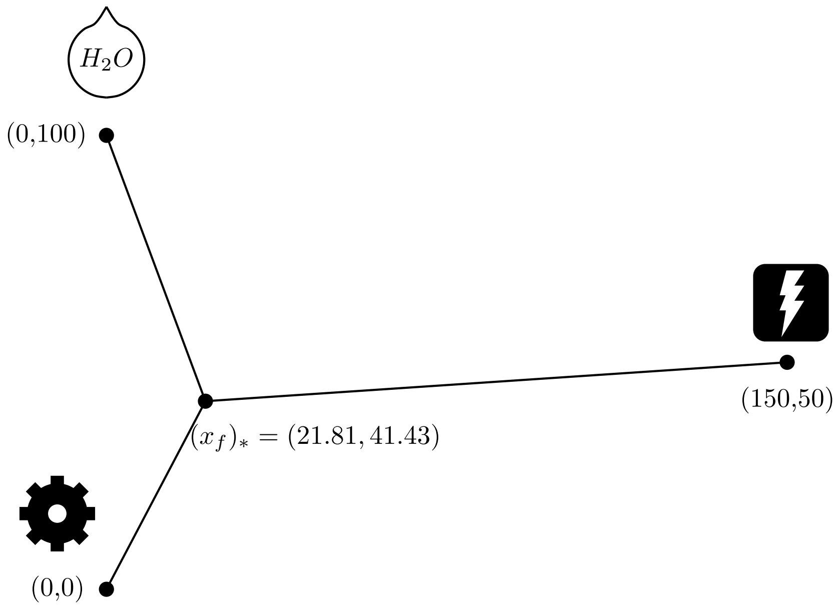

Figure 2.5: Positions of the processing plant, water reservoir and electricity plant and optimal placement of farm.

white nodes are discovered, then technically, and under reasonable assumptions on the objective function and the method, the sequence

$$\|x^l_{\text{unsuccessful}} - x_*\|$$

converges r-linearly to zero. (See e.g. [55], chapter 3.3 for more on convergence rates.)

Chapter 3

Using Curvature Information

3.1 Using Curvature Information in GSS methods

Compass search as presented in the previous chapter is in a sense memory-less, as it makes no use of previously sampled function values. As we will show, these values can be very useful. The search progress of compass search in $\mathbb{R}^2$ we have already illustrated to some extent in Figure 2.4, but in Figure 3.1 one can see one of its major drawbacks, what we will call *zig-zagging.* Since compass search (and all other methods we consider) require that the function value at a new point is lower than at the current point, it is often forced to take very small steps. The reason for this is that the angle between the negative of the gradient and the available search directions can be relatively large (about 45° in the figure), and the effect of this is that the method may have to perform a large number of function evaluations to cover the distance from the initial solution to the optimal solution. In the worst case, for instance a gradient of all ones, the smallest angle in $\mathbb{R}^2$ between any of the positive and negative unit vectors is 45°. This number depends on the dimension n. Zig-zagging can be illustrated with a simple example. Let

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \; \Lambda = \begin{bmatrix} 10^{-4} & \\ & 100 \end{bmatrix},$$

and

$$f(x) = x^T Q^T \Lambda Q x.$$

As one can see the optimal solution is $x_* = 0$, and $f(x_*) = 0$ as well. Let us apply compass search to the problem. If we take $x^0 = \begin{bmatrix} -2 & -2 \end{bmatrix}^T$ as our initial approximation, we have

$$\|x^0 - x_*\| = 2.8284, \text{ and } f(x^0) = 8 \cdot 10^{-4}.$$

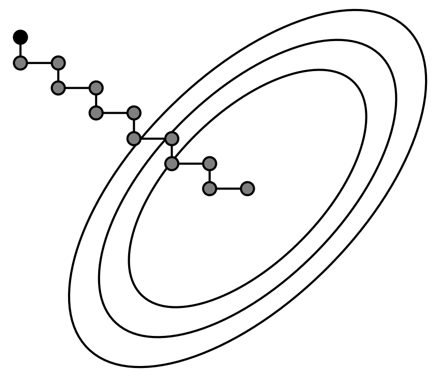

Figure 3.1: Zig-zagging while searching along coordinate directions. Only successful steps shown.

Let us take 0.1 to be the initial step lengths, and search along the positive and negative of the coordinate axes. Let us halve the step length δ_i if the search fails along both the direction d_i and $-d_i$. If the search succeeds along a direction, we immediately test if we can step again along the same direction. If this succeeds, we accept the second step and double the step length. This is how the subroutine *exploratory_moves* in paper I searches. Over the first 14 iterations, all that happens is that the method halves the step lengths, before starting to progress when the step lengths are of the order 10^{-6}. After 30 iterations, we have,

$$\|x^{30} - x_*\| = 2.8280, \text{ and } f(x^{30}) = 7.9976 \cdot 10^{-4},$$

which is very little progress, considering the number of iterations that have been performed. If we instead of searching along the coordinate directions had searched along the positive and negative of the column vectors of Q, then we would have obtained

$$\|x^{30} - x_*\| = 3.0510 \cdot 10^{-6}, \text{ and } f(x^{30}) = 9.3088 \cdot 10^{-16},$$

which is very close to the optimal solution.

As the example shows, the choice of search basis can to a large extent influence how much the method is able to progress per iteration. It would therefore be desirable to have a method which can automatically choose its search directions. This idea was behind two early direct search methods, the method of Hooke and Jeeves [40], which implicitly searches along the direction of average progress in addition to searching along the coordinate directions, and Rosenbrock's method [64], which is basically a variant of compass search, combined with rotation of all its search directions. Specifically, Rosenbrock's method aligns the principal search direction to the direction of average progress. Figure 3.2 illustrates what this means. In the left of the figure we see an example of search progress. One can see that the general direction of progress is down and

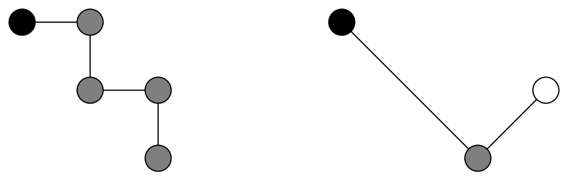

Figure 3.2: Illustration of the logic behind Rosenbrock's method. On the left, search before rotation of search directions, on the right, search after directions are rotated. The step lengths and time of rotation in this figure differ from Rosenbrock's implementation.

to the right. Therefore, a suitable choice of search directions, hopefully allowing for longer steps would be the directions in the right of the figure, one search direction being the direction of average progress, and the rest orthogonal to it. At suitable intervals Rosenbrock's method creates an $n \times n$ matrix where the first column contains the change in the variable since the last basis rotation. The second column contains the change in the variable along all the search directions except the first search direction, the third column the change along all the search directions except the first and the second search direction, an so on. The new n search directions are obtained by applying the Gram-Schmidt process to the columns of this matrix. This way the method adapts its search directions to the function, while maintaining the geometry of mutually orthogonal search directions.

From one to n Function-based Directions Out of the Rosenbrock method's n unique (apart from sign) orthogonal search directions, only one is based on information obtained from the function itself, namely the direction of average progress. The remaining $n-1$ directions are based on a decomposition of the information stored in the first direction. We would like a method where all its n unique directions are based on information from the function, while still preserving a healthy geometry, that is, we want the directions to be orthogonal.

Often, optimisation methods are designed to work well on quadratic functions. We will adopt this methodology, and implicitly use a quadratic model function. The question becomes what can be considered good orthogonal directions for a quadratic function.

We would like directions which allow us to take long steps along as many of the n unique directions as possible, which would then reduce zig-zagging, since zig-zagging is caused by the method only being able to take short steps along its search directions. We know from the theory behind the conjugate gradient method [39] for solving linear equations with symmetric positive definite coefficient matrices that conjugate directions with respect to the coefficient matrix are directions which do not lead to zig-zagging on quadratic functions, since if one minimises the corresponding quadratic function along such a direction, the direction need not be considered again. Recall that two directions p and q are by definition conjugate with respect to the matrix A if and only if

$$p^T A q = 0.$$

Since minimising a quadratic function, say,

$$f(x) = c - b^T x + \frac{1}{2} x^T A x,$$

with a symmetric positive definite Hessian A is the same as solving the linear equation system

$$Ax = b,$$

then conjugate directions should make good search directions, as studied in [10]. Indeed, derivative-free methods that search along conjugate directions have been proposed [20, 58]. However, conjugate directions are not necessarily orthogonal, so if the underlying implicit quadratic model is inaccurate, we could in theory end up with a worse zig-zagging problem than what we sought to prevent. Thus, ideally we would like conjugate, orthogonal search vectors based on curvature information from the function. Fortunately, such directions are easily identifiable, namely the eigenvectors of the Hessian of the underlying model function. We make use of these directions in the method of Paper I.

3.2 Paper I — A GSS Method Using Curvature Information

In paper I we present a moving grid GSS method which makes use of curvature information. This is done by observing that, as can be seen in Figure 2.4, when performing compass search along orthogonal directions, the points which the algorithm evaluates will lie in constellations which can be used by the formula

$$q_i^T \nabla^2 f(\widetilde{x}) q_j = \frac{f(x + hq_i + kq_j) - f(x + hq_i) - f(x + kq_j) + f(x)}{hk}, \qquad (3.1)$$

where the equation holds for two times continuously differentiable functions. To see this for $n > 2$, consider Figure 3.3. The situation depicted in the figure is that after two search directions have been considered. Independent of whether or not the search along either direction is successful, the constellation of points given by the three nodes as drawn in the figure, or rotated or mirrored, will always be found. If, for example both steps are successful, then the constellation occurs when stepping from node 1 to 2 to 3, which is what is shown in the figure. As before the search starts at the black point, and grey nodes signify points which are accepted. If only the first step is successful, then the constellation occurs if the method for instance again starts at point 1, steps to point 2, and

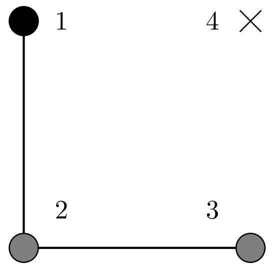

Figure 3.3: Example of constellation of points occurring in compass search.

considers point 3 but does not step. (The colours in the figure are from now on inconsistent with previous usage.) Similarly, with failure along first direction but not along the second direction (start at point 2, try point 1 but fail, try point 3 and step), as well as failure along both directions (start at point 2, try point 1, fail, try point 3, fail), the constellation occurs. If one, given such a constellation computes the function value at point 4 explicitly, one gets four points in a rectangle, which is exactly the set of points needed by formula (3.1). The algorithm shuffles the order of the search directions in order to obtain curvature information along all pairs of directions. An outline of the algorithm is given below:

- Perform compass search along the directions in the set $\mathcal{G}$, initially consisting of the columns of I and $-I$.
- Use formula (3.1) to compute a matrix C_Q, as the search progresses. If the function is two times continuously differentiable C_Q will have, as entry (i,j),
$$(C_Q)_{ij} = q_i^T \nabla^2 f(x^{ij}) q_j,$$
for various x^{ij}.
- Once C_Q is complete, calculate the Hessian approximation
$$\nabla^2 f(x) \approx C = QC_QQ^T.$$
- Compute the eigenvectors of C and replace $\mathcal{G}$ with the positive and negative of these eigenvectors.

A Small Example We illustrate a slightly simplified version of the algorithm on a small example. There are a few technical requirements on the algorithm imposed by the convergence theory which are addressed in the paper. The example is illustrated in figure 3.4. Let:

$$f(x) = f(x_1, x_2) = 5x_1^2 + 2x_1x_2 + 10x_2^2.$$

Let the initial variable be

$$x^0 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

and let the search basis be

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \end{bmatrix}.$$

Let the initial step lengths be

$$\delta = \begin{bmatrix} 0.1 \\ 3 \end{bmatrix}.$$

We order the directions

$$\begin{bmatrix} q_1 & q_2 & -q_1 & -q_2 \end{bmatrix}.$$

Now, the fact that q_1 and q_2 immediately follow each other as the first two search directions enables us to compute $(C_Q)_{12}$. (The algorithm of paper I actually makes use of search directions in groups of three, but we do not touch upon this behaviour in this example.) At our initial point $x = x^0$ we have

$$f(x) = 49.$$

We search along q_1 and find

$$f(x + \delta_1 q_1) = 47.0851,$$

a function value reduction. We then immediately test if we can double the step length along this direction and test

$$f(x + 2\delta_1 q_1) = 45.3002,$$

an additional reduction. So now we set

$$x \leftarrow x + 2\delta_1 q_1,$$

and compute, by (3.1) (with $q_i = q_j$ and $\delta_i = \delta_j$)

$$(C_Q)_{11} \leftarrow \frac{45.3002 - 2 \cdot 47.0851 + 49}{0.1^2} = 13.$$

Finally we set

$$\delta_1 \leftarrow 2\delta_1 = 0.2.$$

Now we are ready to search along q_2. We test

$$f(x + \delta_2 q_2) = 237.5941.$$

The search along q_2 does not provide us with a lower function value, but we leave δ_2 unchanged for the time being. Since we were able to step along q_1 but not along q_2, we are in the situation depicted in Figure 3.3, with the black point in Figure 3.4 corresponding to point 1 in Figure 3.3, and points 2, 3 and 4 being the same. We need to compute the function value at the point marked by a cross, namely

$$f(x - \delta_1 q_1 + \delta_2 q_2) = 244.2939.$$

Now we can compute, by (3.1)

$$(C_Q)_{12} \leftarrow \frac{237.5941 - 244.2939 - 45.3002 + 49}{0.2 \cdot 3} = -5.$$

Note that we have used the updated δ_1 in the denominator. The next direction scheduled for search is $-q_1$. Now, since

$$x - \delta_1 q_1 = x^0,$$

the search direction $-q_1$ is skipped. The next direction for search is $-q_2$. We get

$$f(x - \delta_2 q_2) = 6.0063.$$

Immediately, again, we test

$$f(x - 2\delta_2 q_2) = 119.7123.$$

Stepping twice was not successful, so we set

$$x \leftarrow x - \delta_2 q_2.$$

Now we can compute

$$(C_Q)_{22} = \frac{119.7123 - 2 \cdot 6.0063 + 45.3002}{3^2} = 17.$$

All elements in C_Q are now determined and we can solve for our new search basis. We compute

$$C \leftarrow Q \begin{bmatrix} 13 & -5 \\ -5 & 17 \end{bmatrix} Q^T = \begin{bmatrix} 10 & 2 \\ 2 & 20 \end{bmatrix},$$

the exact Hessian, since f is quadratic. All that remains is to eigenvalue-factorise C and use its eigenvectors as the new search basis.

Numerical Results Our numerical experiments show that this new method performs much better than compass search in terms of the number of function evaluations required to reach the optimal solution, except when the coordinate directions themselves are near-conjugate directions. This is the case for both smooth and noisy problems.

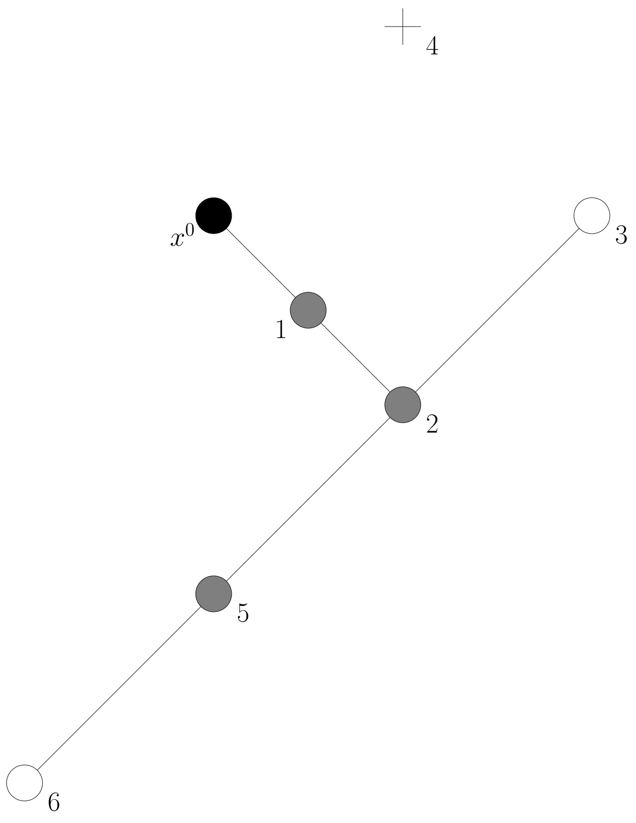

Figure 3.4: Visualisation of the algorithm's progress in the example. (The axes are unequally scaled.) The black point is the starting point. Grey points are taken, the white points are not. The cross marks the extra point evaluated for the computation of $(C_Q)_{12}$. The numbers indicate the order in which the points are evaluated.

Chapter 4

Using Separability Information

4.1 Separability and Sparsity

First derivative and curvature information are good examples of information one can gather about the function as the search progresses. It is also sometimes possible to make statements about the function before it is evaluated, by looking at its representation. The property of partial separability falls into this category. Given the sets

$$\chi_1, \chi_2, \ldots, \chi_\nu, \quad \chi_i \subseteq \{1, 2, \ldots, n\}, \text{ for all } i.$$

Consider the function

$$f : \mathbb{R}^n \mapsto \mathbb{R}, \quad f = \sum_{i=1}^{\nu} f_i, \quad f_i : \mathbb{R}^{|\chi_i|} \mapsto \mathbb{R}, \tag{4.1}$$

where each function depends only on a few of the components of x, specified by the χ-sets. That is, f_i depends on the components of x corresponding to the indices in the set χ_i. An example in $\mathbb{R}^3$ is

$$f(x) = \sin(x_1) + \cos(x_2 x_3), \tag{4.2}$$

which can be written

$$f = f_1 + f_2,$$

with

$$\chi_1 = \{1\}, \quad \chi_2 = \{2, 3\}.$$

Thus, f is the sum of ν *element functions*, each of which ideally depends on only a few variables, and thus has an invariant subspace. Such a function is called a partially separable function. Partial separability was introduced by Griewank and Toint, see e.g. [37], and [38] for the incorporation of partial separability into quasi-Newton methods.

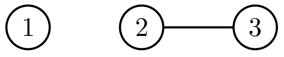

Figure 4.1: Covariation graph corresponding to function (4.2).

The Covariation Graph Separability is usually associated with differentiability. We present a new definition which is applicable to non-differentiable functions as well. Define, for any function $f : \mathbb{R}^n \mapsto \mathbb{R}$ the *covariation graph*,

$$G(V, E), \ |V| = n, \tag{4.3}$$

that is, a graph with n nodes. Let e_i be the ith unit coordinate vector. If, for some x, we have

$$f(x + e_i + e_j) - f(x + e_i) - f(x + e_j) + f(x) \neq 0,$$

then let there be an edge between from node i to node j in the graph. If there is an edge from node i to j there must also be an edge from node j to node i. In other words, the graph G can be viewed as undirected. In the case of the function (4.2), we get the graph in Figure 4.1.

Theorem 1 *Let a continuous function $f : \mathbb{R}^n \mapsto \mathbb{R}$, and its covariation graph be given. If the graph is not complete then the function is separable.*

Proof. If the graph is not complete then there exist i and j such that

$$f(x + e_i + e_j) - f(x + e_i) - f(x + e_j) + f(x) = 0,$$

for all x. Assume without loss of generality that $n = 2$, and that $i = 1$ and $j = 2$. Then we have, for all h and k,

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

Now, let x_1 and x_2 be identical to zero, and let h and k be the independent variables. Then we get

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

which can be written

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

Now we can define, for instance

$$f_1(h) = f(h, 0) + f(0, 0),$$

and

$$f_2(k) = f(0, k),$$

and we have that f can be written on the form (4.1), and thus is separable. □
From the covariation graph we can make a graph adjacency matrix, which is symmetric since the graph G is undirected. The matrix for (4.2) becomes

$$\begin{bmatrix} \times & & \\ & \times & \times \\ & \times & \times \end{bmatrix}. \tag{4.4}$$

If the function is two times differentiable, then the structure of the adjacency matrix arising from the covariation graph is the structure of the Hessian. In addition, the result of Theorem 1 can then be stated in terms of partial derivatives, that is, that if any of the cross-derivatives $\frac{\partial f}{\partial x_i \partial x_j}$ is identical to zero, then the function is partially separable [37].

In paper IV we extend our definition of separability so that it is not dependent on the vectors $e_1, \dots, e_n$, instead depending on n orthogonal or n general linearly independent vectors.

4.2 Exploiting Separability in Direct Search Methods

It is possible to exploit information about separability in direct search methods. A method which does this is the method of Price and Toint, which assumes that each individual element function is available, and takes advantage of this.

The method of Price and Toint In [63], a direct search method which makes use of partial separability of the objective function is presented. Specifically, it does this by noting that for separable functions where the individual element functions are known, evaluating the function at points of the form

$$f \pm he_i, \ i = 1, \dots, n,$$

is relatively inexpensive, since one may not need to evaluate the entire function, only the element functions which are affected by the change in the variable x. In addition, one sometimes obtains function values at specific points at no extra cost. For instance, if the function is totally separable, that is, for instance

$$f(x) = \sum_{i=1}^{n} f_i(x_i),$$

then evaluating the entire function at two points x and y gives the values at the points

$$z, \text{ such that } z_i = x_i \text{ or } z_i = y_i, \ i = 1, \dots, n,$$

provided that the element functions are known explicitly. This approach enables the algorithm to solve problems with a large number of variables (as many as 5000 in the paper) at little cost.

In papers II and IV, we employ separability without assuming that the individual element functions are available. We extend the algorithm of paper I to take advantage of knowledge about the structure of the adjacency matrix of the covariation graph, so that curvature information can be gathered more efficiently.

4.3 Paper II & IV — GSS Methods Exploiting Curvature and Separability

Computing Curvature Information Element by Element In paper II we present an extension to the moving grid method of paper I, by imposing a structure on the matrix C used in the first paper.

In paper IV we extend the algorithm further and answer theoretical questions which were left open in paper II. We impose a sufficient decrease condition on the exploratory moves making the method of paper IV a (provably convergent) sufficient decrease method.

Recall that the algorithm of paper I computed curvature in a rotated coordinate system and assembled it in a matrix C_Q, such that

$$\nabla^2 f \approx C = QC_QQ^T. \tag{4.5}$$

Equation (4.5) can be written in a different form, using Kronecker products. Kronecker products can be defined by

$$AXB = C \Leftrightarrow (B^T \otimes A)\mathbf{vec}(X) = \mathbf{vec}(C), \tag{4.6}$$

Where $\mathbf{vec} : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n^2}$ stacks the entries of the matrix X, column by column in a vector. The matrix $(B^T \otimes A)$ is $n^2 \times n^2$, with entries

$$(B^T \otimes A) = \begin{bmatrix} B_{11}A & \cdots & B_{n1}A \\ \vdots & & \vdots \\ B_{1n}A & \cdots & B_{nn}A \end{bmatrix}. \tag{4.7}$$

If we apply this to the last part of the equation (4.5) we can write

$$C = QC_QQ^T \Longleftrightarrow (Q^T \otimes Q^T)\mathbf{vec}(C) = \mathbf{vec}(C_Q). \tag{4.8}$$

We impose the sparsity structure of the adjacency matrix of the covariation graph of f on C. Now if $Q = I$ the $n^2 \times n^2$ equation system

$$(Q^T \otimes Q^T)\mathbf{vec}(C) = \mathbf{vec}(C_Q) \tag{4.9}$$

can be reduced in size based on knowledge of which elements of C are zero, as well as the fact that C is symmetric, so that we only need to compute the upper or lower triangular elements of C_Q.

We can apply the same idea even if $Q \neq I$, as long as we are a little careful. It turns out not to be obvious which elements of C_Q are to be computed in this case. The resulting coefficient matrix might be singular if elements are not properly chosen. In paper II we employ a simple heuristic for choosing these elements, in paper IV we extend this heuristic to a method which we prove always returns a nonsingular coefficient matrix. In addition, the solution itself will depend on which elements are chosen if the right-hand side $\mathbf{vec}(C_Q)$ contains truncation error. To clarify, consider the following two examples. Let the Hessian be diagonal and constant, say,

$$\nabla^2 f = \begin{bmatrix} 1 & & \\ & 2 & \\ & & 3 \end{bmatrix}.$$

Let

$$Q = \begin{bmatrix} \frac{\sqrt{3}}{3} & \frac{\sqrt{2}}{2\sqrt{3}} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{3}}{3} & -\frac{\sqrt{2}}{\sqrt{3}} & 0 \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{2}}{2\sqrt{3}} & \frac{\sqrt{2}}{2} \end{bmatrix}.$$

$(Q^T \otimes Q^T)$ is a 9×9 matrix, but since we require all off-diagonal elements of C to be zero, we can cut all but three of its columns, giving us the an overdetermined system which can be written

$$(Q^T \otimes Q^T) P_c \begin{bmatrix} C_{11} \\ C_{22} \\ C_{33} \end{bmatrix} = \mathbf{vec}(C_Q). \tag{4.10}$$

Define

$$\overline{\mathbf{vec}}(C) = \begin{bmatrix} C_{11} \\ C_{22} \\ C_{33} \end{bmatrix},$$

Then P_c is the (9×3) matrix such that

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

To turn (4.10) into an equation system with a square coefficient matrix we can either solve for the least squares solution, or we can cut rows from the coefficient matrix, which corresponds to computing only selected elements in C_Q. We would like to compute no more elements of C_Q than strictly needed when the adjacency matrix of the covariation graph is sparse, so we choose the latter option. Now, if we choose to evaluate the three off-diagonal unique elements in C_Q, namely

$$C_Q : \begin{bmatrix} & & \\ \times & & \\ \times & \times & \end{bmatrix},$$

then the resulting coefficient matrix A, which can be written

$$A = P_r (Q^T \otimes Q^T) P_c,$$

will be singular. This can be seen if one tries to compute

$$A \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = P_r \mathbf{vec}((Q^T I Q)) = 0,$$

since P_r cuts all the elements corresponding to the diagonal of C_Q. If one instead tries to compute for instance the elements

$$C_Q : \begin{bmatrix} \times & & \\ & & \times \\ & & \times \end{bmatrix},$$

then A will be nonsingular. Furthermore if the function is two times continuously differentiable, and the elements of C_Q contain truncation errors or equivalently are correspond to derivatives computed at different x, that is

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

$$(C_Q)_{rs} = q_r^T \nabla^2 f(x^{rs}) q_s,$$

where

$$x^{ij} \neq x^{rs},$$

then the solution C will also depend on which elements of C_Q we compute. This is also true if the function is not differentiable. To see this, assume that $\nabla^2 f$ exists and is diagonal but not constant, and that

$$Q = \begin{bmatrix} q_1 & q_2 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Let e be a constant, and let

$$(C_Q)_{11} = q_1^T \begin{bmatrix} 1-e & 0 \\ 0 & 1-e \end{bmatrix} q_1, \quad (C_Q)_{21} = q_2^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} q_1,$$

$$(C_Q)_{12} = (C_Q)_{21}, \quad (C_Q)_{22} = q_2^T \begin{bmatrix} 1+e & 0 \\ 0 & 1+e \end{bmatrix} q_2.$$

Then, if we compute C from $(C_Q)_{11}$ and $(C_Q)_{12}$ we get,

$$C = \begin{bmatrix} 1-e & 0 \\ 0 & 1-e \end{bmatrix},$$

whereas if we compute C from $(C_Q)_{12}$ and $(C_Q)_{22}$ we get

$$C = \begin{bmatrix} 1+e & 0 \\ 0 & 1+e \end{bmatrix},$$

and least squares solution from all elements of C_Q yields $C = I$. Either way, the difference between the actual Hessian and C, roughly speaking, grows with the size of h and k used to compute C_Q elements in formula (3.1), and depends on the condition number of the matrix A. This is elaborated upon in the two papers.

Numerical results show that the algorithms of the two papers both outperform the method of paper I, especially as n grows, and the adjacency matrix of the covariation graph has many identical zeros.

Chapter 5

Limited Memory Methods

5.1 Quasi-Newton Methods

Quasi-Newton methods are powerful gradient-based methods which use gradient information from past iterations to approximate the Hessian. For a comprehensive treatment of the subject, see e.g. [50] and the references therein. The original idea of using acquired gradient information to approximate the Hessian was introduced in 1959 by Davidon [24]. (Available as [25], since the original paper was never published until the first issue of SIAM Journal on Optimization in 1991.) The work of Davidon was continued by Fletcher and Powell [31], resulting in what is called the DFP method. The DFP method maintains a positive definite approximation to either the Hessian, its inverse, or a factorisation of either of the two. At iteration k, let us call the approximation to the Hessian B^k, and the method sets the point x^{k+1} by the formula

$$x^{k+1} = x^k - \alpha^k (B^k)^{-1} \nabla f(x^k),$$

where α^k is usually taken to satisfy e.g. the strong Wolfe conditions, which are

$$\begin{aligned} f(x^k + \alpha^k p^k) &\leq f(x^k) + c_1 \alpha^k \nabla f(x^k)^T p^k & (5.1) \\ |\nabla (f + \alpha^k p^k)^T p^k| &\leq c_2 |\nabla f(x^k) p^k|, & (5.2) \end{aligned}$$

where

$$0 < c_1 < c_2 < 1,$$

and in practice typically $c_1 = 10^{-4}$ and $c_2 = 0.9$ for the methods we will be concerned with (see e.g. [55], section 3.1). The matrix B^k is then updated by the formula

$$B^{k+1} = (I - \gamma^k y^k (s^k)^T) B^k (I - \gamma^k s^k (y^k)^T) + \gamma^k y^k (y^k)^T, \qquad (5.3)$$

where

$$s^k = x^{k+1} - x^k, \ y^k = \nabla f(x^{k+1}) - \nabla f(x^k), \ \gamma^k = \frac{1}{(y^k)^T s^k}.$$

One arrives at the formula (5.3) by solving the problem

$$B^{k+1} = \arg\min_{B} \|B - B^k\|, \tag{5.4}$$

where B is required to be symmetric, and in addition satisfy the secant equation

$$B^{k+1}s^k = y^k. \tag{5.5}$$

The choice of norm in (5.4) plays a role in the sense that different norms result in different update formulas, but this is beyond the scope of this discussion. The most important properties as far as we are concerned is that the method generates descent directions. For this to be the case B^k needs to be positive definite. The reason for this is that only if B^k is positive definite, the angle between the negative of the gradient and the search direction $-B^k\nabla f(x^k)$ is provably in the range $(-90°, 90°)$. The cosine of this angle, say, η is given as

$$\cos(\eta) = \frac{\nabla f(x^k)^T B^k \nabla f(x^k)}{\|\nabla f(x^k)\| \; \|B^k \nabla f(x^k)\|} \tag{5.6}$$

Since the matrix B^k is positive definite, and hence that

$$x^T B^k x > 0, \; \forall x \in \mathbb{R}^n,$$

the numerator of the fraction in (5.6) is always strictly positive, and hence the direction the angle between $-\nabla f(x^k)$ and $-B^k\nabla f(x^k)$ is in the range $(-90°, 90°)$ as required. Furthermore, as long as B^k is positive definite and $\gamma^k > 0$, B^{k+1} is also positive definite.

The most popular and efficient quasi-Newton method in use today is the BFGS method [5]. One can derive this method by instead of solving (5.4), minimising with respect to the inverse of B, which we call H, and solve

$$H^{k+1} = \arg\min_{H} \|H - H^k\|, \tag{5.7}$$

where H is required to be symmetric, as well as to satisfy the inverse of the secant equation (5.5), namely

$$H^{k+1}y^k = s^k. \tag{5.8}$$

The formulas for x^{k+1} and H^{k+1} in the BFGS method are

$$x^{k+1} = x^k - \alpha^k H^k \nabla f(x^k),$$

and

$$H^{k+1} = (I - \rho^k s^k (y^k)^T) H^k (I - \rho^k y^k (s^k)^T) + \rho^k s^k (s^k)^T, \tag{5.9}$$

where as before

$$s^k = x^{k+1} - x^k,$$
$$y^k = \nabla f(x^{k+1}) - \nabla f(x^k),$$

```
Given $f$, $k = 0$, $x^0$, $H^0$, $\nabla f(x^0)$,
While $\|\nabla f(x^k)\| >$ tolerance,
    Set $x^{k+1} \leftarrow x^k - \alpha^k H^k \nabla f(x^k)$, for some $\alpha^k$ satisfying the (strong)
    Wolfe conditions.
    Compute $\nabla f(x^{k+1})$.
    Compute $H^{k+1}$ by (5.9).
    Set $k \leftarrow k + 1$.
end.
```

Figure 5.1: Pseudocode for the BFGS algorithm.

and in addition

$$\rho^k = \frac{1}{(y^k)^T s^k}.$$

H^{k+1} is positive definite as long as H^k is positive definite, and $\rho^k > 0$. It can be shown that if α^k is required to conform to the Wolfe conditions, then ρ^k is positive. The Wolfe conditions are a slightly less restrictive version of the strong Wolfe conditions, or vice versa. Keeping an approximation to the inverse of the Hessian rather than the Hessian itself gives a cheaper algorithm in terms of operation count, since one does not have to solve an equation system at every iteration. As mentioned, it is also possible to store either B or H in factorised form, for instance the Cholesky factorisation of B, or a conjugate factorisation of H, as is done in [19]. As is tested and discussed in [11] among others, there is little difference whether one chooses a factorised or non-factorised implementation as long as gradients are available to machine precision. The authors claim in [11] that when gradients are only available to limited precision (e.g. are approximated with finite difference formulas), then factorised implementations perform better, in the sense that they converge more often.

Pseudo code for the BFGS method is given in Figure 5.1. We now turn our attention to how the initial approximation to the Hessian B^0 or its inverse H^0 should be chosen. The most obvious choice, if one knows little about the function is $H^0 = I$. It has been suggested to let H^0 be a diagonal matrix whose entries are the inverses of the diagonal entries of the Hessian. These are, as noted in [65] expensive to compute, and if the resulting matrix is not positive definite, it cannot be used unaltered, since it might not generate descent directions. It is suggested in [65] to compute x^1 using $H^0 = I$, and scale H^0 before H^1 is computed, by the formula

$$\hat{H}^0 = \frac{(y^1)^T s^1}{(y^1)^T y^1}, \tag{5.10}$$

and then forming H^1 from $\hat{H}^0$ instead of from H^0. The numerical results of [65] show that except for small n, computing H^1 from $\hat{H}^0$ gives a much more effective algorithm in terms of iterations than computing H^1 from $H^0 = I$. If we apply BFGS (updating H^1 from $\hat{H}^0$) to the farm problem (2.2), we get the results in Table 5.1. BFGS does not converge as rapidly as Newton's method, but is much

Iteration	f	$\|\nabla f(x_f)\|$	$(x_f)_1$	$(x_f)_2$
0	1.90208e+06	5.07036e+03	5.00000e+01	5.00000e+01
1	1.83968e+06	3.40752e+03	1.07749e+01	4.44757e+01
2	1.82294e+06	9.94100e+02	2.60807e+01	4.35143e+01
3	1.82083e+06	1.77030e+02	2.25124e+01	4.27033e+01
4	1.82072e+06	6.05756e+01	2.18219e+01	4.19556e+01
5	1.82071e+06	2.05783e+01	2.17742e+01	4.15489e+01
6	1.82071e+06	1.28433e+00	2.18062e+01	4.14305e+01
7	1.82071e+06	8.37229e-02	2.18109e+01	4.14313e+01
8	1.82071e+06	5.27479e-04	2.18112e+01	4.14316e+01
9	1.82071e+06	7.85562e-06	2.18112e+01	4.14316e+01
10	1.82071e+06	6.87501e-06	2.18112e+01	4.14316e+01
11	1.82071e+06	2.73346e-11	2.18112e+01	4.14316e+01
12	1.82071e+06	1.25056e-12	2.18112e+01	4.14316e+01
13	1.82071e+06	1.50071e-12	2.18112e+01	4.14316e+01
14	1.82071e+06	9.11269e-13	2.18112e+01	4.14316e+01

Table 5.1: 14 BFGS iterations on problem (2.2).

faster than GSS, or e.g. steepest descent. BFGS converges superlinearly, that is,

$$\lim_{k\to\infty} \frac{\|x^{k+1} - x_*\|}{\|x^k - x_*\|} = 0.$$

Since quasi-Newton algorithms perform only one gradient evaluation per iteration disregarding gradient computations in the line search, the same as steepest descent, they are a good example of how one can use previously obtained information effectively.

5.2 Derivative-Based Limited Memory Methods

When n is large, the memory required to store a full Hessian or approximations to it, namely $O(n^2)$ may be more than is practical on a given computer system. The aim of limited-memory methods is to be able to solve the step equation using only $O(n)$ memory. Two such methods are limited memory BFGS [48, 54, 12] and discrete Newton [56, 27].

Limited memory BFGS is a variant of BFGS which instead of maintaining an approximation H^k to the inverse of the Hessian based on the k vector pairs

$$y^0, \dots, y^k \text{ and } s^0, \dots, s^k,$$

keeps only the m most recent pairs, where m can be chosen by the user. This requires little storage if m is not too large (usually less than about 30), since the product of H^k with an arbitrary vector v can be computed without constructing H^k itself, using the procedure in Figure 5.2. Between the two for-loops we have

```
Given $p$, $H_0^k$, $s^{k-m}, \dots, s^{k-1}$, $y^{k-m}, \dots, y^{k-1}$,
$q \leftarrow v$,
For $i = k-1$ to $k-m$,
    $\alpha_i \leftarrow \rho^i (s^i)^T q$,
    $q \leftarrow q - \alpha^i y^i$,
end.
$r = H_0^k q$,
For $i = k-m$ to $k-1$,
    $\beta \leftarrow \rho_i (y^i)^T r$,
    $r \leftarrow r + s^i(\alpha_i - \beta)$,
end.
```

Figure 5.2: Two-loop procedure to compute $r = H^k v$. α is a vector of length m used for storage.

the assignment

$$r = H_0^k q.$$

The matrix H_0^k plays the same role as the matrix H^0 in the first line of the regular BFGS method in Figure 5.1, in the sense that is serves as the initial approximation to the inverse of the Hessian, which the gradient-variable difference pairs (s^i, y^i) subsequently modify. In the regular BFGS method H^0 is only used as the approximation of the inverse of the Hessian at the starting point x^0. In L-BFGS, since we only keep the m most recent difference pairs, H_0^k at iteration k serves as an approximation to the inverse of the Hessian at x^j, $j = \max\{1, k-m\}$, that is, at iteration k, $k > m$ the method assumes that

$$H_0^k \approx \left[\nabla^2 f(x^{k-m})\right]^{-1}.$$

This leads to the question of how to choose H_0^k. Liu and Nocedal [48] discuss several choices for H_0^k, and find that the dynamic scaling of the identity matrix

$$H_0^k = \frac{(y^{k-1})^T s^{k-1}}{(y^{k-1})^T y^{k-1}} I, \tag{5.11}$$

leads to a more effective algorithm than for instance $H_0^k = I,\ k = 1, \ldots$ or

$$H_0^k = \frac{(y^1)^T s^1}{(y^1)^T y^1} I,\ k = 1, \ldots,$$

as is done in the regular BFGS method. Different rules for H_0^k are also discussed in e.g. [34], where the best choice tested for H_0^k is a diagonal matrix based on a full Hessian approximation B.

Discrete Newton methods are variants of Newton's method that use an iterative method for solving the step equation, that is

$$\nabla^2 f(x^k) p^k = -\nabla f(x^k). \tag{5.12}$$

Since iterative solvers such as the conjugate gradient method need only the product of the coefficient matrix with an arbitrary vector, not the matrix itself, the approximation

$$\nabla^2 f(x) v \approx \frac{\nabla f(x + \epsilon v) - \nabla f(x)}{\epsilon}. \tag{5.13}$$

can be used to this end. In addition, although not related to memory, one does not have to solve the step equation to full accuracy at each iteration. A useful rule is to solve (5.12), at iteration k, to accuracy

$$\frac{\|\nabla^2 f(x^k)\widetilde{p}^k + \nabla f(x^k)\|}{\|\nabla f(x^k)\|} \leq \min\left\{\frac{1}{k}, \|\nabla f(x^k)\|\right\}, \tag{5.14}$$

where $\widetilde{p}^k$ is an approximate solution to (5.12) [27]. This is called a discrete truncated Newton method, or a truncated Newton method if the product $\nabla^2 f v$ is computed from Hessian information rather than from (5.13).

5.3 Hessian Sparsity in Derivative-Based Methods

Sparsity can be used effectively to evaluate Hessian matrices at little cost (see e. g. [22, 13, 53]). For instance, assume that the Hessian of the objective function is tridiagonal, and that it is known only as the product of the Hessian with an arbitrary vector, $\nabla^2 f \cdot v$, either by a finite difference formula or through automatic differentiation. Then, an $n \times n$ Hessian can be directly determined using only three evaluations. To see how this can be the case, consider the tridiagonal matrix structure

$$\nabla^2 f = \begin{bmatrix} h_{11} & h_{12} & & & \\ h_{21} & h_{22} & h_{23} & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-1,n-2} & h_{n-1,n-1} & h_{n-1,n} \\ & & & h_{n,n-1} & h_{nn} \end{bmatrix}. \tag{5.15}$$

Assume without loss of generality that n is divisible by 3. If we now choose to compute the three products

$$\nabla^2 f \cdot s_1,\ \nabla^2 f \cdot s_2,\ \nabla^2 f \cdot s_3,$$

where

$$s_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad s_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad s_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{5.16}$$

then the resulting vectors will be

$$\nabla^2 f \cdot s_1 = \begin{bmatrix} h_{11} \\ h_{21} \\ h_{34} \\ h_{44} \\ h_{54} \\ h_{67} \\ \vdots \end{bmatrix}, \quad \nabla^2 f \cdot s_2 = \begin{bmatrix} h_{12} \\ h_{22} \\ h_{32} \\ h_{45} \\ h_{55} \\ h_{65} \\ \vdots \end{bmatrix}, \quad \nabla^2 f \cdot s_3 = \begin{bmatrix} h_{23} \\ h_{33} \\ h_{43} \\ h_{56} \\ h_{66} \\ h_{76} \\ \vdots \end{bmatrix}, \tag{5.17}$$

and we can read the values of the sought Hessian elements directly. The vectors s_1, s_2, s_3 are often called *seed* vectors. Since the Hessian is symmetric and only has $2n-1$ unknowns in this case and each Hessian vector product produces n equations we can determine a tridiagonal Hessian from only two such products, but we then have to solve a system of equations to find the explicit values of the elements. Assuming, again without loss of generality, that n is even, then a pair of seed vectors which accomplishes this is

$$s_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \quad s_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{5.18}$$

Note that many choices of seed vectors would help us determine the Hessian, as long as the resulting equation system

$$\nabla^2 f \cdot \begin{bmatrix} s_1 & s_2 \end{bmatrix} = W, \tag{5.19}$$

has a unique solution. Thus, if one knows in advance that the Hessian of the objective function is sparse, one can compute the entire Hessian cheaply, and hence optimise the function with a Newton-type method with less operations than if one does not make use of information about sparsity.

5.4 Paper V — A Class of Methods Combining TN and L-BFGS

In this paper we present a new class of hybrid limited memory methods. The class can be described based on the L-BFGS method. Instead of performing the assignment

$$r = H_0^k q, \tag{5.20}$$

between the two for-loops in the procedure to compute $H^k v$, we solve the equation system

$$\nabla^2 f(x^k) r = q, \tag{5.21}$$

inexactly, using an iterative equation solver, and subject to some forcing sequence. Apart from this assignment and that we will allow m to vary, the methods in the class are equal to L-BFGS. This way, if $m = 0$ throughout, the method reduces to a truncated Newton algorithm, and if m is constant and the iterative equation solver applied to (5.21) returns (5.20), then the method becomes L-BFGS.

We test a member of the class with $m = 3$ and a simple forcing sequence for (5.21). We simulate the cost for different situations, whether or not derivatives are available through AD, and whether or not we are able to exploit the techniques of section 5.3. We find our preliminary numerical results very promising, and for the concrete method tested we find that it preforms well compared with L-BFGS with $m = 3$ and truncated Newton with forcing sequence (5.14), and that it generally performs well if one of its parent methods does. A nice example is given in Figure 5.3, where the method starts as effectively as L-BFGS, and when L-BFGS stagnates it continues progressing towards the optimal solution at about the rate of truncated Newton. We conclude that the class of methods is a promising one, and that more research should be conducted to identify forcing sequences and values of m which can give rapid convergence with little memory.

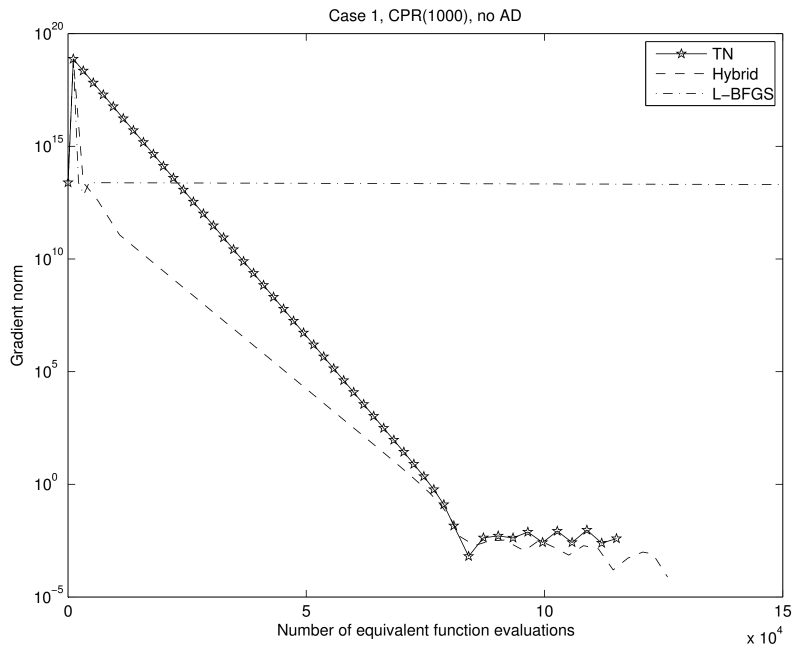

Figure 5.3: Plot of a hybrid method, TN and L-BFGS on the Penalty I function from the CUTEr collection [35]. See paper V for a detailed description of the plot.

Chapter 6

Application of GSS to Maximum Likelihood Estimation

6.1 Maximum Likelihood Estimation

Consider the situation where we want to determine some quantity found in nature. This may be, for instance the height of men aged 25. We assume, correctly or incorrectly, that this quantity has a normal distribution, with mean μ and variance σ^2. The probability density of the normal distribution is given as

$$\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right). \tag{6.1}$$

We conduct multiple observations with different results. Let us say we observe ten men, with observations as in Table 6.1. The table consists of values drawn from a pseudo-random normal distribution with $\mu_{\text{real}} = 180$ and $\sigma_{\text{real}} = 10$. What are the most likely values of the quantities μ and σ, given the observations? This is what maximum likelihood estimation addresses. Associated with the distribution function (6.1) and the recorded observations is a log-likelihood function, in this case given as

$$l(\mu,\sigma) = -10\log(\sqrt{2\pi}\sigma) - \frac{-\sum_{i=1}^{10}(x_i-\mu)^2}{2\sigma^2}, \tag{6.2}$$

We can now use optimisation methods to maximise this function, to obtain the estimates

$$\mu = 182.31 \text{ and } \sigma = 8.38.$$

It should be noted that the log-likelihood function (6.2) can be maximised analytically, but for the more complicated model we shall discuss below analytical results are not available. Even though the data set we generated comes from

Observed heights
178.13
187.25
174.11
201.83
178.63
181.13
190.66
180.59
179.04
171.67

Table 6.1: Observed heights in centimeters of ten men.

a normal distribution (or a distribution very close to one) we do not recover the true values for μ and σ. Maximum likelihood estimates are unbiased *in the limit*, which means that if we had an infinite number of samples from the true distribution, we would recover the values μ_{real} and σ_{real}. Since we have a quite small data set, we experience bias. See for instance [57] for more on likelihood estimation.

6.2 Paper III — Estimating Mortality of Norwegian Spring Spawning Herring

In this paper we try to determine the amount of data required to make statements about variations in the mortality of Norwegian spring spawning herring. By data sets we mean acoustical observations of the herring stock like those presented in [41]. This we do via a model which can be applied to other species as well. A *cohort* is the group of herring born a particular year. Let N_{jt} be the size of cohort j the year t, with $t = 0$ being the year it is born in our model. Let C_{jt} be the number of herring from cohort j caught in year t. (The age of herring can be determined by looking at their shells, which is similar to determining the age of a tree by looking at its growth rings.) In addition, a portion of each cohort dies of causes not related to catch, so that we can express the size of a cohort recursively by the expression

$$N_{j,t} = (N_{j,t-1} - C_{j,t-1})e^{-(M+\epsilon_{t-1})}. \tag{6.3}$$

The portion of a cohort which does not survive to the next year due to other causes than catch is controlled by $M + \epsilon_{t-1}$, which we call the mortality parameter for year $t-1$. We assume that the individual $\epsilon_t,\ t = 0, \ldots, n-1$, are normally distributed, with mean 0 and variance τ^2. Our goal is to estimate τ.

To obtain estimates of τ we construct artificial data sets, the same way we did for the example of heights, and test for which sizes of the data set the maximum likelihood estimate exists, and its level of accuracy. The likelihood function we are interested in can be written as $l(\theta, \epsilon)$, where θ and ϵ both are vectors of variables. ϵ is the vector containing $\epsilon_t,\ t = 0, \ldots, n-1$ from (6.3), and θ is a vector containing τ as well as some additional variables not of primary interest to us, such as M and $N_{j,0},\ j = 1, .., A$, from (6.3), A being the number of cohorts under consideration. We are interested in the marginal likelihood function with respect to the components of θ, so we are interested in maximising the function

$$l(\theta) = \log \left[\int \exp\Big(l(\theta, \epsilon) \Big)\, d\epsilon \right]. \tag{6.4}$$

Since ϵ in our experiments consists of 20 variables, solving the integral (6.4) is very expensive, so instead we maximise an approximation. The approximation to the integral (6.4) we use is called a Laplace approximation [66], and has the form

$$l^*(\theta) = -\frac{1}{2} \log \det \big(-\nabla^2 l(\theta, \bar{\epsilon}) \big) + l(\theta, \bar{\epsilon}), \tag{6.5}$$

where we by

$$\nabla^2 l(\theta, \bar{\epsilon}),$$

mean the Hessian of l with respect to the elements of ϵ, given θ and evaluated at $\bar{\epsilon}$, det is the determinant and

$$\bar{\epsilon} = \arg\max_{\epsilon} l(\theta, \epsilon). \tag{6.6}$$

That is, $\bar{\epsilon}$ is the value of ϵ that maximises l for a given value of θ. This is a two-level optimisation problem, since evaluating the function l^* requires the optimisation of the function l. l^* is therefore expensive to evaluate, and its derivatives are difficult to obtain. It is possible to compute $\bar{\epsilon}$ inexactly to reduce the cost of evaluating (6.5). This introduces numerical noise, and optimising l^* is therefore a problem well suited to GSS methods. In Table 6.2 we show the results of performing 25 iterations on (6.5) with our GSS method of paper I, with the slight modification that it enforces sufficient decrease rather than simple decrease as in the paper. 25 iterations are usually sufficient to obtain an estimate of τ with a reasonable degree of precision. As a starting value we use the "true" values from which our data sets are generated. Let ∇l denote the gradient of $l(\theta, \epsilon)$ with respect to ϵ. In the heading of Table 6.2 the convergence criterion used for obtaining $\bar{\epsilon}$ is listed. As one can see in the table, the level of accuracy used for obtaining $\bar{\epsilon}$ can be very loose and still only affect the estimated value of τ to little extent. In addition, a loose tolerance translates to less computation time. In the implementation behind the results using GSS, Newton's method was used for maximising $l(\theta, \epsilon)$. It is likely that the relative gains in time can be made bigger by for instance employing truncated Newton instead.

In paper III we were able to work with a proprietary commercially available package for nonlinear statistical models, AD Model builder [33] (ADMB).

	GSS, $\|\nabla l\| \leq 10^{-6}$		GSS, $\|\nabla l\| \leq 1$		ADMB
Problem	τ	time	τ	time	τ
1	1.50266e-01	18.0s	1.50137e-01	13.5s	1.528e-01
2	1.39902e-01	17.8s	1.39901e-01	13.2s	1.395e-01
3	2.31647e-01	14.4s	2.31647e-01	11.4s	2.321e-01
4	2.41657e-01	17.5s	2.41657e-01	13.8s	2.405e-01
5	2.09133e-01	17.1s	2.09133e-01	13.1s	2.091e-01
6	1.83468e-01	17.8s	1.83466e-01	13.5s	1.834e-01
7	1.80755e-01	21.1s	1.80755e-01	16.5s	1.779e-01
8	1.93294e-01	21.8s	1.93294e-01	16.8s	1.924e-01

Table 6.2: Estimates of τ on eight randomly generated data sets using our GSS method with two tolerances for the inner optimisation (6.6), and the time used to obtain the estimate. The estimate in the last column is from AD Model Builder, and should be considered the most accurate for each problem.

ADMB is able, through a combination of AD and hand-coded derivatives to obtain the gradient of the Laplace approximation (6.5), and hence optimise l^* effectively with a quasi-Newton method. Nevertheless, we contend that the problem (6.5) is a useful and realistic example of the applicability of our method to a numerically noisy problem.

Statistical Findings As for the results pertaining to the likelihood estimation itself, we find that it is difficult to obtain an accurate estimate of τ if the true value of τ is low, say, 0.05. If the true value is somewhat larger, e.g. 0.2, we have a better chance of getting a good estimate.

Chapter 7

Concluding Remarks

7.1 Summary

In this thesis we have showed how we can take a very basic and arguably slow GSS algorithm, compass search, and speed it up using average curvature information accumulated over a region. We have shown that the necessary curvature information can be obtained effectively for partially separable functions, in turn leading to a more effective method. In addition, separability is not intrinsically linked to differentiability, which makes it a useful concept for functions which are not differentiable as well.

Numerical testing has indicated that updating the search basis of GSS algorithms using the eigenvectors of a matrix based on curvature information reduces zig-zagging, sometimes significantly.

We have applied one of our methods to a problem from maximum likelihood estimation, showing that the method is applicable to a difficult class of numerically noisy problems.

We have defined a new class of limited memory methods for smooth unconstrained optimisation, and shown that a particular instance of the class performs well compared to its parent methods.

7.2 Possible Further Research

There are several issues we have not been able to address, which could be studied in the future. An obvious avenue of further research is developing effective forcing sequences and schemes for varying m for the class of methods presented in paper V. For GSS methods, some possibilities are outlined below.

Enhancing the Exploratory Moves Our GSS algorithms base themselves on a particular method for their exploratory moves, listed in Paper I. This subroutine searches in $2n$ directions at each iteration. In theory this may lead to points being evaluated more than once, and if there is a clear trend as to where in n-space the method progresses, as many as half of these evaluations if not more may easily be unsuccessful. It should be possible to incorporate a heuristic which tries to minimise the number of unsuccessful evaluations, while still collecting curvature information.

Expanding the Role of the Quadratic Model Function We have so far only used an implicit quadratic model function, since the curvature matrix C we work with can be used as a basis for a family of quadratic functions. We have not tried to exploit a model function further, for instance by using interpolation to predict suitable values for step lengths, or where in n-space to search next. This could possibly speed our methods.

Treating Element Functions individually In our work on separable functions, we have yet to study the case where the individual element functions are available to us. If this were the case, we could build up (average) curvature information about each of the element functions, and then assemble this information in C prior to the eigenvalue-factorisation.

Computing Curvature Information by Approximate Gradients So far, our work on GSS methods has revolved around computing curvature information element by element. It is also possible to apply techniques like those of section 5.3, that is, computing (approximate) Hessian-vector products and exploiting sparsity to obtain the full Hessian at little cost. Hessian-vector products can be obtained with a formula such as the one used by the discrete truncated Newton method. Gradients can be approximated by the formula

$$(Q^T \nabla f(x))_i \approx \frac{f(x + \epsilon q_i) - f(x)}{\epsilon}.$$

The constellation of points needed to computed the entire gradient in the coordinate system defined by the columns of Q can be obtained if a compass search iteration tries many points before accepting a new point. A simple example is given in Figure 7.1. In the figure the algorithm tries three different points before stepping to the fourth point evaluated. The function values of the unsuccessful points can then be used for gradient computation. In this case we have four points around the starting point so that we can compute a central-difference approximation to the derivative. If the function is two times differentiable, then given two such gradient approximations one can either approximate the product of the Hessian with a vector, of create a quasi-Newton-like approximation to the entire Hessian matrix.

This matrix we could then, for instance, eigenvalue-factorise as we do in our current methods, and use the eigenvectors as the new search directions.

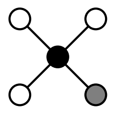

Figure 7.1: Possible outcome after an iteration of compass search.

Bibliography

[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] E. J. Anderson and M. C. Ferris. A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal on Optimization*, 11(3):837–857, 2001.

[3] C. Audet. Convergence results for generalized pattern search algorithms are tight. *Optimization and Engineering*, 5:101–122, 2004.

[4] C. Audet and J. E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 12(2):889–903, 2002.

[5] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995. 2nd edition 1999.

[6] 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.

[7] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17(1):1–13, 1999.

[8] 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.

[9] 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.

[10] D. Byatt, I. D. Coope, and C. J. Price. Conjugate grids for unconstrained optimisation. *Computational Optimization and Applications*, 29:49–68, 2004.

[11] D. Byatt, I. D. Coope, and C. J. Price. Performance of various BFGS implementations with limited precision second-order information. *The ANZIAM Journal*, 45:511–522, 2004.

[12] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

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

[14] B. Colson and P. L. Toint. Exploiting band structure in unconstrained optimization without derivatives. *Optimization and Engineering*, 2:399–412, 2001.

[15] B. Colson and P. L. Toint. Optimizing partially separable functions without derivatives. *Optimization methods and software*, 20(4–5):493–508, 2005.

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

[17] A. R. Conn, K. Scheinberg, and L. N. Vicente. Error estimates and poisedness in multivariate polynomial interpolation. Technical Report Research Report RC 22990, IBM T. J. Watson Research Center, Yorktown, USA, 2003.

[18] A. R. Conn and P. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In Nonlinear Optimization and Applications, G. Di Pillo and F. Giannessi, (Eds.), Plenum, 1995.

[19] I. D. Coope. A conjugate direction implementation of the BFGS algorithm with automatic scaling. *Journal of the Australian Mathematical Society Series B*, 31:122–134, 1989.

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

[21] I. D. Coope and C. J. Price. On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, 11(4):859–869, 2001.

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

[23] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. Technical Report 04-35, Dept. of Mathematics, Univ. of Coimbra, 2004.

[24] W. C. Davidon. Variable metric method for minimization. Technical Report 5990, Argonne National Laboratory, 1959.

[25] W. C. Davidon. Variable metric method for minimization. *SIAM Journal Optimization*, 1(1):1–17, February 1991. With a belated preface for ANL 5990.

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

[27] R. S. Dembo and T. Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.

[28] J. E. Dennis and V. Torczon. Managing approximation models in optimization. In N. M. Alexandrov and M. Y. Hussaini, editors, *Multidisciplinary Design Optimization: State of the Art*, pages 330–347. SIAM, Philadelphia, 1997.

[29] 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.

[30] 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.

[31] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6:163–168, 1963.

[32] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and Practice, Second Edition in C.* The Systems Programming Series. Addison–Wesley, 1997. ISBN 0–201–84840–6.

[33] D. Fournier. An introduction to AD MODEL BUILDER Version 6.0.2 for use in nonlinear modeling and statistics. Available from http://otter-rsch.com/admodel.htm, 2001.

[34] J. C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, 45:407–435, 1989.

[35] N. I. M. Gould, D. Orban, and P. L. Toint. CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. Technical Report RAL–TR–2002–009, Computational Science and Engineering Department, Rutherford Appleton Laboratory, 2002.

[36] A. 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.

[37] A. Griewank and P. L. Toint. On the unconstrained optimization of partially separable functions. In M. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312. 1982.

[38] A. Griewank and P. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39(1):119–137, 1982.

[39] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[40] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8(2):212–229, Apr. 1961.

[41] ICES. Report of the northern pelagic and blue whiting fisheries working group. ICES CM 2002/ACFM19, 2002.

[42] 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.

[43] R. M. Lewis and V. Torczon. Rank ordering and positive bases in pattern search algorithms. Technical Report 96–71, Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681–2199, 1996.

[44] R. M. Lewis and V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, 9(4):1082–1099, 1999.

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

[46] 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.

[47] R. M. Lewis, V. Torczon, and M. W. Trosset. Why pattern search works. *Optima*, 59:1–7, October 1998. Also available as ICASE Technical Report 98–57. ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681–2199.

[48] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[49] 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.

[50] L. Lukšan and E. Spedicato. Variable metric methods for unconstrained optimization and nonlinear least squares. *Journal of Computational and Applied Mathematics*, 124:61–95, 2000.

[51] M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming Series A*, 91(2):289–305, 2002.

[52] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965.

[53] G. N. Newsam and J. D. Ramsdell. Estimation of sparse Jacobian matrices. *SIAM J. Alg. Disc. Meth.*, 4(3):404–417, 1983.

[54] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

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

[56] D. P. O'Leary. A discrete Newton algorithm for minimizing a function of many variables. *Mathematical Programming*, 23(1):20–33, 1982.

[57] Y. Pawitan. *In All Likelihood: Statistical Modelling and Inference Using Likelihood*. Oxford University Press, 2001.

[58] M. J. D. Powell. An efficient method for finding the minimum of a function without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964.

[59] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.

[60] M. J. D. Powell. UOBYQA: Unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[61] M. J. D. Powell. On trust region methods for unconstrained minimization without derivatives. *Mathematical Programming, Series B*, 97:605–623, 2003.

[62] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. Technical Report DAMTP NA2004/08, University of Cambridge, 2004.

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

[64] 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.

[65] D. Shanno and K.-H. Phua. Matrix conditioning and nonlinear optimization. *Mathematical Programming*, 14:149–160, 1978.

[66] H. Skaug and D. Fournier. Evaluating the Laplace approximation by automatic differentiation in nonlinear hierarchical models. Technical report, Inst. of Marine Research, Box 1870 Nordnes, 5817 Bergen, Norway, 2005.

[67] W. H. Swann. Direct search methods. In W. Murray, editor, *Numerical Methods for Unconstrained Optimization*, pages 13–28. Academic Press, London and New York, 1972.

[68] 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.

[69] V. Torczon. On the convergence of the multidirectional search algorithm. *SIAM Journal on Optimization*, 1(1):123–145, 1991.

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

[71] 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.

[72] M. W. Trosset. I know it when I see it: Toward a definition of direct search methods. *SIAG/OPT Views-and-News: A Forum for the SIAM Activity Group on Optimization*, 9:7–10, Fall 1997.

[73] M. Wright. Direct search methods: Once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1995 (Proceedings of the 1995 Dundee biennial Conference in Numerical Analysis)*, pages 191–208. 1996.

[74] W. Yu. Positive basis and a class of direct search techniques. *Scientia Sinica*, Special Issue of Mathematics, 1:53–67, 1979.

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

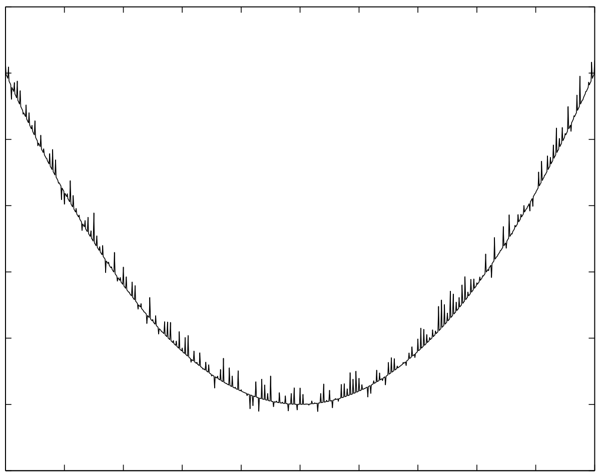

Figure 1.1. Quadratic function with noise.

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 $q_i,\ i = 1 \ldots n$ 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

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

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.

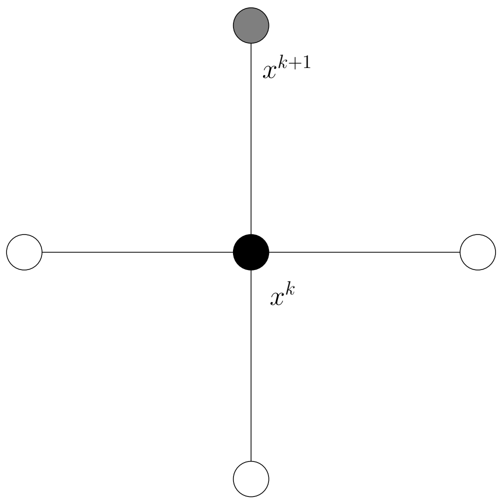

Figure 1.2. Coordinate search in $\mathbb{R}^2$.

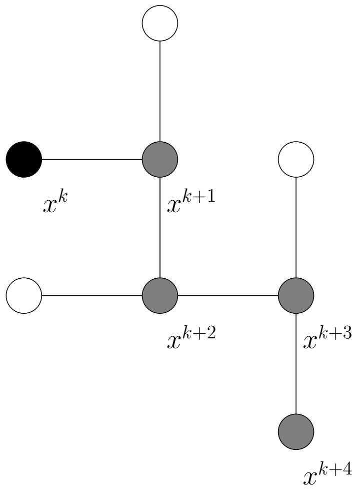

Figure 1.3. Compass Search in $\mathbb{R}^2$: Search along positive and negative coordinate directions, and step immediately if function reduction found.

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,

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

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.

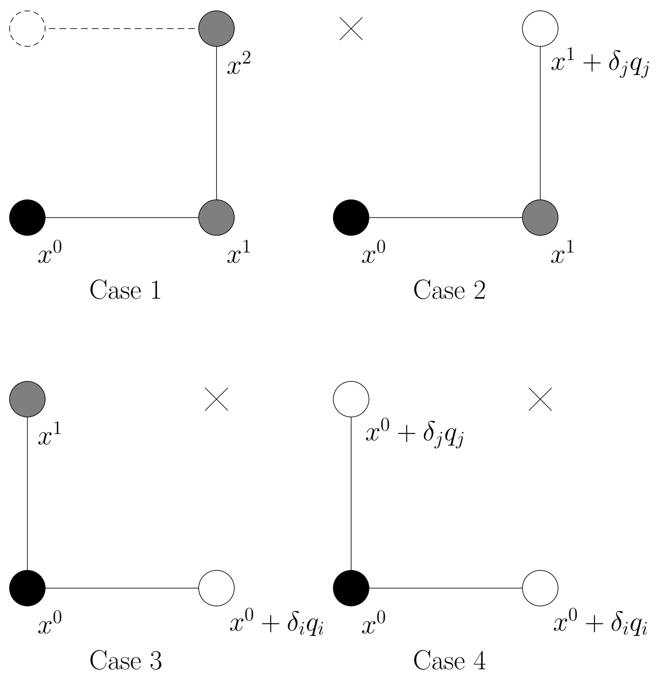

Figure 1.4. The four possible outcomes of successive searches along q_i and q_j. A grey node signifies a step which has been taken, a white node signifies a step not taken. The search starts at the black node in each case.

2.1. COMPUTING THE MATRIX C_Q SYSTEMATICALLY

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

$$\begin{bmatrix} q_i & q_j & -q_i & \cdots & -q_j \end{bmatrix},$$

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

$$\begin{bmatrix} q_i & q_j & -q_i \end{bmatrix}, \tag{5}$$

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 $C \leftarrow Q C_Q Q^T$.
    Eigenvalue-factorise $C$: $C = X \Lambda X^T$.
    Set $\delta \leftarrow 2 \cdot X Q^T \delta$.
    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*

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

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.

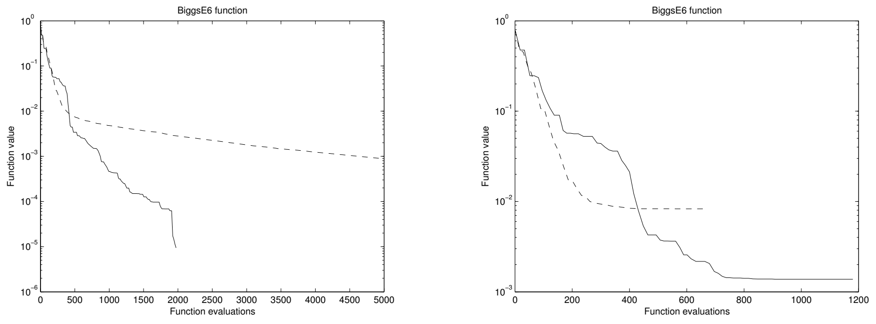

Figure 1.7. Logarithmic plot of function values vs. the number of function evaluations on function #18, without noise added in the left plot, and with noise in the right plot. The solid curve corresponds to the new algorithm, the dashed curve to Compass Search.

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

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

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.

Paper II

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.

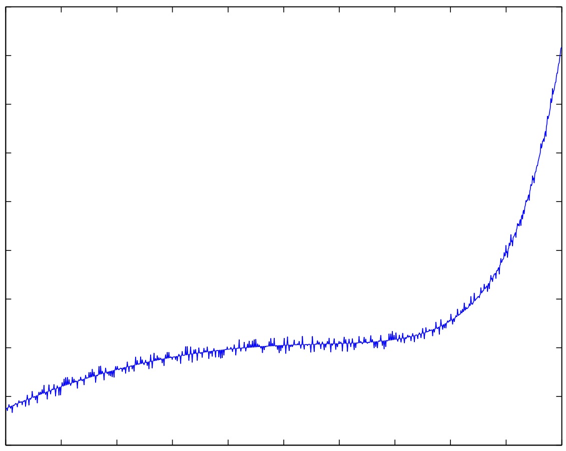

Figure 1: $e^x - x^2$ with noise.

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 $\mathbf{vec}$ 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,

$$C = \left[\begin{array}{cccc} c_1 & c_2 & \cdots & c_n \end{array}\right].$$

Then

$$\mathbf{vec}(C) = \begin{pmatrix} c_1^T & c_2^T & \cdots & c_n^T \end{pmatrix}^T. \tag{12}$$

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 $\overline{\textbf{vec}}$.

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

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

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, $l = 1, \dots, \rho$ 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*

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

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 $h_l = \overline{\mathbf{vec}}(\nabla^2 f(\widetilde{x}^l))$, $l = 1, \dots, \rho$. 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

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

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

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

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

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

By (15) we have

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

This turns (24) into

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

and finally, by (15),

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

□

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 $O(\frac{\varrho}{n})$ 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.

Paper III

Nested optimisation; Application to estimation of variation in annual mortality in fish populations[*]

Lennart Frimannslund[†] Hans Julius Skaug[‡]

March 31, 2006

Abstract

We study a population dynamics model incorporating natural mortality as well mortality due to human exploitation. The model is applicable to Norwegian spring spawning herring. Our goal is to make inference about annual variation in natural mortality. Using the Laplace approximation of the marginal likelihood, we derive a likelihood function for the unknown parameters in the model. The statistical properties of the estimators are investigated using simulated data sets. We do not find evidence for annual variation in mortality for Norwegian spring spawning herring, but our simulation experiments indicate that one would need much more data than currently available to be able to detect such an effect.

Keywords: Laplace Approximation, Monte Carlo-Simulation, Automatic Differentiation, Pattern Search.

1 Introduction

The annual mortality rate M is an important demographic parameter in wildlife populations. The probability that an individual survives from one year to the next is e^{-M}, and this serves to define M. In a fish population there can be large annual variations in M, due to

[*]This work was supported by the Norwegian research council (NFR).

[†]Department of Informatics, University of Bergen, Pb. 7800, 5020 Bergen, Norway.

[‡]Department of Mathematics, University of Bergen, Johannes Brunsgate 12, 5008 Bergen, Norway.

changes in environmental conditions and variation in predation pressure. Denote by $M + \epsilon_t$ the mortality rate in year t, where ϵ_t is a perturbation around the average mortality rate M. We assume that there is no direct measurement of ϵ_t available, and hence we shall view ϵ_t as a stochastic variable with expectation 0 and unknown variance τ^2. We formulate a stochastic population dynamic model, and derive an objective function that allows M and τ to be estimated from catch data and data from scientific surveys. For most marine fish populations such data are scarce, and estimation of M is a difficult problem. Needless to say, estimation of the level of annual variation (τ) is an even harder problem. Hence, with the current level of information we cannot hope to get reliable estimates of τ. In the present paper we use simulations to investigate how much data would need to be available in order for τ to be identifiable with a reasonable degree of precision.

The stochastic population dynamic model we use is an instance of a state-space model. There are two types of unknown quantities in such models: 1) the state variables, which here are the number of individuals being alive each year, and 2) the (structural) parameters: M, τ, along with some other parameters to be defined later. State-space models are often fit to data using Kalman-filter techniques [10]. When the model is non-linear, the equations must be linearised before one can apply the standard Kalman machinery. We use the Laplace approximation [7] to integrate out the state variables from the likelihood function. This leaves us with the marginal likelihood, which becomes our objective function for estimation of M and τ. The Laplace approximation is itself phrased as an optimisation problem, so our approach involves nested optimisation. The inner optimisation problem is solved using a quasi-Newton algorithm. We solve the outer problem using two approaches, quasi-Newton and a pattern search algorithm, the latter of which which allows for the inner problem to be solved inexactly, hence more cheaply.

The rest of the paper is organised as follows. In section 2 we outline the stochastic population dynamic model. In section 3 we outline the computational methods, and in section 4 we apply the method to data for Norwegian Spring Spawning Herring along with simulated datasets, which we discuss in section 5.

2 Population Dynamics of Exploited Fish Stocks

Most of our large fish populations are subject to human exploitation. We assume that the number of individuals C removed from the population each year by fisheries is known. The mortality rate M referred to above is the "natural" mortality, and does not include the mortality caused by the fisheries.

We consider a period of n years, labeled $t = 1, \ldots, n$ for simplicity. Our population consists of A independent cohorts. In real life a cohort consists of all fish born in a particular year, but for simplicity we shall here treat the "cohorts" as being coexisting, but otherwise unrelated, developing populations. The basic equation governing the population dynamics of the jth, $j = 1, \ldots, A$ cohort is

$$N_{j,t} = (N_{j,t-1} - C_{j,t-1})\, e^{-(M+\epsilon_{t-1})}, \quad t = 1, \ldots, n, \tag{1}$$

where the quantities are:

$N_{j,t}$	Number of individuals in cohort j in year t,
$M + \epsilon_t$	Mortality in year t (applies to all cohorts),
$C_{j,t}$	Catches in numbers of individuals in cohort j in year t.

The model specification is completed with the requirement that ϵ_t has a Gaussian distribution with mean 0 and variance τ^2. Note that this assumption allows for $e^{-(M+\epsilon_t)} > 1$, which does not have an interpretation in terms of survival.

2.1 Available Data

In addition to the catch numbers C, data from acoustic scientific surveys are available. These surveys provide relative indices I of population size, in the sense that I is an estimate of $q \cdot N$, where q is a number satisfying $0 < q < 1$. We refer to q as the "catchability" parameter, and it may be given the interpretation that the survey covers only a proportion of the total population. By reading the age of individual fish in a random sample it is possible to calculate a survey index for each cohort. In the Norwegian Spring Spawning Herring data, and in our simulated datasets, there are four surveys each year, each with their own catchability parameter. The key quantities involved are:

$I_{j,s,t}$	Survey index for cohort j in survey s in year t,
q_s	"Catchability" in survey s.

The statistical assumption we make is that $\log(I_{j,s,t})$ has a normal distribution with expectation $\log(q_s \cdot N_{j,t})$ and variance σ^2.

2.2 Likelihood Function

In order to initialise the system (1) we need values for $(N_{1,0}, \ldots, N_{A,0})$, i.e. the state vector at time zero. These values will be estimated along with the other parameters of the model. Hence, the parameters are: $\theta = (M, \tau, \sigma, q_1, \ldots, q_S, N_{1,0}, \ldots, N_{A,0})$. The other independent variables in the model are the parameters dealing with variation in mortality, $(\epsilon_1, \ldots, \epsilon_n)$. Let ϵ (without subscript) denote the vector $(\epsilon_1, \ldots, \epsilon_n)$, and similarly for the other variables. The log-likelihood function, from which we shall construct our objective function, has two parts:

$$\begin{aligned} l(\theta, \epsilon) &= \sum_{t=1}^{n}\sum_{s=1}^{S}\sum_{j=1}^{A}\left[-\log(\sigma) - \frac{(\log(I_{j,s,t}) - \log(q_s N_{j,t}))^2}{2\sigma^2}\right] \\ &\quad + \sum_{t=0}^{n-1}\left[-\log(\tau) - \frac{\epsilon_t^2}{2\tau^2}\right]. \end{aligned} \tag{2}$$

The first part arises from the distributional assumptions made about $I_{j,s,t}$, while the second part comes from the distributional assumptions made about the ϵ_t. Note that the two parts are coupled through (1), where $M + \epsilon_t$ occurs.

2.3 Laplace Approximation

Denote the function (2) by $l(\theta, \epsilon)$ where θ denotes all other independent variables than ϵ. It is well established in the statistical literature (e.g. [6], p. 466) that joint maximisation of l with respect to θ and ϵ does not give a good estimate of θ, and hence not of τ which is the parameter of primary interest to us. Instead, one can use the Laplace approximation [8]

$$l^*(\theta) = -\frac{1}{2}\log\det(-H(\theta)) + l(\theta, \bar{\epsilon}(\theta)), \tag{3}$$

of the marginal log-likelihood

$$l(\theta) = \log\left[\int \exp\{l(\theta, \epsilon)\}\, d\epsilon\right]. \tag{4}$$

In (3), $\bar{\epsilon}(\theta)$ is the maximiser of $l(\theta, \epsilon)$ with respect to ϵ for a fixed value of θ, and the symmetric matrix function H is defined as

$$H(\theta) = \frac{\partial^2}{\partial\epsilon^2} l(\theta, \epsilon)|_{\epsilon=\bar{\epsilon}(\theta)}. \tag{5}$$

Numerical evaluation of $l^*(\theta)$ may be done as follows:

- Maximise $l(\theta, \epsilon)$ with respect to ϵ to obtain $\bar{\epsilon}(\theta)$. This optimisation step is referred to as the "inner optimisation".

- Evaluate H at $\bar{\epsilon}$.
- Compute the determinant of H by means of a Cholesky factorisation and compute the expression (3).

The inner optimisation can be performed efficiently with a quasi-Newton method, with the gradient computed by Automatic Differentiation (AD), and H can also be computed by AD. AD (see, e.g. [3]) is a collection of techniques which can compute derivatives of a function defined through computer code, to machine precision. These techniques are attractive since they are usually transparent to the user, and can compute the gradient of a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ at between four and five times the cost of evaluating the function itself. AD may however require a large amount of storage space.

When $\bar{\epsilon}(\theta)$ does not maximise $l(\theta, \epsilon)$ exactly, in which case we write $\tilde{\epsilon}$, we must include a correction term in the Laplace approximation,

$$l^*(\theta) = -\frac{1}{2}\log\det(-H(\theta)) + l(\theta, \tilde{\epsilon}(\theta)) - \frac{1}{2}\nabla l^T H^{-1} \nabla l, \tag{6}$$

where ∇l is the gradient of $l(\theta, \epsilon)$ with respect to ϵ evaluated at $\tilde{\epsilon}$, and H given by (5) now is evaluated at $\tilde{\epsilon}$. A proof of the result goes as follows. By a second order Taylor expansion, and skipping the argument θ from our notation, we get

$$l(\epsilon) \approx l(\tilde{\epsilon}) + \nabla l^T(\epsilon - \tilde{\epsilon}) + \frac{1}{2}(\epsilon - \tilde{\epsilon})^T H(\epsilon - \tilde{\epsilon}). \tag{7}$$

By algebraic manipulation we find that

$$\begin{aligned}&(\epsilon - \tilde{\epsilon} + H^{-1}\nabla l)^T H(\epsilon - \tilde{\epsilon} + H^{-1}\nabla l)\\ &\qquad = (\epsilon - \tilde{\epsilon})^T H(\epsilon - \tilde{\epsilon}) + 2\nabla l^T(\epsilon - \tilde{\epsilon}) + \nabla l^T H^{-1}\nabla l,\end{aligned}$$

which can be used to rewrite the Taylor expansion (7) as

$$l(\epsilon) = l(\tilde{\epsilon}) - \frac{1}{2}\nabla l^T H^{-1}\nabla l + \frac{1}{2}(\epsilon - \tilde{\epsilon} + H^{-1}\nabla l)^T H(\epsilon - \tilde{\epsilon} + H^{-1}\nabla l).$$

Further, we have the multivariate normal integral

$$\int \exp\left[\frac{1}{2}(\epsilon - v)^T H(\epsilon - v)\right] d\epsilon = c \cdot \det(-H)^{-1/2}, \tag{8}$$

where $c = (2\pi)^{n/2}$ is a constant that can be ignored in the present context. Since (8) holds for all values of v, and in particular

$$v = \tilde{\epsilon} - H^{-1}\nabla l,$$

the approximation (6) of the marginal likelihood (4) follows (after a few lines of thought).

3 Optimising the Likelihood Function

We consider two different methods for optimising the objective function (3). The first approach is to solve the problem by using the Quasi-Newton solver that is built into AD Model builder [1] (ADMB), a commercially available package for nonlinear statistical models. The second is to use a variant of the pattern search method of [2] for the outer problem, and the BFGS method to solve the inner problem, with gradients and H computed by the ADOL-C package [4].

A difference between the two approaches from a theoretical point of view is that the former requires the gradient of $l^*(\theta)$, which involves third order mixed derivatives of $l(\theta, \epsilon)$ [8], whereas the latter only requires the second derivative $H(\theta)$. When using AD, the computation of the gradient of a functional ($l^*(\theta)$ in our case) can be done with less than or equal to five times the amount of work required to compute the function value itself, and thus our problem appears well suited for a gradient-based method. The price to pay for the "cheap" gradient is that one has to store a computational graph (or an execution trace, sometimes called a *tape*) the size of which can be substantial in the sense that it can be larger than the available disk space. The size of the tape depends on the number and nature of the operations required to compute the function value.

In our case, the length of the vector ϵ, n, is important. In order to obtain the gradient of (3) we need to differentiate the Cholesky factorisation of the Hessian H, whose dimension is $n \times n$. The computational graph, and corresponding overhead, used in computing ∇l^* will therefore grow as n grows, and make the computation of ∇l^* more cumbersome. The principle that a gradient can be obtained at five times the cost in operation count of the function still applies, but the storage requirements may be substantial for large n, a problem which is not encountered on the same scale when calculating function values only.

In our context, however, since we only have one time step per year (that is, the formula (1) is only applied one time for each year considered) and fish have a limited life span, we do not expect ϵ to have significantly more elements than 20. Consequently, the computational graphs involved are of acceptable size on a personal computer, and the gradient-based method performs well. In addition, the ADMB package contains a differentiated version of the Cholesky algorithm (see e.g. [9]), which reduces storage requirements.

As for pattern search, its ability to cope with non-smoothness means that one may solve the inner optimisation problem inexactly initially and solve it more and more accurately as one approaches the solution of the outer problem (6). This reduces the total time of the optimisation, since inexact function evaluations can be performed at a relatively low cost. The suggested pattern search method must be either be modified slightly to handle constraints, or one can handle the constraints by using Lagrange multipliers and returning infinity (or negative infinity) for points outside the domain of the function.

Both methods seem to benefit from the two-phase strategy outlined in [7]. The two-phase strategy should not be confused with the nested (inner-outer) optimisation scheme that is common to both the methods we discuss. In phase I the objective function is taken to be (2), but with $\epsilon = 0$ and τ fixed at some initial value. In particular, there is no Laplace approximation involved in the first phase. Note that the second term in (2) now can be ignored. Phase I hence provides estimates for all components of θ except τ. These estimates are used as initial values for phase II in which the objective function is taken to be the Laplace approximation (3).

Summing up, both methods are applicable to the problem; in our context the ADMB package is faster than our experimental codes, so we use the former in the next section.

4 Simulation Experiments

The question we ask in this section is: what type of, and how much, data do we need to be able to estimate τ? For this purpose we generate artificial data from the model (1) via Monte Carlo simulation. Hence we know what the true parameter values θ are. Then, we fit the model to the simulated data, as explained above and obtain an estimate $\widehat{\theta}$. This procedure is repeated many times, and we can measure the statistical properties (mean and standard deviation) of the estimator $\widehat{\theta}$. The variable of main interest to us is τ, so we created 1,000 data sets for each of the values

$$\tau_{\text{real}} = \{0.05,\ 0.1,\ 0.2\},$$

where in addition

$$N_0 = \left[\begin{array}{cccc} 200 & 200 & 200 & 200 \end{array}\right]^T,$$

(implying four cohorts)

$$q = \left[\begin{array}{cccc} 0.5 & 0.5 & 0.5 & 0.5 \end{array}\right]^T,$$

implying four surveys, and finally

$$\sigma = 0.2,\ M = 0.15,\ n = 20.$$

a: $\tau_{\text{true}} = 0.05$

Data available	Mean($\widehat{\tau}$)	Std($\widehat{\tau}$)	# $\tau_{\min}$
50%	0.0181 (0.0559)	0.0274 (0.0183)	689
75%	0.0246 (0.0528)	0.0281 (0.0166)	544
100%	0.0297 (0.0488)	0.0265 (0.0160)	400

b: $\tau_{\text{true}} = 0.1$

Data available	Mean($\widehat{\tau}$)	Std($\widehat{\tau}$)	# $\tau_{\min}$
50%	0.0782 (0.0899)	0.0398 (0.0280)	132
75%	0.0881 (0.0927)	0.0319 (0.0254)	50
100%	0.0890 (0.0912)	0.0280 (0.0245)	25

c: $\tau_{\text{true}} = 0.2$

Data available	Mean($\widehat{\tau}$)	Std($\widehat{\tau}$)	# $\tau_{\min}$
50%	0.1870 (0.1908)	0.0424 (0.0373)	3
75%	0.1869	0.0386	0
100%	0.1905	0.0377	0

Table 1: Numerical results (1000 Monte Carlo replica) for different values of τ_{true} The numbers in parentheses show the results when only the instances where $\hat{\tau} \neq \tau_{\min}$ are included.

4.1 Results

The results are given in Table 1 a)–c). The columns of the tables signify, from left to right, the amount of survey data available, mean and standard deviation of $\widehat{\tau}$ and the number of times where $\widehat{\tau}$ was equal to the lower bound ($\tau_{\min} = 10^{-3}$) set by the optimisation algorithm. The table also shows results when the cases where $\hat{\tau}$ is equal to the lower bound are excluded. By available survey data, we mean the percentage of the $4 \cdot 4 \cdot 20 = 320$ acoustical observations available, where which observations are available is randomly selected.

5 Discussion

From Table 1 we draw the following conclusions:

- The more available data, the closer the mean of $\hat{\tau}$ is to τ_{true}.

- The more available data, the smaller the standard deviation of $\hat{\tau}$.
- Using only the cases where $\hat{\tau} \neq \tau_{\min}$ reduces the bias in the estimator.
- The larger the value of τ_{true}, the fewer cases of $\hat{\tau} = \tau_{\min}$.
- The larger the value of τ_{true}, the larger the standard deviation of $\hat{\tau}$.

We also applied the method to time series data from Norwegian spring spawning herring [5], which resulted in an estimate of $\tau = 0$. Apparently, there is no year-to-year variation in mortality, but an important point is that the uncertainty associated with the estimate is large. The objective function (3) is flat near its optimum, which is actually located at the boundary of the parameter space (τ must be non-negative). Hence, data provided little information about the true value of τ, which is what we expected. The simulation results presented in Table 1 support this conclusion, in that the probability that $\hat{\tau}$ ends up at zero is high, particularly when τ is small, like $\tau = 0.05$.

References

[1] D. Fournier. An introduction to AD MODEL BUILDER Version 6.0.2 for use in nonlinear modeling and statistics. Available from http://otter-rsch.com/admodel.htm, 2001.

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

[3] A. 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.

[4] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, June 1996.

[5] ICES. Report of the northern pelagic and blue whiting fisheries working group. ICES CM 2002/ACFM19, 2002.

[6] Y. Pawitan. *In All Likelihood: Statistical Modelling and Inference Using Likelihood.* Oxford University Press, 2001.

[7] H. Skaug and D. Fournier. Evaluating the Laplace approximation by automatic differentiation in nonlinear hierarchical models.

Technical report, Inst. of Marine Research, Box 1870 Nordnes, 5817 Bergen, Norway, 2005.

[8] H. Skaug and D. Fournier. Automatic approximation of the marginal likelihood in non-gaussian hierarchical models. Technical report, 2006. To appear in Computational Statistics and Data Analysis.

[9] S. P. Smith. Differentiation of the Cholesky algorithm. *Journal of Computational and Graphical Statistics*, 4(2):134–147, 1995.

[10] M. West and P. Harrison. *Bayesian Forecasting and Dynamic Models.* Springer-Verlag, New York, 1997.

Paper IV

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,

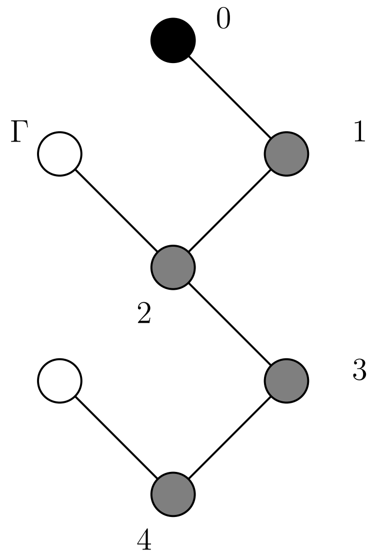

Figure 2: Compass search in $\mathbb{R}^2$ along vectors 45° to the coordinate axes.

it partitions the available directions into triples T of the form

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

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

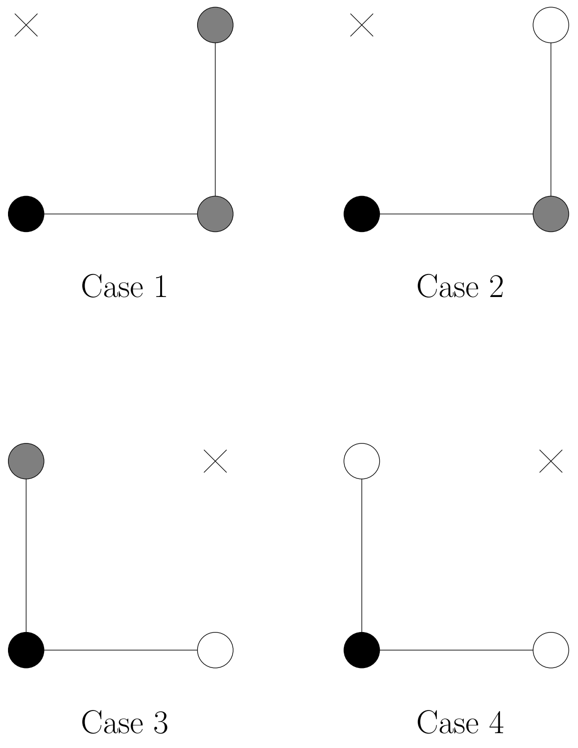

Figure 3: The four possible outcomes of successive searches to the east and north. A grey node signifies a step which has been taken, a white node signifies a step not taken. The search starts at the black node in each case.

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,

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

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 $e_1, \ldots, e_n$, 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. $\square$

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

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

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

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

so that

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

which implies

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

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

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

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. $\square$

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 over-determined 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.

Paper V

A class of Methods Combining L-BFGS and Truncated Newton[*]

Lennart Frimannslund[†] Trond Steihaug[‡]

March 31, 2006

Abstract

We present a class of methods which is a combination of the limited memory BFGS method and the truncated Newton method. Each member of the class is defined by the (possibly dynamic) number of vector pairs of the L-BFGS method and the forcing sequence of the truncated Newton method. We exemplify with a scheme which makes the hybrid method perform like the L-BFGS method far from the solution, and like the truncated Newton method close to the solution. The cost of a method in the class of combined methods is compared with its parent methods on different functions, for different cost schemes, namely the cost of finite difference derivatives versus AD derivatives, and whether or not we can exploit sparsity. Numerical results indicate that the example hybrid method usually performs well if one of its parent methods performs well, to a large extent independent of the cost of derivatives and available sparsity information.

Keywords: Limited memory BFGS, truncated Newton, automatic differentiation, sparsity.

1 Introduction

We consider the unconstrained optimisation problem

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1.1}$$

[*]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

where $f : \mathbb{R}^n \to \mathbb{R}$ is two times continuously differentiable. If the Hessian is available one may use Newton's method for solving (1.1). At the heart of a Newton iteration is the solution of the step equation,

$$\nabla^2 f(x^k)\; p^k = -\nabla f(x^k). \tag{1.2}$$

If only gradients are available, one can use quasi-Newton methods, such as the BFGS method, which typically maintains an approximation H to the inverse of the Hessian, which gives the step equation

$$-H^k \nabla f(x^k) = p^k, \tag{1.3}$$

See e.g. [12] and the references therein. Both Newton's method and quasi-Newton methods require $O(n^2)$ storage. If memory is limited and the product of the Hessian with an arbitrary vector is available, one may use an iterative method to solve (1.2). Solving (1.2) full accuracy at each iteration corresponds to the regular Newton's method. Alternatively, one can solve (1.2) inexactly when far form the solution, and to an increasing degree of accuracy as one approaches the optimal solution. The latter method is called a truncated Newton (TN) method [6, 7]. If only gradients are available and memory is limited, one may use a discrete truncated Newton method (DTN) such as in [19, 7]. DTN uses gradients to approximate the product of the Hessian with an arbitrary vector. An alternative is a limited memory quasi-Newton method such as the limited memory BFGS (L-BFGS) method [17, 11, 8], which maintains a user-defined portion of the information contained in the Hessian approximation of the full BFGS method.

Several attempts have been made to create a method which combines the properties of the (discrete) truncated Newton method and the L-BFGS method. It has been proposed to use the difference pairs accumulated by L-BFGS as a preconditioner for the iterative solution of the step equation [16], using difference pairs from within a conjugate gradient (CG) solution process to provide fresh difference pairs for the L-BFGS method [3, 13], and using inexact or exact Hessian information to create an incomplete or modified Cholesky preconditioner for determining the matrix H_0^k used by the L-BFGS method [10, 20]. This idea stems from proposition 1.7.3 in [1]. In this paper we present a novel class of methods which combines the truncated Newton method and the L-BFGS method. We determine the matrix H_0^k using Hessian information in conjunction with an iterative equation solver such as the conjugate gradient (CG) method. The reason we consider the L-BFGS specifically, is that is has the advantage over other limited memory quasi-Newton methods that the product involving H_0 is isolated from the rest of the computations [2]. Specifically, we will numerically test the performance of a method in the class which behaves like the L-BFGS method when far from the solution, and like truncated Newton when close to the solution. The reason for this is the same logic that lies behind truncated Newton, that accurate second-order information is more important close to the solution than far from it.

This paper is organised as follows. In section 2 we describe the truncated Newton method, BFGS and L-BFGS. In section 3 we outline the hybrid approach, give numerical results in section 4 and give some concluding remarks in section 5.

2 Truncated Newton and Limited Memory BFGS

A truncated Newton method makes use of an iterative solver to solve (1.2), inexactly. Let $\tilde{p}^k$ be an inexact solution to (1.2). If the relative residual of the step equation, that is,

$$\frac{\|\nabla^2 f(x^k)\tilde{p}^k + \nabla f(x^k)\|}{\|\nabla f(x^k)\|},$$

is forced to conform to a forcing sequence, a sequence $\eta^k,\ k = 1, \ldots$ where

$$\eta^k < 1 \quad \text{for all } k,$$

then the truncated Newton method can be shown to converge [6]. An example of an effective forcing sequence is

$$\frac{\|\nabla^2 f(x^k)\tilde{p}^k + \nabla f(x^k)\|}{\|\nabla f(x^k)\|} \leq \eta^k, \quad \eta^k = \min\left\{1/k, \|\nabla f(x^k)\|\right\}, \tag{2.1}$$

which was introduced in [7]. Most iterative solvers do not require an explicit representation of the coefficient matrix itself, instead only the product of the matrix with an arbitrary vector. Discrete Newton methods take advantage of this by observing that

$$\nabla^2 f(x)v \approx \frac{\nabla f(x + \epsilon v) - \nabla f(x)}{\epsilon}, \tag{2.2}$$

and use this finite difference estimate as the product needed by the iterative equation solver. If one combines these two techniques, one gets a discrete truncated Newton method.

Limited-memory BFGS or L-BFGS [17, 11] is a modification of the BFGS method. BFGS usually works by by maintaining an approximation to the inverse of the Hessian. It performs a rank-two update at each iteration. Specifically, given the approximation H^k at iteration k,

$$H^k \approx \left(\nabla^2 f(x^k)\right)^{-1},$$

and given

$$y^k = \nabla f(x^{k+1}) - \nabla f(x^k) \quad \text{and} \quad s^k = x^{k+1} - x^k,$$

then H^{k+1} is taken to be

$$H^{k+1} = (I - \rho^k s^k (y^k)^T) H^k (I - \rho^k y^k (s^k)^T) + \rho^k s^k (s^k)^T, \qquad (2.3)$$

where

$$\rho^k = \frac{1}{(y^k)^T s^k}.$$

The initial approximation to the inverse of the Hessian is up to the user. From the update formula (2.3) one can say that BFGS "remembers" the effect of all difference pairs $(y^i, s^i),\ i = 1, \ldots, k$. In L-BFGS the number of difference pairs (y^i, s^i) is a user-defined parameter, and only the most recent, say, m differences are kept. This leads to significant reduction in memory usage when $m \ll n$, since the product of H^k with an arbitrary vector can be computed from a two-loop formula without the need for storing an $n \times n$ matrix, only storing m difference pairs. If $k < m$, then the method uses the k difference pairs it has available.

L-BFGS Update Formula Let μ be a vector of the appropriate length, and μ_i be its ith component. The two-loop procedure for computing $r = H^k v$ is given below:

```
$q = v$
for $i = k - 1$ step -1 until $k - m$
    $\mu_i = \rho^i \cdot (s^i)^T q$
    $q = q - \mu^i y^i$
end
$r = H_0^k q$
for $i = k - m$ step 1 until $k - 1$
    $\beta = \rho^i \cdot r^T y^i$
    $r = r - (\beta - \mu_i) s^i$
end
```

Note that the choice of matrix H_0^k is, again, not defined but up to the user. One may choose $H_0^k = I$ to avoid an extra matrix-vector product, but a choice which has proved to be successful is to set

$$H_0^k = \frac{(s^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}} \cdot I, \tag{2.4}$$

which is supported by numerical testing in [11], where several choices for H_0 are tested. In [8] similar numerical tests indicate that a diagonal matrix based of the diagonal of a BFGS-type approximation to the Hessian may perform even better than (2.4).

The reduced cost of the above formula compared to regular BFGS comes at the cost of convergence rate, which is linear for the L-BFGS method. L-BFGS can also require a very large number of iterations on ill-conditioned problems.

Globalization For Newton-based methods to be globally convergent, that is, converge to the solution from an arbitrary starting point, one approach is to use line searches. We will use line searches satisfying the strict Wolfe conditions (see e.g. [18], chapter 3.1), that is, given the solution to (1.2), at iteration k then the next iterate, x^{k+1} is taken to be

$$x^{k+1} = x^k + \alpha^k p^k,$$

for α^k satisfying

$$f(x + \alpha^k p^k) \leq f(x^k) + c_1 \alpha^k \nabla f(x^k)^T p^k, \tag{2.5}$$

and

$$|\nabla f(x^k + \alpha^k p^k)^T p^k| \leq c_2 |\nabla f(x^k)^T p^k|, \tag{2.6}$$

with $c_1 = 10^{-4}$ and $c_2 = 0.9$ [11]. In our experiments we will use a line search procedure based on that of [15].

3 Hybrid Approach

Consider the assignment

$$r = H_0^k q, \tag{3.1}$$

between the two for-loops in the L-BFGS formula for $r = H^k v$. If H_0^k were the inverse of the Hessian at x^k, then (3.1) would correspond to

$$\nabla^2 f(x^k)\ r = q. \tag{3.2}$$

Let $\tilde{r}$ be an inexact solution to (3.2), with relative residual

$$\frac{\|q - \nabla^2 f(x^k)\tilde{r}\|}{\|q\|}.$$

We wish to combine TN with L-BFGS by replacing r from (3.1) in the two-loop formula with $\tilde{r}$, obtained by applying an iterative equation solver to (3.2). As a starting point for the iterative method we will use the formula normally used by L-BFGS, that is

$$\frac{(s^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}} \cdot q. \tag{3.3}$$

This gives the following algorithm:

```
Given $x^0,\ k = 0$.
**while** $\|\nabla f(x^k)\| >$ tolerance,
    $k \leftarrow k + 1$.
    $v = -\nabla f(x^k)$.
    Perform first L-BFGS for-loop, resulting in vector $q$.
    Solve $\nabla^2 f(x^k) r = q$ to some tolerance with (3.3) as the
    initial guess, resulting in vector $\tilde{r}$.
    Set $r = \tilde{r}$.
    Perform second L-BFGS loop, resulting in search direction $p^k$.
    Find $\alpha^k$ satisfying (2.5) and (2.6).
    Set $x^{k+1} \leftarrow x^k + \alpha^k p^k$.
    Optionally adjust $m$.
    Update difference pairs $(s^i, y^i),\ i = \max\{1, k - m\}, \ldots, k$.
**end**
```

If $m = 0$ the code and (3.2) is solved to tolerance (2.1), then the code reduces to truncated Newton with (2.1) as forcing sequence. If no iterations are performed on (3.2) and the initial solution (3.3) is returned, then the code becomes L-BFGS. In our numerical experiments we for the hybrid method use the rule that (3.2) should be solved to tolerance

$$\frac{\|q - \nabla^2 f(x^k)\tilde{r}\|}{\|q\|} \leq \tau \tag{3.4}$$

where k is the (outer) iteration number, and τ is given by the rule:

- If $\|q\| \geq 1$, $\tau = 1$.
- If $\|q\| < 1$, $\tau = \max\{1/k, \|q\|\}$.

Note that this particular rule should not necessarily be used for $m = 0$, were we recommend (2.1) instead. In our experiments we use (2.1) for the truncated Newton method. See [6].

In our experiments we solve (3.2) with the conjugate gradient method. We do not require in general that the number of difference pairs m is constant, but in our experiments we use $m = 3$.

4 Numerical Testing

We wish to test the theoretical cost of the chosen hybrid method compared to the theoretical cost of the corresponding L-BFGS and truncated Newton methods. We calculate the following costs:

- L-BFGS: One function evaluation and one gradient evaluation per iteration.
- Truncated Newton: One function evaluation, one gradient evaluation, and a variable amount of Hessian-vector products per iteration.
- Hybrid method: One function evaluation, one gradient evaluation, and a variable amount of Hessian-vector products per iteration.

When it comes to line search, in our initial experiments the value $\alpha^k = 1$ was usually accepted. If $\alpha^k = 1$ is the first test value for α^k [11], then one does not need interpolation or similar procedures which incur extra cost most of the time. If $\alpha^k = 1$ is not accepted, then the cost of a line search may vary to a very large extent depending on the implementation. In our experiments we add 1/10 times the cost of one gradient and one function evaluation to the cost of an iteration, which corresponds to one extra function and gradient evaluation by the line search once every ten iterations. When it comes to the cost of gradients and Hessian-vector products, we test four different situations. These are whether or not AD is available, and whether or not the sparsity structure of the Hessian can be exploited. If the Hessian is sparse and the sparsity structure is known, then the Hessian can be obtained cheaply from a compressed Hessian matrix with techniques like that of Curtis, Powell and Reid (CPR) [4]. If CPR techniques are available we take the view that if the number of Hessian-vector products needed in the iterative method is larger than ρ, the number of products needed to determine $\nabla^2 f$ from a compressed Hessian matrix, then the cost is only that of ρ such products. This may or may not be a realistic view, but this depends on the relative cost of derivative computations to the operations used to form Hessian vector products

Case	AD	Compr. Hess.	$C(\nabla f)$	$C(\nabla^2 f \cdot v)$
1	No	Yes	$nC(f)$	$(n+1)C(f)$
2	No	No	$nC(f)$	$(n+1)C(f)$
3	Yes	Yes	$5C(f)$	$12C(f)$
4	Yes	No	$5C(f)$	$12C(f)$

Table 4.1: The four scenarios plotted for each test function.

from a compressed Hessian. We take the view that when AD is available, the gradient can be computed at five times the cost of a function evaluation, and that a Hessian-vector product costs 12 function evaluations (see e.g. [18], chapter 7.2). Actual costs in time and memory are discussed in [14], and these numbers are in accordance with the numbers used here. Define $C(f)$ as the cost of one function evaluation. As mentioned, for each problem we test four situations. These are listed in Table 4.1. The first column in the table lists the number for each case. If we for instance look at case 1, then we see that AD is not available, but we can determine Hessian-vector products from a compressed Hessian. Since AD is not available, we calculate the cost of gradients and Hessian vector products as they would be if computed with finite difference formulas. The cost of a gradient is then usually $(n+1)$ times that of a function evaluation, but since we (in the context of an iterative optimisation method) already have the function value, we set the cost to be n times that of a function value. Similarly for Hessian vector products, this usually requires two gradients, but since we already have one of the necessary gradients in the optimisation method, we write only the cost of one gradient. This cost scheme covers relative costs of gradients to Hessian vector products in the discrete truncated Newton method, where a Hessian vector product costs the same as an (extra) gradient evaluation. Similarly, it covers the situation where an analytically available gradient costs the same as an analytically available Hessian-vector product if we ignore the relative cost of derivatives to the cost of function values.

We test the three methods on the three problems of [5], as well as six problems from the CUTEr collection [9], with convergence criterion

$$\|\nabla f(x)\| \leq 10^{-4}.$$

For each problem we list four figures, with equivalent cost in function evaluations along the x-axis, and the norm of the gradient along the y-axis. For each problem we have AD-derivatives or hand-coded derivatives, so the costs in the figures are estimates. The four figures correspond to the four cases of Table 4.1, with case 1 and 2 in the first row, and 3 and 4 in the second. The dash-dotted, sometimes oscillating curve corresponds to L-BFGS, the solid line with stars (one star for each outer iteration) corresponds to truncated Newton, and the dashed line corresponds to the hybrid approach. On the HILBERTB function (Figure 4.7) the L-BFGS and hybrid curves are indistinguishable.

On the first three functions, when CPR techniques are available the curves corresponding to the truncated Newton method take sharp dips. This sometimes happens for the hybrid method as well, notably on problem two, but at a slightly later stage than for TN. When CPR techniques are not available the methods perform quite similarly, with the hybrid method frequently in second place, regardless of which method is the fastest.

On the extended Rosenbrock function the curve of the hybrid method is takes a dip when that of TN does, at a slightly later stage. When CPR techniques are not available, it performs as good as identically to L-BFGS (which performs the best) when AD is not available, and close to it when AD is available (case 4).

On the DIXMAANL problem, the hybrid method performs similarly to TN, which performs the best in three of four cases.

On the chained Rosenbrock problem the hybrid method performs the best in case two, slightly worse than L-BFGS in case 4 and is able to benefit from CPR techniques like it did on the second problem, that is, in a similar fashion as TN but at a slightly later stage.

On the HILBERTB problem, the hybrid method together with L-BFGS performs the best in all cases.

On the penalty I function we get very interesting curves, which are in a sense optimal for a hybrid method. Initially, the hybrid method follows the steepest curve, namely that of L-BFGS, and when L-BFGS stagnates continues at the pace of TN.

A similar, but less obvious picture appears for the penalty II function.

5 Concluding Remarks

We have presented a class of hybrid L-BFGS/TN methods, and tested the performance of one of the members in the class compared with its corresponding parent methods. In some of our tests, the hybrid method followed the best curve of its parent methods closely, in other tests it fell between its two parents.

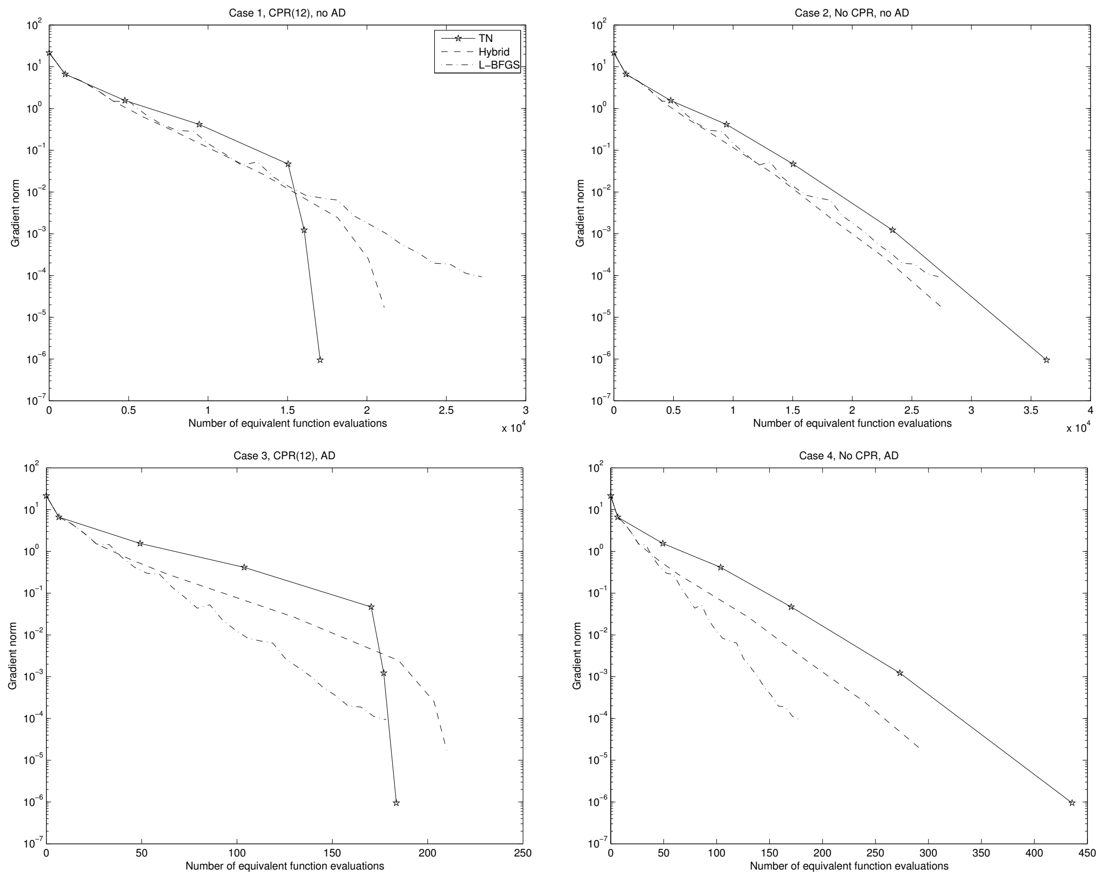

Figure 4.1: Problem 1 from [5], well conditioned, $\rho = 12,\ n = 916$.

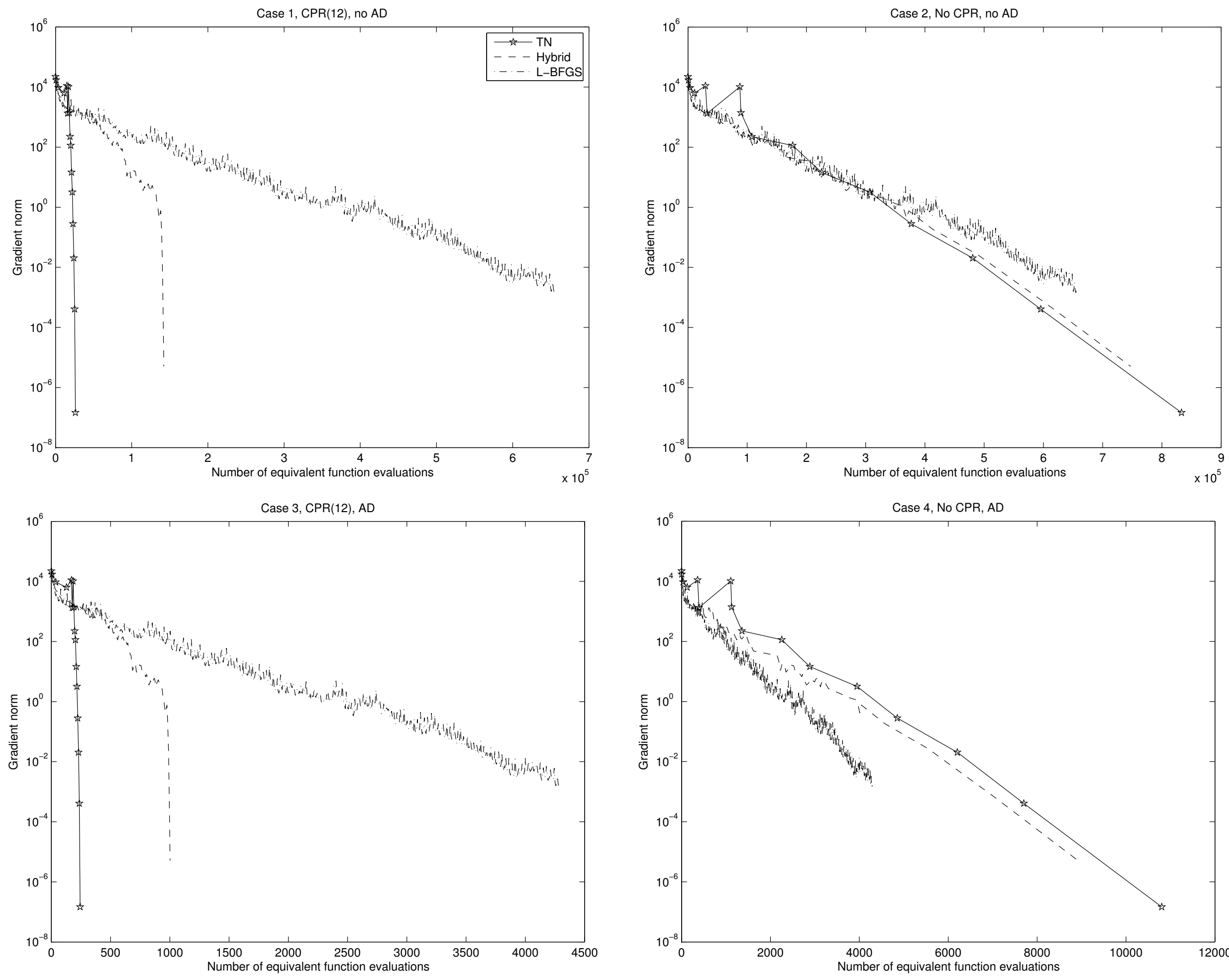

Figure 4.2: Problem 2 from [5], ill-conditioned, $\rho = 12,\ n = 916$.

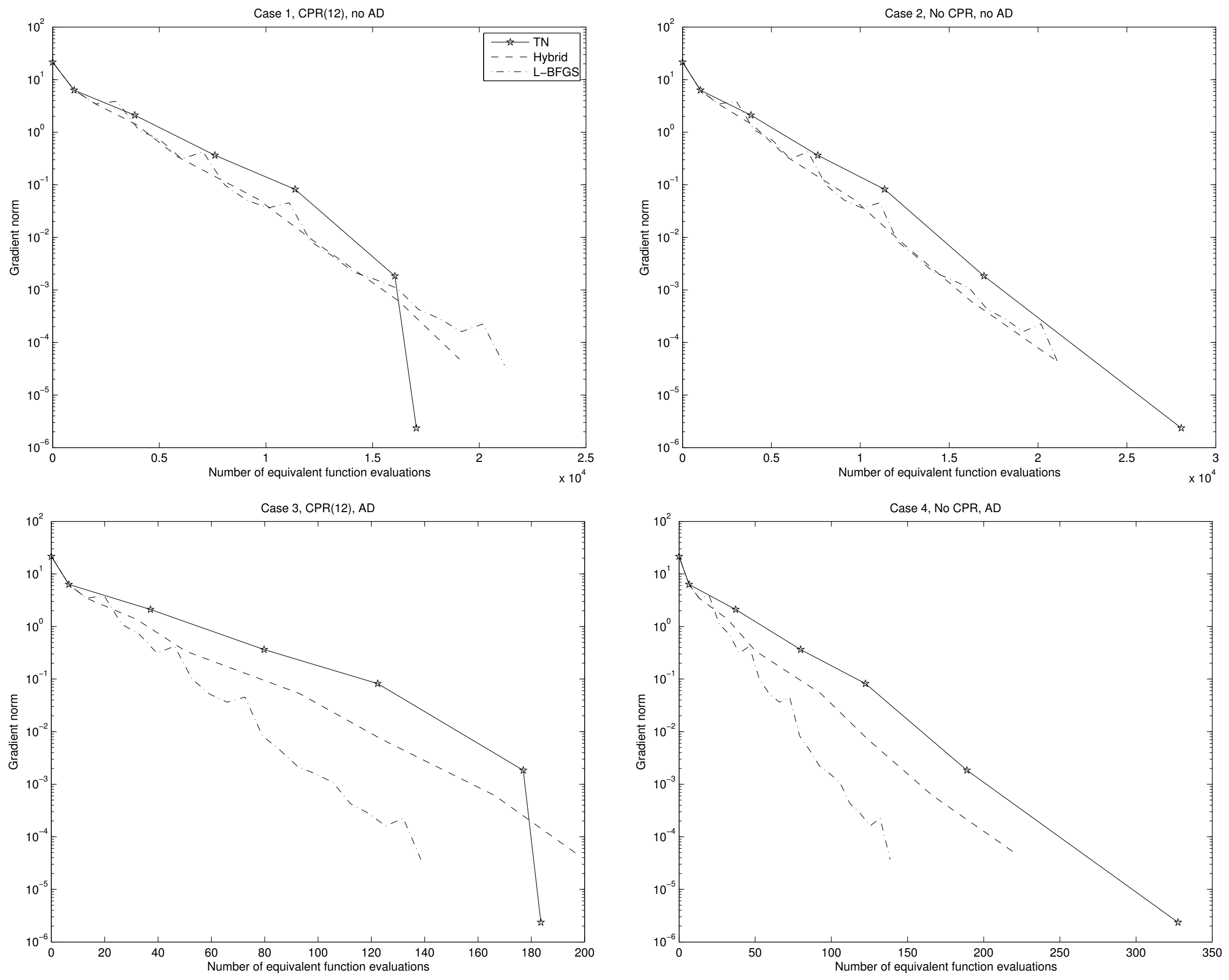

Figure 4.3: Problem 3 from [5], quadratic, $\rho = 12,\ n = 916$.

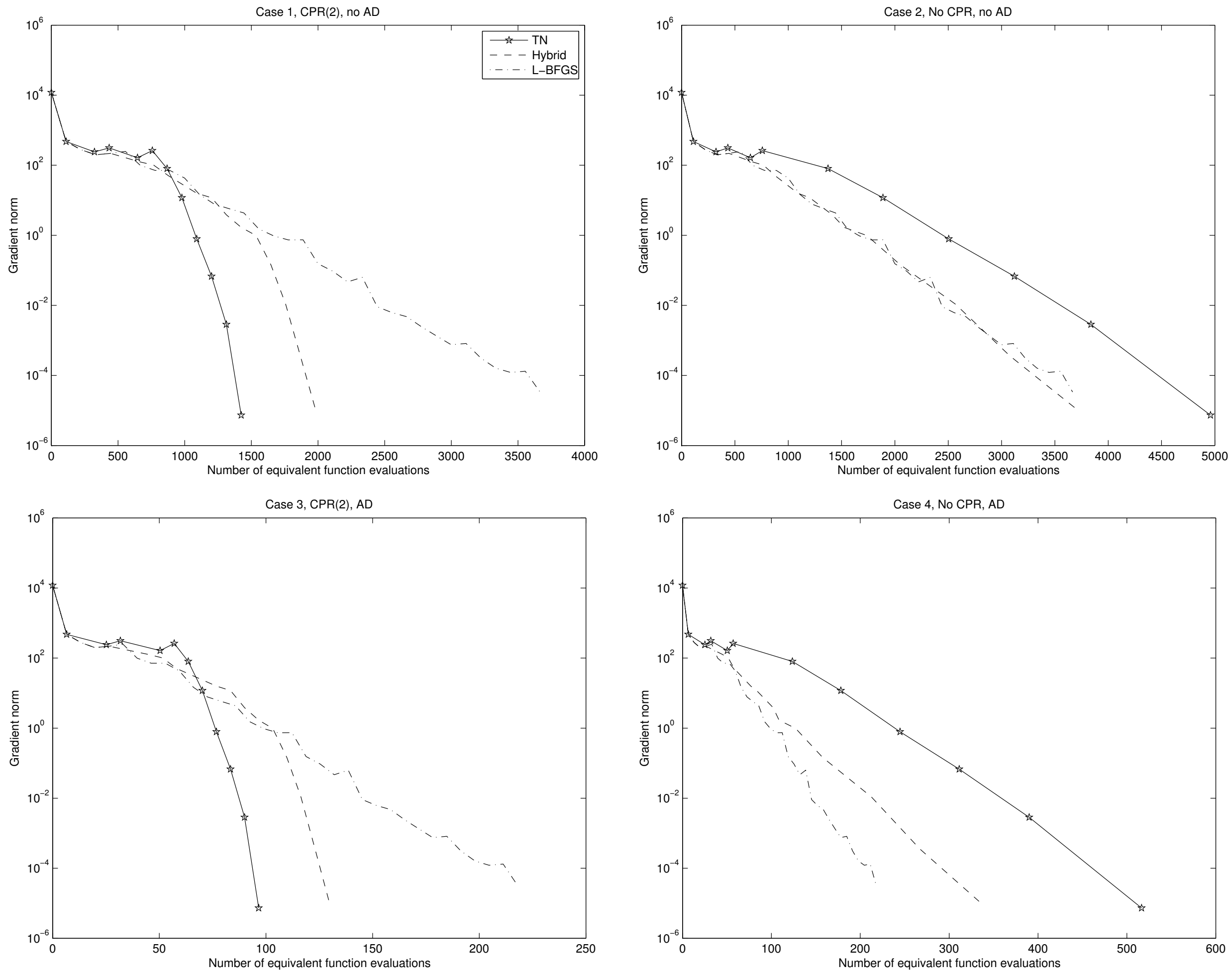

Figure 4.4: Extended Rosenbrock (EXTROSNB), $\rho = 2,\ n = 100$.

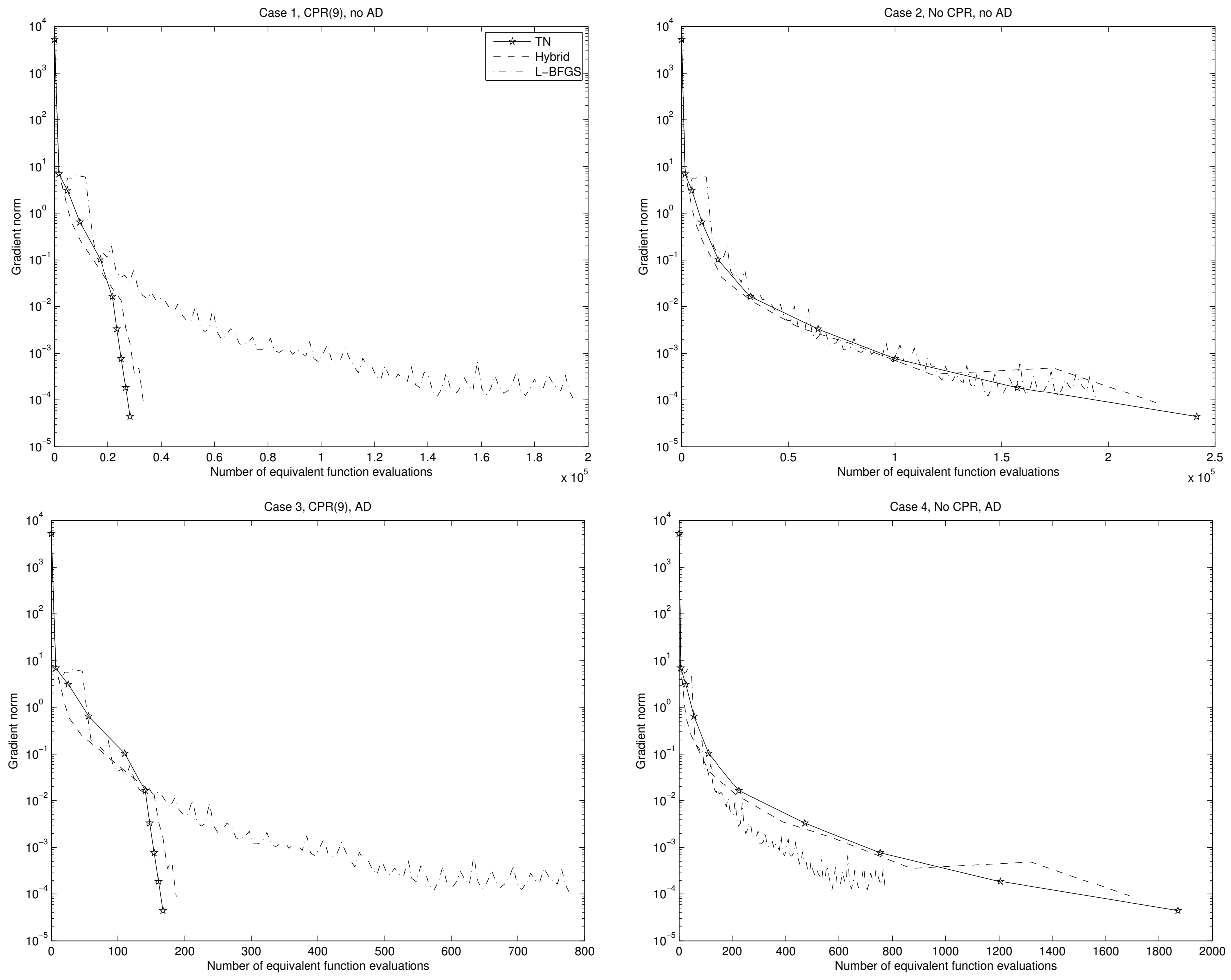

Figure 4.5: DIXMAANL, $\rho = 9,\ n = 1500$.

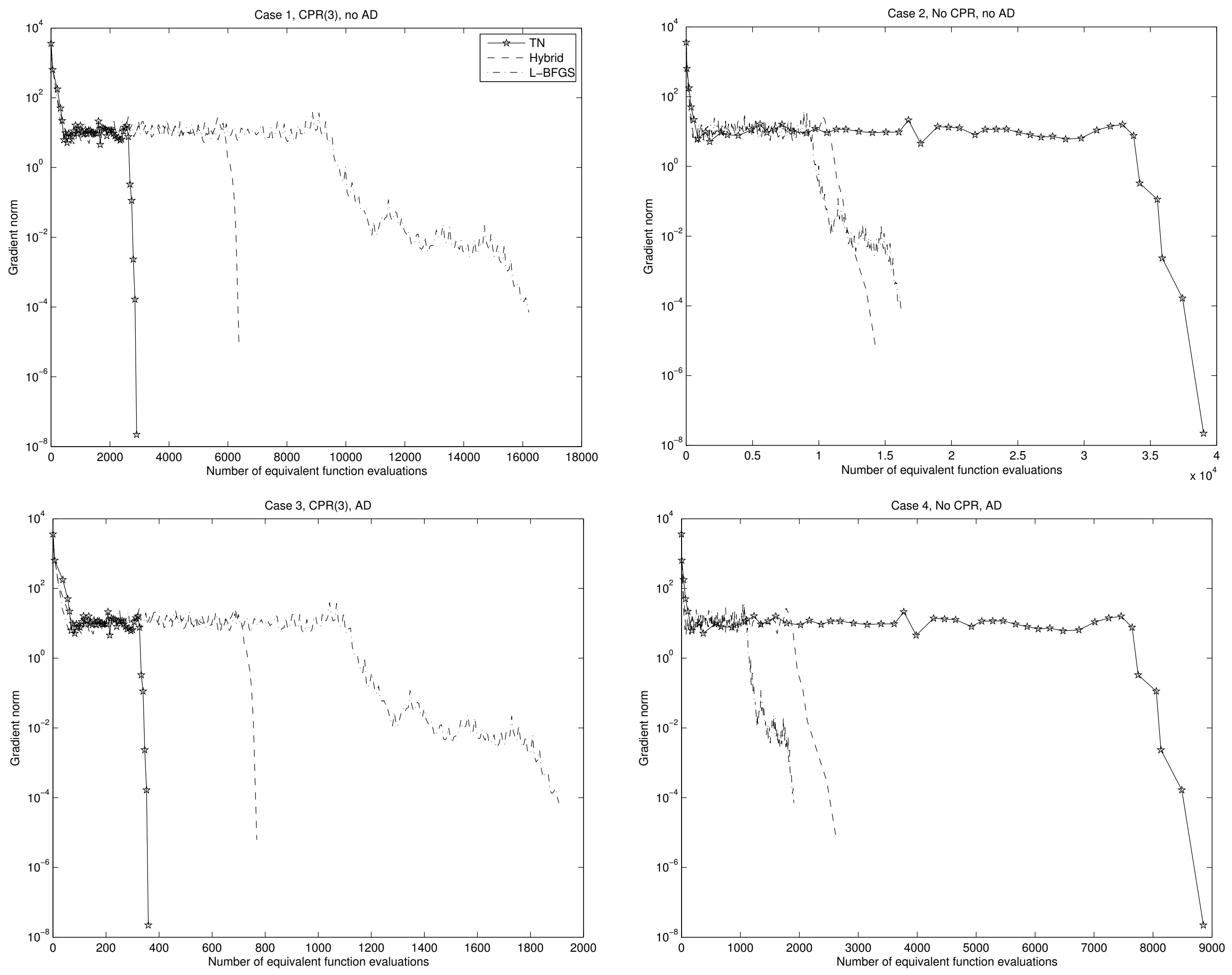

Figure 4.6: Chained Rosenbrock (CHNROSNB), $\rho = 3,\ n = 50$.

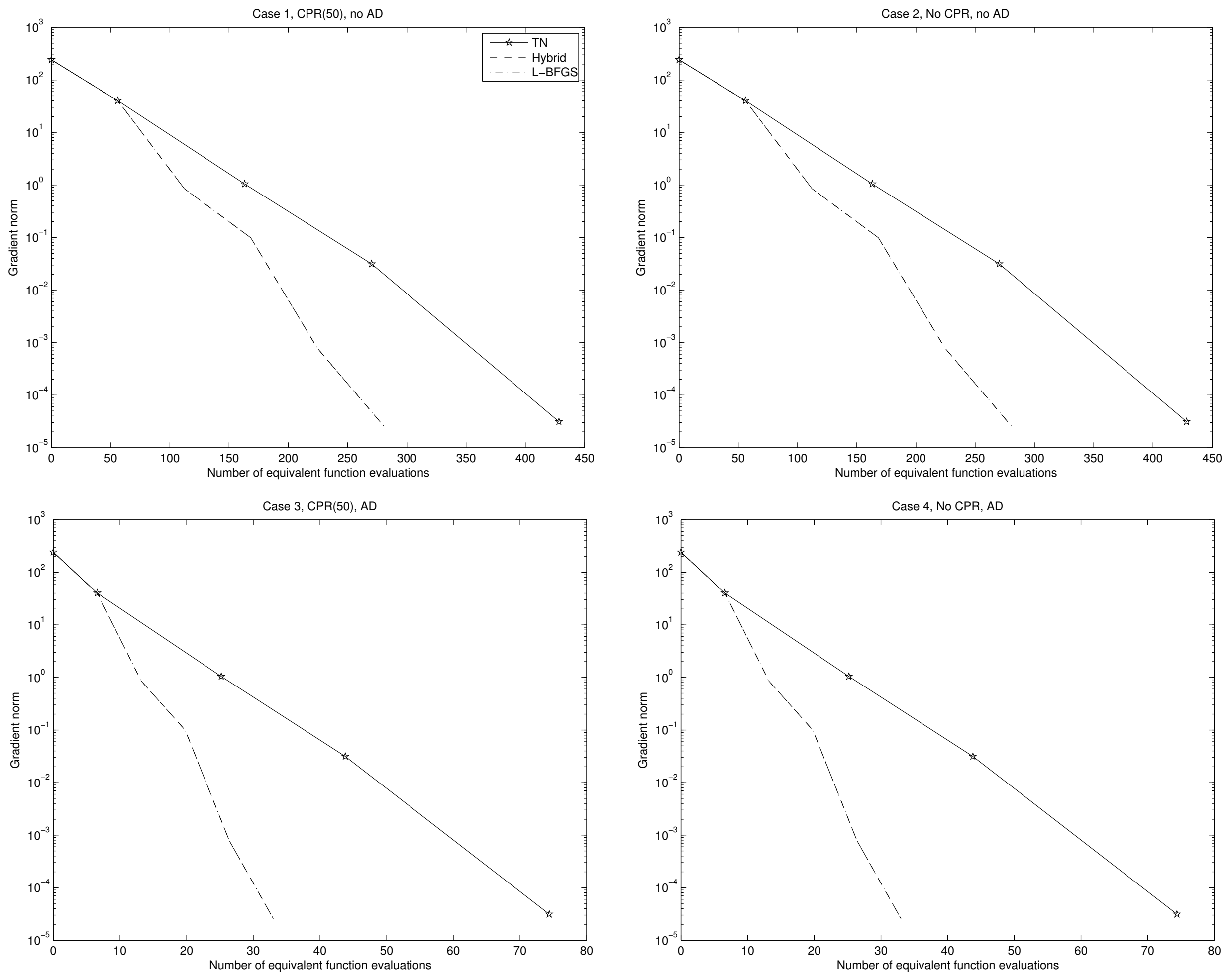

Figure 4.7: HILBERTB, $\rho = 50,\ n = 50$.

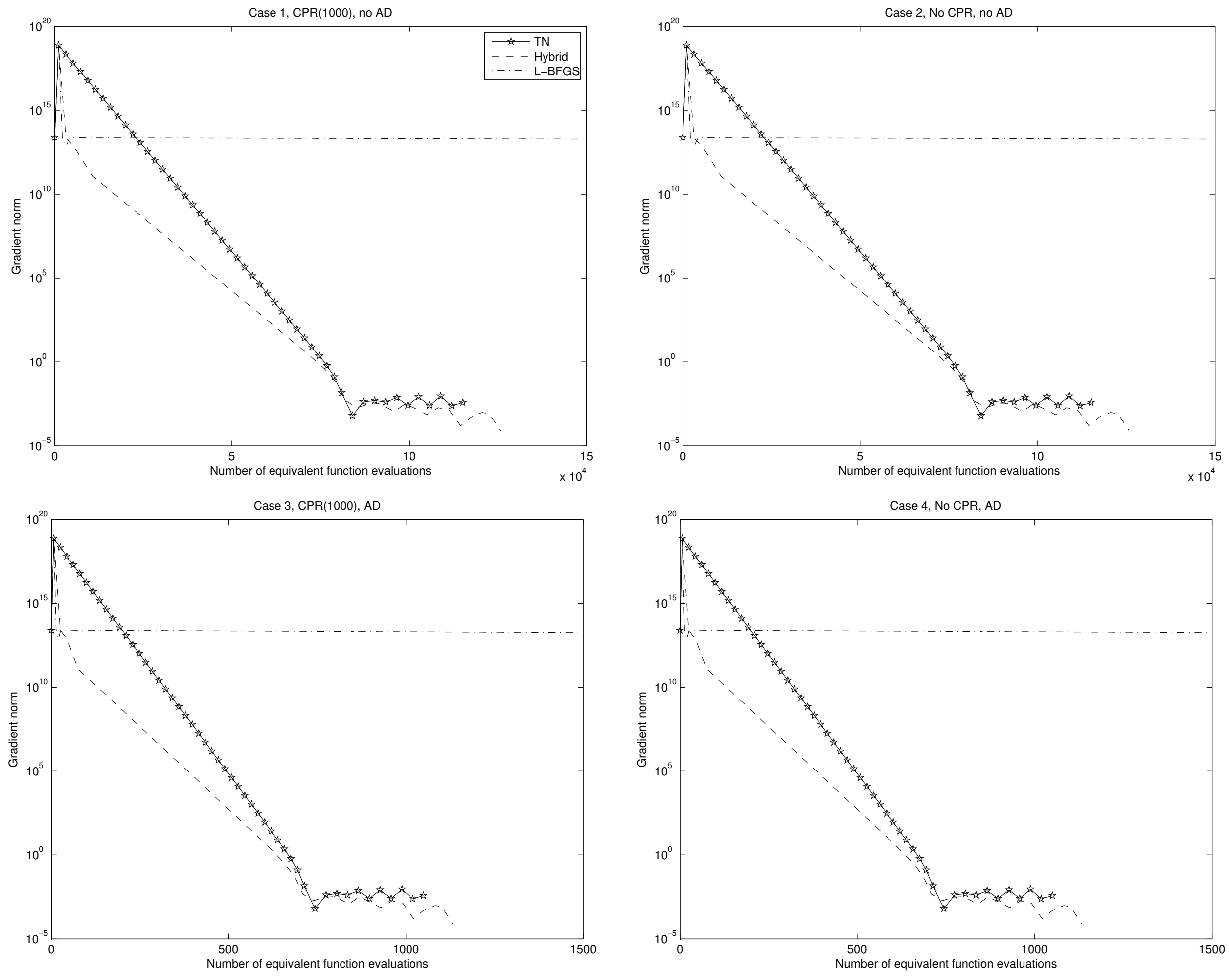

Figure 4.8: PENALTYI, $\rho = 1000,\ n = 1000$.

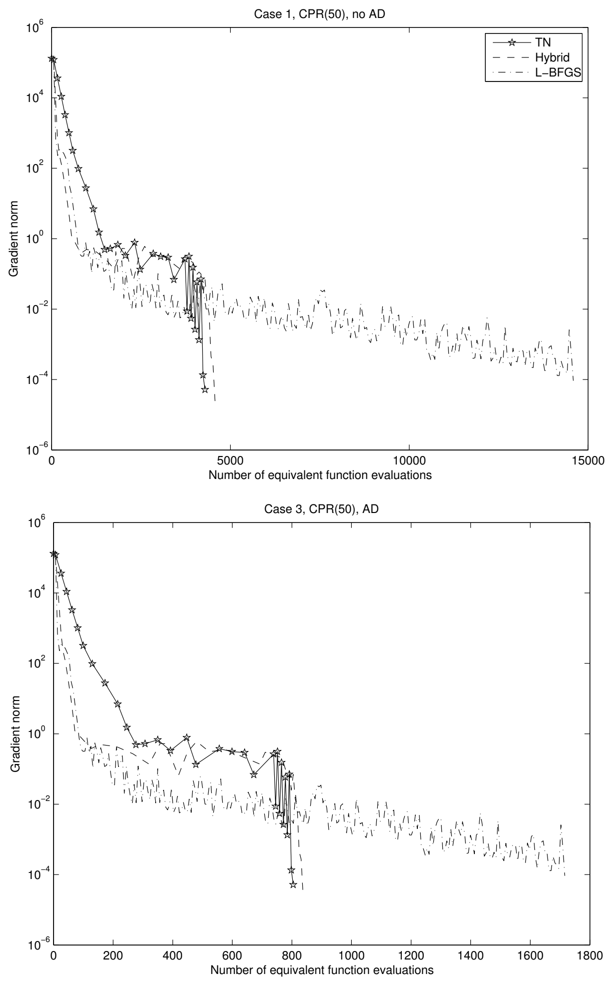

Figure 4.9: PENALTYII, $\rho = 50,\ n = 50$.

The exception to this rule comes when TN is able to exploit CPR techniques effectively at a very early stage by performing many cheap CG iterations, although the tested hybrid method converges very quickly when it starts performing many CG iterations as well.

We feel the our preliminary numerical results are very promising, and that there should exist forcing sequences and possibly dynamic choices of m which should result in effective methods using little memory.

We have not discussed preconditioning of the iterative equation solver (CG in our tests) itself, and this an aspect that should be looked into in the context of our class. Similarly, other choices than CG should also be investigated.

References

[1] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995. 2nd edition 1999.

[2] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

[3] Richard H. Byrd, Jorge Nocedal, and Ciyou Zhu. Towards a discrete Newton method with memory for large-scale optimization. Technical Report OTC 95/01, Optimization Technology Center, 1996.

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

[5] R. S. Dembo and T. Steihaug. A test problem generator for large-scale unconstrained optimization. *ACM Transactions on Mathematical Software*, 11(2):97–102, 1985.

[6] Ron Dembo, Stanley Eisenstat, and Trond Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[7] Ron S. Dembo and Trond Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.

[8] Jean Charles Gilbert and Claude Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, 45:407–435, 1989.

[9] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. Technical Report RAL–TR–2002–009, Computational Science and Engineering Department, Rutherford Appleton Laboratory, 2002.

[10] Lianju Jiang, Richard H. Byrd, Elisabeth Eskow, and Robert B. Schnabel. A preconditioned l-BFGS algorithm with application to molecular energy minimization. Technical Report CU-CS-982-04, Department of Computer Science, University of Colorado, Boulder, Colorado 80309, 2004.

[11] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[12] Ladislav Lukšan and Emilio Spedicato. Variable metric methods for unconstrained optimization and nonlinear least squares. *Journal of Computational and Applied Mathematics*, 124:61–95, 2000.

[13] José Luis Morales and Jorge Nocedal. Enriched methods for large-scale unconstrained optimization. *Computational Optimization and Applications*, 21:143–154, 2002.

[14] Jorge J. Moré. Automatic differentiation tools in optimization software. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation of Algorithms*, 2002.

[15] Jorge J. Moré and David J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307, September 1994.

[16] Stephen G. Nash. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6(3):599–616, 1985.

[17] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

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

[19] Diane P. O'Leary. A discrete Newton algorithm for minimizing a function of many variables. *Mathematical Programming*, 23(1):20–33, 1982.

[20] Dexuan Xie and Tamar Schlick. Efficient implementation of the truncated Newton method for large scale chemistry applications. *SIAM Journal on Optimization*, 10(1):132–154, 1999.