Continuous Steepest Descent Paths for Non-Convex Optimization

by

Salah Beddiaf

A thesis submitted in partial fulfilment of the
requirements of the University of Hertfordshire
for the degree of
Doctor of Philosophy

University of Hertfordshire

March 2015

Acknowledgement

I would like to express my very great appreciation to my supervisors and colleagues, Mike Bartholomew-Biggs and Steven Kane for making this research possible by virtue of their advice, patience, guidance, immense knowledge and general motivational support. Their willingness to give their time so generously has been very much appreciated. I would like to thank Alan Davies, who as a colleague, was always willing to help and give his best suggestions as well as Ralf Napiwotzki for his help with producing some of the graphs. Completing this doctoral work has been a wonderful and often overwhelming experience.
I would also like to thank my family for the constant support and understanding as well as for the many sacrifices they have to endure during these times as without them I would have not got this far.

Abstract

In this thesis, we investigate methods of finding a local minimum for unconstrained problems of non-convex functions with n variables, by following the solution curve of a system of ordinary differential equations.
The motivation for this was the fact that existing methods (e.g. those based on Newton methods with line search) sometimes terminate at a non-stationary point when applied to functions $f(x)$ that do not a have positive-definite Hessian (i.e. $\nabla^2 f \succ 0$) for all x. Even when methods terminate at a stationary point it could be a saddle or maximum rather than a minimum. The only method which makes intuitive sense in non-convex region is the trust region approach where we seek a step which minimises a quadratic model subject to a restriction on the two-norm of the step size. This gives a well-defined search direction but at the expense of a costly evaluation.

The algorithms derived in this thesis are gradient based methods which require systems of equations to be solved at each step but which do not use a line search in the usual sense. Progress along the Continuous Steepest Descent Path (CSDP) is governed both by the decrease in the function value and measures of accuracy of a local quadratic model.

Numerical results on specially constructed test problems and a number of standard test problems from CUTEr [38] show that the approaches we have considered are more promising when compared with routines in the optimization tool box of MATLAB [46], namely the trust region method and the quasi-Newton method. In particular, they perform well in comparison with the, superficially similar, gradient-flow method proposed by Behrman [7].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preliminaries

In this thesis, we are concerned with finding a local solution of the unconstrained optimization problem

$$\underset{x}{\text{minimise}} \;\; f(x), \;\; x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n, \tag{1.1}$$

where $f(x)$ is a single real valued function assumed to be twice continuous differentiable. Problems of this type arise in many practical situations such as Finance, Economics, Commerce, Science, Engineering, Management as well as Mathematics itself.

As is well known, the first order necessary condition for the optimal solution x_* to (1.1) is given by solving the n non-linear system of equations

$$\nabla f(x_*) = 0. \tag{1.2}$$

A number of algorithms have been proposed over the years for solving (1.1) to fulfil this optimality condition. Many methods are appropriate only for certain types of these problems. Thus, it is important to be able to recognise the characteristics of the problem in order to identify an appropriate solution technique. Within each class of problems there are different minimisation methods, varying

in computational requirements, convergence properties, and so on. A discussion on the relative strength and weaknesses of the available techniques within a given class of problems can be found in many of the excellent textbooks ([1],[4], [6], [8], [22], [43], [33], [36], [44]).

Solving the non-linear system of equations (1.2) usually involves generating a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ which converge to the solution x_*. We usually approach this type of optimization problem assuming that x_* exists, is unique and can be located to some desired accuracy, by the chosen method in a finite number of steps. While this is often the case, it is important to realize this ideal situation may fail to hold such as when the function is unbounded below (e.g. $f(x) = x^3$), or even when $f(x)$ is bounded (e.g. $f(x) = e^{-x}$). It may also not have a unique solution as in $f(x) = \sin(x), x \in \mathbb{R}$. In general, it is often only important to find a local minimum. Solving for a global minimum can have considerable difficulties (see Dixon and Szego [29]).

In this thesis we are only interested in local optimization and not global optimization. There are many iterative numerical optimization techniques (mainly gradient methods) which can be applied to (1.1). Most of these methods use an iteration of the form

$$x_{k+1} = x_k + \alpha_k p_k, \tag{1.3}$$

where p_k is a descent search direction (i.e. $p_k^T \nabla f < 0$) and α_k is a step length obtained by a one-dimensional search to ensure the decrease of the function i.e. $f(x_{k+1}) < f(x_k)$.

Sometimes these techniques enter a region where the Hessian ($\nabla^2 f$) is not positive definite and they may fail or exhibit slow convergence when they do. As an example the Newton search direction

$$p = -(\nabla^2 f)^{-1} \nabla f,$$

may point towards a saddle or a local maximum if $\nabla^2 f$ is not positive definite. Quasi-Newton methods, which use a positive definite approximation to the Hes-

sian, cannot perform a standard update to revise their estimate of the Hessian when a step is taken along a direction of negative curvature. In fact, in a non-convex region, none of the iterative methods whose search direction is based on minimising a quadratic model function have much theoretical validity. One method which does make sense in this case is the *trust region method* ([23], [33]).

The strategy, we are going to follow for solving (1.1), is also a gradient method and is related to the trust region method. It follows the *Continuous Steepest Descent Path* (**CSDP**). Basically we associate an ordinary differential equation with (1.1). This approach has already been looked at by a number of authors, ([6], [7], [9], [10], [11], [15], [16], [55]). It has not been as widely used as other methods, such as *Conjugate Gradient, Steepest Descent, Newton and Quasi-Newton*, mainly because of the difficulties inherent in solving a system of *non-linear* ordinary differential equations. However computational speed and size of computer memory have greatly increased since *CSDP* methods were first proposed and their disadvantages may not now be so significant.

1.2 Unconstrained optimization

Given a twice differentiable function f of n variables $x_1, x_2, \cdots, x_n$, we define the partial derivative relative to variable x_i, written as $\dfrac{\partial f}{\partial x_i}$, to be the derivative of f with respect to x_i treating all variables except x_i as constant.
Let x denote the vector $(x_1, x_2, \cdots, x_n)^T$, with this notation the gradient of $f(x)$ can be written as

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n} \right)^T ,$$

and the Hessian as

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} .$$

Given a smooth function, an unconstrained optimization problem is

$$\underset{x}{\text{minimise}} \ f(x) \text{ subject to } x \in \mathbb{R}^n$$

where $f(x) : \mathbb{R}^n \to \mathbb{R}$.

Definition 1.2.1 *$x \in \mathbb{R}^n$ is a local minimum of $f(x)$ if there exists $\epsilon > 0$ such that $f(x) \leq f(y)$ for all y such that $\|x - y\| < \epsilon$.*

Definition 1.2.2 *$x \in \mathbb{R}^n$ is a global minimum of $f(x)$ if $f(x) < f(y)$ for all $y \in \mathbb{R}^n$.*

Definition 1.2.3 *$x \in \mathbb{R}^n$ is a strict local minimum of $f(x)$ if there exists $\epsilon > 0$ such that $f(x) < f(y)$ for all y such that $\|x - y\| < \epsilon$, $y \neq x$.*

Definition 1.2.4 *$x \in \mathbb{R}^n$ is a strict global minimum of $f(x)$ if $f(x) < f(y)$ for all $y \in \mathbb{R}^n$, $y \neq x$.*

We will also recall some useful facts from linear algebra that we refer to in later chapters.
Any real symmetric matrix A is diagonalisable. More precisely, if A is symmetric, then there is an orthogonal matrix R such that $D = R^T AR = R^{-1}AR$ is diagonal.

Definition 1.2.5 *An $n \times n$ matrix A is called*

- *positive definite if $x^T Ax > 0$ for all $x \in \mathbb{R}^n, x \neq 0$,*
- *positive semi-definite if $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$.*

In order to know that a given point x_* of a given function $f(x)$ is a minimum, local or a global, we need the so called *optimality conditions*. They provide tests for optimality, and are the basis for most algorithms.

First order necessity
Suppose that x_* is a local minimum of $f(x)$,

- no perturbations about x_* will result in function decrease,
- first order function approximation can be used to derive necessary conditions (i.e., if x_* minimises $f(x)$, then $\nabla f(x_*) = 0$).

Note that if $\nabla f(x_*) = 0$, then x_* is a stationary point, but not necessarily a minimiser, which means that the first order necessary condition does not distinguish maximum, minimum, or saddle points. To discriminate between them we need more information from the function, e.g. second order derivatives of $f(x)$. With second order derivatives we can build necessary and sufficient condition for a minimum.

Second order sufficiency
Suppose that x_* is a local minimum of $f(x)$,

- no perturbations about x_* will result in a function decrease,
- second order function approximation can be used to derive necessary conditions (i.e., if $\nabla^2 f$ is positive definite at x_*, then x_* minimises $f(x)$).

In unconstrained optimization, we can always solve the problem (1.1) using the *necessary conditions*, i.e. by solving the nonlinear system $\nabla f(x_*) = 0$. One of the drawbacks in using this method however is that we might lose some important information related to $f(x)$. A better approach of course would be to use all the given function information and try to build iterative procedures where we start from a point x_0 and then compute a sequence of points $\{x_k\}$ such that $f(x_{k+1}) \leq f(x_k)$, until convergence.

There are two main prototypes of unconstrained optimization methods that are often used this way, mainly the line search methods and the trust region methods. Both of these types are based on quadratic approximation of $f(x)$, but they differ in the order in which they choose the search direction and step sizes.

1.2.1 Line search methods

A generic algorithm for solving unconstrained minimisation problems (1.1), using this iterative method is as follow:

Algorithm 1.2.1 *(Line Search Algorithm)*
Step 1 *Given an initial estimate* x_0,
Step 2 *determine a direction of search* p_k *along which* f *decreases, i.e.*

$$p_k^T \nabla f(x_k) < 0,$$

Step 3 *find a step size* α_k *such that* $f(x_k + \alpha_k p_k)$ *decreases sufficiently, for example in as in a weak line search sense*

$$f(x_k + \alpha_k p_k) < f(x_k) - \delta_k \text{ for some } \delta_k > 0.$$

Alternatively choose α_k *to give the biggest decrease in* $f(x)$ *by an exact line search (using this type of step can be rather expensive).*

$$f(x_k + \alpha_k p_k) = \min_{\alpha \in \mathbb{R}} f(x_k + \alpha p_k),$$

Step 4 *set* $x_{k+1} = x_k + \alpha_k p_k$,
Step 5 *repeat until convergence*

Different methods correspond to different ways of computing the direction p_k in step (2) and the step size α_k in step (3).
Step (3) includes the one-dimensional minimization problem which may not be solved exactly. In fact most of the modern methods implement the weak (inexact) line-search mainly to ensure steps are neither 'too long' nor 'too short', and try to pick a useful initial step size for fast convergence.

1.2.2 Trust region methods

The trust region method is an alternative to the line search method. In this method we first pick a step size Δ, say, and the choose p_k to reduce a model of $f(x_k + p)$ subject to the restriction $\|p_k\| \leq \Delta$. We then accept $x_{k+1} = x_k + p_k$,

if the decrease predicted by the model compares well with the actual value of $f(x_k + p_k)$. If the existing model produces a poor step however then then 'trust region radius' Δ is reduced and the iteration is repeated.

Consider the quadratic model of $f(x_k + p)$

$$f(x_k + p) \approx Q(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B p, \tag{1.4}$$

where B is a symmetric approximation of the Hessian $\nabla^2 f(x_k)$.
The trust region sub-problem is therefore to

$$\min_{p \in \mathbb{R}^n} Q(p) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T B p, \text{ subject to } \|p\| \leq \Delta_k \tag{1.5}$$

where Δ_k is a suitably chosen radius for the step.

To solve (1.5) is equivalent to finding a vector p which satisfies

$$\|p\| \leq \Delta_k,$$

and there should exist a scalar $\mu \geq 0$ such that

$$(B + \mu I)p = -\nabla f(x_k),$$

$$\mu = 0 \text{ or } \|p\| = \Delta_k,$$

$$B + \mu I \text{ is positive semi-definite },$$

(see Nocedal and Wright, [52]).

We then compute the ratio of the expected and the actual reduction of the function by using

$$r = \frac{f(x_k) - f(x_k + p)}{Q(0) - Q(p)}. \tag{1.6}$$

This ratio is used to decide whether to increase or decrease the trust region radius. If the ratio is close to 1.0, then we can say that there is a good benefit in

increasing the radius for future iterations. If on the other hand this is not the case (i.e. there is poor agreement between the predicted and the actual decrease), then the current step should be rejected and we reduce Δ_k in the next iterations. This technique avoids the difficulty caused by non-positive definite Hessian matrices in line searches.

Algorithm 1.2.2 *(Trust Region Algorithm)*

Step 1 *Given initial point* x_0, $\bar{\Delta}$, $\Delta_0 \in (0, \bar{\Delta})$, $\epsilon \geq 0$, $0 < \eta_1 \leq \eta_2 < 1$ *and* $0 < \gamma_1 < 1 < \gamma_2$, $k := 0$,

Step 2 *if* $\|\nabla f(x)\| \leq \epsilon$, *stop,*

Step 3 *approximately solve the subproblem (1.5) for* p_k

Step 4 *Compute* $f(x_k + p_k)$ *and* r_k *from (1.6).*

$$x_{k+1} = \begin{cases} x_k + p_k, & \text{if } r_k \geq \eta_1, \\ x_k, & \text{otherwise} \end{cases}$$

Step 5 *If* $r_k < \eta_1$, *then* $\Delta_{k+1} \in (0, \gamma_1 \Delta_k]$
If $r_k \in [\eta_1, \eta_2)$, *then* $\Delta_{k+1} \in [\gamma_1 \Delta_k, \Delta_k]$
If $r_k \geq \eta_2$, *then* $\Delta_{k+1} \in [\Delta_k, \ min \ \{\gamma_2 \Delta_k, \bar{\Delta}\}]$

Step 6 *generate* B_{k+1}, *update* Q_k, *set* $k = k + 1$ *and go to Step 2.*

1.2.3 Rate of convergence

Consider a sequence $\{x_k\}$ of estimates to a minimizer x_*. If $\lim_{k \to \infty} x_k = x_*$ then we say that the convergence is

(a) linear if there is a constant $0 < r < 1$ such that

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = r;$$

(b) superlinear if

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} = 0;$$

(c) quadratic if there is a constant $C > 0$ such that

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|^2} = C.$$

These ratios show how fast the iterates converge in the neighbourhood of the minimiser x_*. In general linear convergence is only satisfactory if the constant r is small say $r \leq \frac{1}{4}$. However, in most methods the aim is for a better convergence rate.

1.2.4 Convexity and minimization

In general, optimization has its mathematical foundation from linear algebra and multivariate calculus. However the area of convexity is also very important. Here we only summarise some of the concepts and results. For an in depth study of the properties and proofs of the theorems in this area, see [12].

Definition 1.2.6 *If $x, y \in \mathbb{R}^n$, then points of the form $\lambda x + (1-\lambda)y$ for $\lambda \in [0, 1]$ are called convex combinations of x and y.*

Definition 1.2.7 *A set $S \in \mathbb{R}^n$ is called a convex set if for all $x, y \in S$ and for all $\lambda \in [0, 1]$ it holds that $\lambda x + (1 - \lambda)y \in S$.*

Definition 1.2.8 *Function $f(x) : S \to \mathbb{R}$, where S is a nonempty convex set, is a convex function if*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

for all $x, y \in S$ and for all $\lambda \in [0, 1]$.

Theorem 1.2.1 *Suppose S is a convex set, $f(x) : S \to \mathbb{R}$ is a convex function, and x_* is a local minimum of $f(x)$. Then x_* is a global minimum of $f(x)$ over S.*

Theorem 1.2.2 *Suppose S is a non-empty open convex set, and $f(x) : S \to \mathbb{R}$ is differentiable. Then $f(x)$ is a convex function if and only if $f(x)$ satisfies the following gradient inequality:*

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) \text{ for all } x, y \in S.$$

Theorem 1.2.3 *Suppose S is a non-empty open convex set, and $f(x) : S \to \mathbb{R}$ is twice differentiable. Let $\nabla^2 f(x)$ denote the Hessian of $f(x)$. Then $f(x)$ is convex if and only if $\nabla^2 f(x)$ is positive semi-definite for all $x \in S$.*

Theorem 1.2.4 *Suppose $f(x) : S \to \mathbb{R}$ is convex and differentiable on S, then $x \in S$ is a global minimum, if and only if $\nabla f(x) = 0$.*

In general, convex functions are relatively easy to minimize because they can be modelled reasonably well by a positive definite quadratic. But many real-life minimisations involve functions which are not convex for all x. They may have a minimum in a convex region, but a search for this minimum from a bad starting guess may have to pass through a region of non-convexity. It is important to be able to traverse such regions efficiently and the main focus of this thesis is on algorithms which can do this.

1.3 Thesis outline

The structure of this thesis is as follows. In the next Chapter we introduce the idea of Continuous Steepest Descent Path (CSDP) methods for unconstrained optimization and give a brief literature review. In Chapter three we look at the approximation of the CSDP and algorithms for solving problems of the form (1.1). Numerical results are compared with the well known methods for solving such problems such as trust regions and quasi-Newton methods implemented in MATLAB. In the fourth Chapter we look at further algorithmic investigations mainly by varying some of the parameters involved in the curvilinear search and considering some selected test problems as well as results from the CUTEr Test problems.

In Chapter 5 we describe the use of updates to replace the actual Hessian. Three standard methods are used (BFGS, DFP and SR1) but without the need for preserving positive definiteness. Two diagonal approximation methods are also considered. All these methods use the same curvilinear search as in Chapter 4. There will be a full set of CUTEr results for all these algorithms.

In Chapters 3 - 5 all our methods have treated convex regions the same as non-convex ones. Chapter 6 considers a hybrid approach which follows a CSDP in non-convex regions but reverts to a conventional method near a solution when the function is convex. We test these algorithms on our own specially constructed test problems as well as the CUTEr ones.

Chapter 7 proposes a curvilinear search method that offers substantial reduction in overhead cost per iteration. This is based on solving the steepest descent equation by an explicit second order Runge Kutta (RK) approach and this lets us avoid the matrix computations needed by the algorithms in Chapters 3 - 6. We report some results on specially constructed test problems. The conclusion will be given in the last Chapter.

Chapter 2

The Continuous Steepest Descent Path

2.1 Introduction to CSDP

Consider the unconstrained optimization problem (1.1). We suppose it involves a nonlinear twice continuously differentiable objective function $f(x)$ with the gradient vector $g(x) = \nabla f(x)$ and the Hessian matrix denoted by $G(x) = \nabla^2 f$ with G^{-1} representing the inverse of the Hessian and H will denote an updated approximation to its inverse.

As mentioned in Chapter 1 a basic general algorithm to find a local minimum using iterative techniques for solving unconstrained minimisation problem is as follows.

Algorithm 2.1.1 *(Basic Algorithm)*
Step 1 *A starting point x_0 is chosen*
Step 2 *For $k = 0, 1, 2, 3...$ until convergence.*
(a) Calculate a search direction p_k
(b) Determine a step length α to ensure $f(x_k + \alpha p_k) < f(x_k)$
(c) Set $x_{k+1} = x_k + \alpha p_k$
Step 3 *Test for convergence.*

Different methods corresponds to different ways of choosing p_k in step (2.a), based

on information available in the problem. Step (2.b) is the line search subproblem. The three most used techniques based on the line search methods are the *Steepest Descent*,

$$x_{k+1} = x_k - \alpha_k g(x_k), \tag{2.1}$$

Newton,

$$x_{k+1} = x_k - \alpha_k G^{-1}(x_k) g(x_k), \tag{2.2}$$

and Quasi-Newton

$$x_{k+1} = x_k - \alpha_k H(x_k) g(x_k). \tag{2.3}$$

where H denotes a positive definite approximation of $G^{-1}(x)$ which is updated at the end of each iteration. From a given point x_0, and the scalar step length α_k, (normally chosen to ensure $f(x_{k+1}) < f(x_k)$) these iterative schemes generate a sequence of points (x_{k+1}) and search directions $(x_{k+1} - x_k)$, constructed to converge to the true solution x_*.

The *Continuous Steepest Descent Path* (CSDP) which is related to (2.1) can be defined as the solution to the initial value problem

$$\frac{dx}{dt} = -g(x(t)), \quad x(0) = x_0. \tag{2.4}$$

This could also be derived by rearranging (2.1) as

$$\frac{x_{k+1} - x_k}{\alpha_k} = -g(x_k).$$

In the limit as $\alpha_k \to 0$, this reduces to (2.4) where α_k is the time step in the forward Euler discretisation. The idea of finding a local minimum by following the solution path of (2.1) was first proposed by Arrow, Hurwitz & Uzawa [3]. These authors do not discuss practical implementations of the idea; but a good many authors have subsequently looked at optimization methods based on (2.4) and a review of some of these techniques appears below as section 2.5.

Similarly continuous versions to (2.2) and (2.3) can respectively be defined by

$$\frac{dx}{dt} = -G^{-1}(x(t))g(x(t)) \tag{2.5}$$

$$\frac{dx}{dt} = -H(x(t))g(x(t)) \tag{2.6}$$

with initial condition $x(t) = x_0$ specified at $t = 0$.

The solution curve $x(t)$ for $t > 0$ such that $\lim_{t\to\infty} x(t) = x_*$ to any of the above systems of ordinary differential equations, if it exists, can be followed to give x_* as a stationary point of $f(x)$ ([7], [15], [11], [55]). Since this point is reached by a path of continuous descent then x_* must be a local minimum or a saddle point, depending on whether or not $G(x_*)$ is positive definite.

The methods that we will be considering in this thesis are based on equation (2.4). From $x(0) = x_0$, let $p(t)$ be a curve that is an *approximation* to the integral curve $x(t)$ of the vector field from this point using the solution to (2.4) with $p(0) = 0$. We then search along this curve $p(t)$ for $t > 0$, continuing to increase t as long as the objective function is being *sufficiently reduced* and $p(t)$ is remaining *sufficiently close* to $x(t)$. We shall discuss these criteria in more detail later on in Chapter 3. If the search along $p(t)$ is terminated at a point x_1 (e.g. because $p(t)$ seems too far from $x(t)$) then another search path $p(t)$ is constructed as an approximate solution of problem (2.4) with initial condition changed to $x(0) = x_1$. A search along $p(t)$ will then yield a new point x_2; and this process is repeated until a point is found that satisfies certain convergence criteria such as $\|g(x_*)\| < \epsilon$, where ϵ is some specified tolerance.

2.2 Gradient flow method

A *gradient flow method* of the kind just described has been proposed by Behrman [7] for (2.4) which is based upon the solution of a system of n linear ordinary differential equations, where n denotes the number of independent variables in the objective function.

The k-th iteration of Behrman's algorithm uses the Taylor series approximation of the vector field $-g$ about $x = x_k$ and then uses the integral curves of the approximating vector fields. The linearised CSDP can then be written as

$$\frac{dx}{dt} = -g(x_k) - G(x_k)(x - x_k) \tag{2.7}$$

where x_k denotes the starting point of the k^{th} iteration. Equation (2.7) has an analytical solution, passing through x_k given by

$$x(t) = x_k + p_k(t) \text{ where } p_k(t) = -R\Lambda R^T g_k \tag{2.8}$$

and $R = R(x_k)$ is the matrix whose columns are the normalised eigenvectors of $G(x_k)$ while Λ is a diagonal matrix whose elements are derived from the eigenvalues $\lambda_1, ..., \lambda_n$ via

$$\Lambda_{ii} = \begin{cases} \frac{1}{\lambda_i}\left(e^{-\lambda_i t} - 1\right) & \text{for} \quad \lambda_i \neq 0 \\ t & \text{for} \quad \lambda_i = 0 \end{cases} \tag{2.9}$$

From any given point, Behrman's algorithm calculates a curve that is initially tangential to the negative gradient. Hence $f(x_k + p_k(t))$ is initially decreasing as t increases. A new point x_{k+1} is found along $x_k + p_k(t)$ such that $f(x_k + p_k(t)) < f(x_k)$ (and also certain other criteria are met) and the process is repeated until convergence.

2.3 Solving systems of first order ODEs

We shall in this section focus our attention exclusively on solving the system of first order linear differential equations of the form

$$x'(t) = Ax(t) + b.$$

where x' denotes $\dfrac{dx}{dt}$ and x and b are $(n \times 1)$ vectors and A is an $(n \times n)$ symmetrical matrix.

2.3.1 The Homogeneous system

If $b = 0$, then $x'(t) = Ax(t)$ is homogeneous.

Theorem 2.3.1 *Let $x' = Ax$ be a homogeneous linear first-order system. If $x = ve^{\lambda t}$ is a solution to this system (where $v = [v_1, v_2, ..., v_n]^T$), then λ is an eigenvalue of A and v is the corresponding eigenvector.*

Theorem 2.3.2 *If A is a real $n \times n$ matrix with n distinct eigenvalues, $\lambda_1, \lambda_2, ..., \lambda_n$ and associated eigenvectors, $v_1, v_2, ..., v_n$, then*

$$x(t) = \sum_{i=1}^{n} c_i v_i e^{\lambda_i t}$$

is also a solution to the homogeneous system $x' = Ax$, where $c_1, ..., c_n$ are arbitrary, possibly complex, constants.

The proofs of theorems 2.3.1 and 2.3.2 can be found in any standard textbook see [31].

If the eigenvalues are not distinct, the solution to this differential equation might not be easy to get, but nonetheless, as repeated roots are not robust, or *"structurally unstable"* (i.e. do not survive small changes in the coefficients of A), then these can be generally ignored for practical purposes [51].

Recall that a matrix A is *"diagonalizable"* if there is a matrix, R, such that $R^{-1}AR$ is a diagonal matrix.

It is also worth noting that Taylor's expansion of the function $f(t) = e^{at}$ around $t = 0$ is

$$f(t) = e^{at} = 1 + at/1! + a^2t^2/2! + a^3t^3/3! + ...$$

This is fundamental to the proofs of the following theorems, see [31].

Theorem 2.3.3 *An $n \times n$ matrix is diagonalizable if and only if it has n independent eigenvectors.*

Theorem 2.3.4 *The solution of* $x'(t) = Ax(t)$, $x(0) = x_0$ *is* $x(t) = e^{At}x_0$.

Proof: Taylor's expansion of $x(t)$ around $t = 0$ yields:

$$x(t) = x(0) + x'(0)t/1! + x'(0)t^2/2! + x'''(0)t^3/3! + ...$$

As $x'(t) = Ax(t)$, then $x''(t) = Ax'(t) = AAx(t) = A^2x(t)$.
Similarly, $x'''(t) = A^3x(t)$ and so on.
Thus, at $t = 0$, we have $x'(0) = Ax(0) = Ax_0$,
$x''(0) = A^2x(0) = A^2x_0$,
$x'''(0) = A^3x(0) = A^3x_0$, etc.
Using the initial condition $x(0) = x_0$, and replacing these in the Taylor's expansion

$$x(t) = x_0 + Ax_0t/1! + A^2x_0t^2/2! + A^3x_0t^3/3! + ...$$

factorising out by x_0

$$x(t) = [I + At/1! + A^2t^2/2! + A^3t^3/3! + ...]x_0$$

where I is the identity matrix. But, we know that

$$e^{At} = I + At/1! + A^2t^2/2! + A^3t^3/3! + ...$$

hence we get the required answer $x(t) = e^{At}x_0$. $\Diamond$

Computing the exponential of a matrix is an expensive operation and plays a key role in solving the differential equation. Moler and Van Loan carefully reviewed up to nineteen different numerical algorithms for computing the exponential of a matrix in their classic paper [48]. They looked at all methods that appear to be practical and assessed their effectiveness.

The conclusion reached was that all of the methods have some strengths as well as weaknesses, leaving the choice of a particular method to depend on circumstances - none was clearly and unfailingly superior to all the others.

The method we have chosen to use is based on a matrix factorisation which includes eigenvalue eigenvector calculation. This is shown in Theorem 2.3.5 and works particularly well for symmetric matrices as the eigenvectors will be orthogonal. Some difficulties, however, could occur in the case when the matrix does not have a complete set of linearly independent eigenvectors and is thus defective. If the matrix is close to being singular, the calculation of its inverse is numerically unstable. Such possible difficulties need not concern us because the main ideas in this thesis involves numerical rather than analytical solutions to the differential equation which defines the CSDP and hence do not involve the calculations of the exponentials of the matrices.

Theorem 2.3.5 *The solution of* $x'(t) = Ax(t)$, $x(0) = x_0$, *where* A *is diagonalizable, is*

$$x(t) = e^{At}x_0 = Re^{\Lambda t}R^{-1}x_0$$

where $R = [v_1, v_2, ..., v_n]$ *is the modal matrix whose columns are eigenvectors of* A *and* Λ *is a diagonal matrix whose diagonal elements are distinct eigenvalues of* A.

Proof: Distinct eigenvalues ensure linearly independent eigenvectors and hence non-singularity of R and, by our previous theorem, the diagonalizability of A.
Thus, $R^{-1}AR = \Lambda$ or $A = R\Lambda R^{-1}$.
and $A^2 = AA = (R\Lambda R^{-1})(R\Lambda R^{-1}) = R\Lambda I\Lambda R^{-1} = R\Lambda^2 R^{-1}$.
Similarly, $A^3 = R\Lambda^3 R^{-1}$ and so on.
Now, recall that

$$e^{At} = I + At/1! + A^2t^2/2! + A^3t^3/3! + ...$$

By substituting for A, A^2, etc. and recalling that $I = RR^{-1}$, then

$$e^{At} = RR^{-1} + (R\Lambda R^{-1})t/1! + (R\Lambda^2 R^{-1})t^2/2! + (R\Lambda^3 R^{-1})t^3/3! + ...$$

by factorising out R to the left and R^{-1} to the right

$$e^{At} = R\left[I + \Lambda t/1! + \Lambda^2 t^2/2! + \Lambda^3 t^3/3! + ...\right]R^{-1}.$$

By definition

$$e^{\Lambda t} = I + \Lambda t/1! + \Lambda^2 t^2/2! + \Lambda^3 t^3/3! + ...$$

Thus this reduces to

$$e^{At} = Re^{\Lambda t}R^{-1}$$

hence

$$x(t) = e^{At}x_0 = Re^{\Lambda t}R^{-1}x_0$$

as required. $\Diamond$

2.3.2 The Non-homogeneous system

Let us now turn to the non-homogeneous system of linear first order differential equations. Consider the system

$$x'(t) = Ax(t) + b$$

where $b \neq 0$ and is not a function of t.

Theorem 2.3.6 *The solution to $x' = Ax + b$ with initial condition $x(0) = x_0$ is*

$$x(t) = e^{At}k - A^{-1}b$$

where $k = x_0 + A^{-1}b$ or, if A is diagonalizable,

$$x(t) = Re^{\Lambda t}R^{-1}k - A^{-1}b.$$

Proof: Let $y = x + A^{-1}b$, then, as b is independent of t, taking the derivative gives, $y' = x'$. Thus by substitution we get

$$y' = Ax + b = Ax + AA^{-1}b = A(x + A^{-1}b) = Ay,$$

i.e. we obtain a homogeneous system $y' = Ay$.
We know the solution to this homogeneous system is

$$y(t) = e^{At}y_0 = Re^{\Lambda t}R^{-1}y_0.$$

Using $y = e^{At}y_0$ implies

$$x(t) + A^{-1}b = e^{At}[x(0) + A^{-1}b]$$

or simply

$$x(t) = e^{At}\left[x(0) + A^{-1}b\right] - A^{-1}b$$

and so by the definition of k,

$$x(t) = e^{At}k - A^{-1}b.$$

Using $y(t) = Re^{\Lambda t}R^{-1}y_0$ implies

$$x(t) + A^{-1}b = Re^{\Lambda t}R^{-1}\left[x(0) + A^{-1}b\right]$$

or

$$x(t) = Re^{\Lambda t}R^{-1}\left[x(0) + A^{-1}b\right] - A^{-1}b,$$

or, once again, by definition of k,

$$x(t) = Re^{\Lambda t}R^{-1}k - A^{-1}b.\diamondsuit$$

2.4 The exact integral curve

We know the solution to

$$x' = Ax + b \tag{2.10}$$

is

$$x(t) = Re^{\Lambda t}R^{-1}\left[x(0) + A^{-1}b\right] - A^{-1}b, \tag{2.11}$$

where $x = x_k$ at $t = t_0$ and $A = R\Lambda R^{-1} \Rightarrow A^{-1} = R\Lambda^{-1}R^{-1}$.

So now consider the equation (2.7), restated as

$$x' = -g(x_k) - G(x_k)(x - x_k). \tag{2.12}$$

Comparing equations (2.10) and (2.12) and putting $x(0) = x_k$ and using
$A = G(x_k) = R(-\Lambda)R^{-1}$, (where the diagonal of Λ are the eigenvalues of G and the columns of R are its associate eigenvectors)
and $b = G(x_k)x_k - g(x_k) = R\Lambda R^{-1}x_k - g(x_k)$

we get the solution to (2.12) as

$$\begin{aligned} x(t) &= Re^{-\Lambda t}R^{-1}\left[x_k + R(-\Lambda)^{-1}R^{-1}\left[R\Lambda R^{-1}x_k - g(x_k)\right]\right] - \\ &\quad R(-\Lambda)^{-1}R^{-1}\left[R\Lambda R^{-1}x_k - g(x_k)\right] \\ &= Re^{-\Lambda t}R^{-1}\left[x_k + (R(-\Lambda)^{-1}R^{-1})(R\Lambda R^{-1})x_k - R(-\Lambda)^{-1}R^{-1}g(x_k)\right] - \\ &\quad R(-\Lambda)^{-1}R^{-1}R\Lambda R^{-1}x_k + R(-\Lambda)^{-1}R^{-1}g(x_k) \\ &= Re^{-\Lambda t}R^{-1}\left[x_k - x_k + R(\Lambda)^{-1}R^{-1}g(x_k)\right] + x_k - R(\Lambda)^{-1}R^{-1}g(x_k) \\ &= Re^{-\Lambda t}R^{-1}R(\Lambda)^{-1}R^{-1}g(x_k) + x_k - R(\Lambda)^{-1}R^{-1}g(x_k). \end{aligned}$$

Thus, we obtain

$$x(t) = x_k + R\left(e^{-\Lambda t} - I\right)(\Lambda)^{-1}R^{-1}g(x_k) \tag{2.13}$$

This equation is the basic equation for Behrman's method [7] and it can also be written as

$$x(t) = x_k + \left(e^{-t\,G} - I\right)G^{-1}g(x_k). \tag{2.14}$$

We now write $g_k = g(x_k)$ and $G_k = G(x_k)$ and we note that if t_* corresponds to a minimum of $f(x)$ along $p(x_k, t) = \left(e^{-t\,G_k} - I\right)G_k^{-1}g_k$, it must be a root of $df/dt = 0$. But since, by (2.14),

$$\frac{df}{dt} = g^T(x(t))\frac{dx(t)}{dt} = -g^T(x(t))e^{-tG_k}g_k$$

we have

$$g^T(x(t_*))e^{-t_*G_k}g_k = 0 \quad \text{or} \quad g_{k+1}^T e^{-t_*G_k}g_k = 0, \tag{2.15}$$

i.e. the tangent vector to the curve $p(x_k, t)$ at $t = t_*$, $e^{-t_*\,G_k}g_k$ and g_{k+1} are orthogonal.

Consider the case where $f(x)$ is quadratic and of the form

$$f(x) = a + b^T x + \frac{1}{2} x^T Q x,$$

with Q positive definite, having a unique minimum at

$$x_* = -Q^{-1} b.$$

Then $g(x) = Qx + b$, and so at any point $x(t)$ given by (2.14)

$$\left[Q\left[x_k + \left(e^{-t_* Q} - I\right) Q^{-1}(Qx_k + b)\right] + b\right]^T e^{-t_* Q}(Qx_k + b) = 0,$$

or

$$[e^{-t_* Q}(Qx_k + b)]^T [e^{-t_* Q}(Qx_k + b)] = 0.$$

This equation is satisfied when $Qx_k + b = 0$, i.e. $x_k = -Q^{-1}b$ or when $e^{-t_* Q} = 0$ which is true $t_* \to +\infty$.

From the exact solution (2.14) we see that, when $t_* \to +\infty$ and Hessian $G = Q$ is positive definite, then

$$x_{k+1} = x_k - Q^{-1} g_k = x_k - Q^{-1}(Qx_k + b) = -Q^{-1} b = x_*.$$

Which means that for quadratic functions we can approach the solution to arbitrarily high accuracy if we chose t large enough.

Theorem 2.4.1 *Let $x(t)$ be the solution to (2.7). For a fixed $t_0 \geq 0$ if $g(x(t)) \neq 0$ for all $t > t_0$, then $f(x(t))$ is strictly decreasing with respect to t, for all $t > t_0$.*

Proof (as given in [7])
We know

$$\begin{aligned} \frac{df(x(t))}{dt} &= g(x(t))^T \frac{dx(t)}{dt} && (2.16) \\ &= -g(x(t))^T g(x(t)) && (2.17) \\ &= -\|g(x(t))\|_2^2. && (2.18) \end{aligned}$$

Since $g(x(t)) \neq 0$ when $t > t_0$, it follows that $\frac{df(x(t)}{dt} < 0$.
i.e. $f(x(t))$ is strictly decreasing with respect to $t > t_0$. $\Diamond$

From (2.18) we see that $f(x(t))$ is strictly decreasing in Euclidean norm as t increases along any solution of the Ordinary Differential Equations (ODE), unless of course $g(x(t)) = 0$. Hence solving for a large t might be considered as finding the local minimum of f.

It can be seen from (2.7) that if the only information given about f at x_k is its gradient $g(x_k)$, then we can use $p_k(t) = -tg(x_k)$ as the solution to (2.7). In this case using the ray $x_k + p_k(t)$ to search for a new point that satisfies certain search criteria and repeating the process is just the *steepest descent* method.

For a quadratic objective function, the curve that Behrman's algorithm calculates is the exact integral curve, and for positive-definite quadratic the algorithm finds the minimiser in one step. This is identical to the Newton step.

If at x_k the gradient $g(x_k)$ and the Hessian $G(x_k)$ for a general function $f(x)$ are known, we can use the the first order approximation of the gradient so that

$$g(x) = g(x_k) + G(x_k)(x - x_k)$$

and then use the readily available solution using (2.8). This is done by using the quadratic approximation of the function about x_k. Starting at point x_0, we compute $p_0(t)$ and find x_1 along the curve $x_0 + p_0(t)$. The search is continued in this way to find other points x_k and paths $p_k(t)$. By joining these curves $p_k(t)$ together and pasting parts of them to form a piecewise-smooth curve $p(t)$ that connects the initial point x_0 with a critical point x_* of f as shown in Figure (2.1). One important point we could make here is that the minimum could be reached in one step for any function if the actual solution curves to the differential equations were available. (The dotted curves in Figure 2.1 represent the CSDP that would be obtained by solving (2.4) exactly.)

On each iteration, the algorithm's search curve is initially tangent to the negative gradient, and if the Hessian at the initial point of the search curve is positive definite, then the search curve will be bounded and the step to the end of the curve is a Newton step. Hence, we can obtain quadratic convergence near the solution. The objective function value of an indefinite quadratic is unbounded below. We shall see in the next section how to deal with this case.

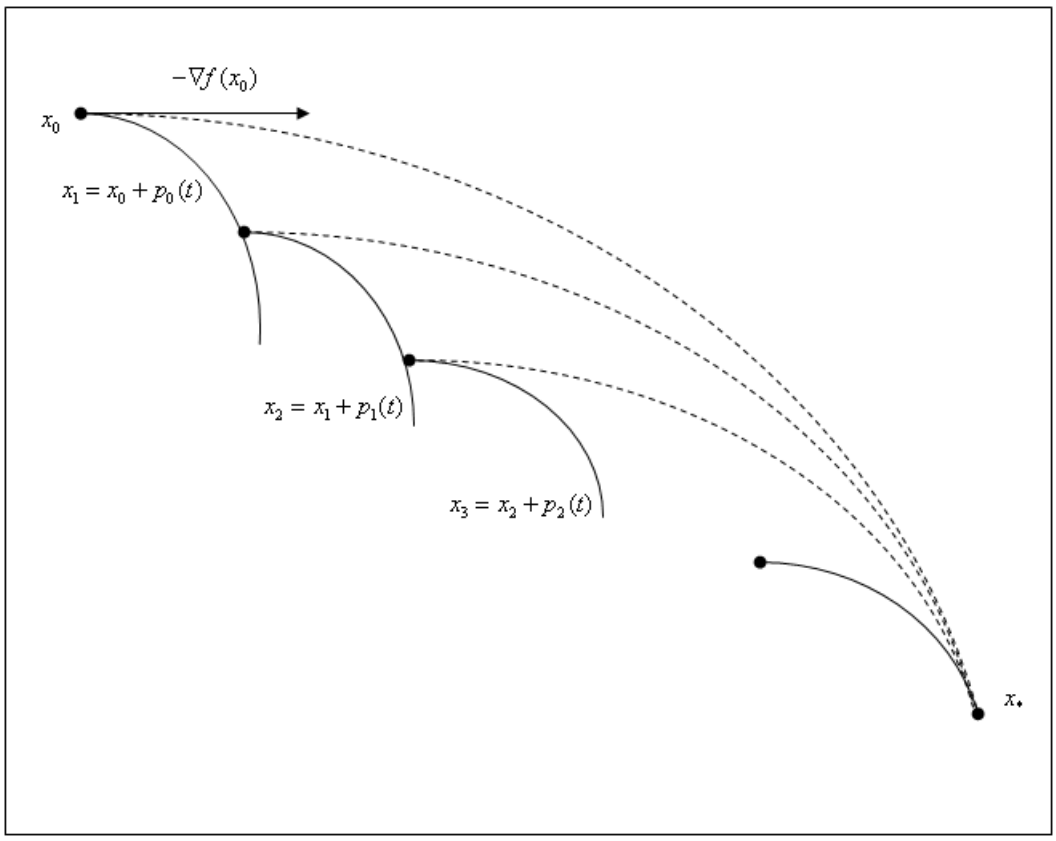

Figure 2.1: Following approximate CSDPs

An important point to make about the piecewise descent path illustrated in Figure 2.1 is that it exists and is computable even if $f(x)$ is not convex in the region around one of the iterates x_k. Hence the CSDP provides an effective search direction until the search for a minimum enters a convex region within which a solution x_* is to be found. In this respect therefore, CSDP provides a powerful alternative to minimization methods such as Newton, quasi-Newton and conjugate gradient algorithms which are all based on approximating $f(x)$ by a

convex quadratic model. The steepest descent method implemented as a search direction/line search method is also known to be slow (classic example given in [4]) because it ignores curvature. The CSDP approach allows it to maintain descent until the search re-enters a convex region in which a minimum can be found.

We have looked at one particular CSDP method which uses the analytical solution of the gradient equation. Many other approaches use numerical approximations to the solution of $\frac{dx}{dt} = -g(x)$. In the next section we briefly review some of the proposals that have been made.

2.5 Literature review

We are concerned in this thesis with methods for solving the unconstrained minimisation problem

$$\underset{x}{\text{minimise}} \;\; f(x), \;\; x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n,$$

by following the solution curve of ordinary differential equations. We have introduced one possible ODE-based minimisation algorithm by describing the ideas of Behrman [7], who described gradient flow algorithms by solving systems of linear first order differential equations. He also showed that his method could scale large problems by projecting the linear ODE onto a space spanned by a small number of Lanczos vectors of the Hessian matrix H. Del Gatto further improved on this in his PhD thesis [35] by proving convergence in the case where the ODE is projected onto two dimensional subspaces. These are just two methods which have been published in the area of our research. We now summarise some further proposals.

Equation (2.12) has been discussed extensively by Botsaris and Jacobson [11], and also by Botsaris [10], [9]. Brown and Bartholomew-Biggs [16] gave a brief review for minimising functions by following solution curve of system of differential equations and have shown by using variety of algorithms, that the ODE approach can be competitive with traditional or more conventional techniques.

Vial and Zang [55] have suggested a gradient path algorithm, requiring the first derivative of the objective function and using a rank-one correction which maintains and updates approximate second order information. This results in a quasi-Newton type algorithm for minimising a differentiable function based on the explicit solution (2.8) of quadratic function f. Vial and Zang also used a formula for the step size with an Armijo-type rule to give a sufficient decrease in the function when using inexact minimisation along the integral curve. Like many authors however, they ignore the case of non-convexity in the problem and the process of updating the second order information is quite involved. Zang, [62] used a traditional positive definite Hessian approximation and a search along the integral curve which is in two phases. In the first stage the function being minimised corresponds to some local quadratic approximation. This method initially attempts a curvilinear search with the step size being reduced until a suitable reduction is obtained. The second phase consists of a conventional search along a straight line. Again the case of non-convexity in the problem is ignored.

Shi and Shen, [59] used a descent algorithm with curve search rule for unconstrained minimization problems, so that at each iteration the search direction and the step size are determined simultaneously using a method that resembles Wolfe's line search rule [60],[61]. They also used a feature similar to the conjugate gradient method which avoids the use of matrices. Luo et al. [45] have considered a combination of the Trust Region technique and a second order Rosenbrock method to select the time step for the IVP. These are a special class of Runge-Kutta methods and very efficient for stiff problems. This technique is different from the local error approach. In this approach the first order Rosenbrock step calculation is derived from the implicit Euler method (which we consider in section (3.1)). The two-stage Rosenbrock solves the following system

$$\begin{aligned} (I + h_k\, r\; G_k)\, d_k &= -h_k\, g(x_k) \\ (I + h_k\, r\; G_k) p_k &= -h_k\, g(x_k + a\, d_k) \\ x_{k+1} &= x_k + b_1 d_k + b_2 p_k \end{aligned} \tag{2.19}$$

where $r,\ a,\ b_1,\ b_2$ are constants, with $r = 1-\sqrt{2}/2,\ a = (\sqrt{2}-1)/2,\ b_1 = 0,\ b_2 = 1$

and the time step is adjusted via a trust region technique. This method works with the exact Hessian and recognizes that it may be non positive definite. But it only deals with this by making h very small. There seems to be no search procedure in this algorithm. Instead we just accept or reject a new point and adjust h for the next iteration. In the algorithm the search step p_k has to satisfy some given conditions similar to the Wolfe condition.

Ou, Zhou and Lin, [53] combined the trust region technique, with ODE-based method, using the implicit Euler approach. In this method, a system of linear equations is solved to obtain a trial step. When this is not accepted, the method generates an iterative point whose step length is defined by a formula. Under some standard assumptions, it is proven that the algorithm is globally convergent and locally superlinear convergent. Ou, [54] also proposed a method for solving a class of non-linear equations, which combines the Trust Region technique and ODE-based methods, and claimed numerical examples show that this method is efficient and reliable.

In his PhD thesis Mohr, [47] combined low-order implicit Runge-Kutta methods for gradient systems and quasi-Newton type updates of the Jacobian matrix were considered to find local minimisers. These hybrid algorithms numerically approximate the gradient flow, but the exact Jacobian matrix is not used to solve the non-linear system at each step. Instead, a quasi-Newton update approximates the Hessian matrix and matrix-vector multiplications are performed in a limited memory setting to reduce storage, computations and the need to calculate Jacobian information. These hybrid algorithms are based on RK methods of at least order two, and a curvilinear search is implemented instead of the standard line search used in quasi-Newton algorithms. Step size control techniques are also performed to control the step size associated with the underlying Runge-Kutta methods.

All the methods considered in the preceding paragraphs are based on different ways of computing (approximate) solutions to the continuous steepest descent equation (2.4). Before concluding this section we mention an interesting ap-

proach devised by Branin & Hoo [14] which uses the differential equation (2.5) and hence follows a *continuous Newton path*. It might appear at first that a good numerical solution of (2.5) would involve the use of third derivatives and hence be quite expensive. However [14] shows that an effective and relatively cheap solution method can be employed using the fact that at every point along the solution curve $x(t)$ of (2.5) the gradient vector $g(x(t))$ remains parallel to the initial gradient $g(x(0))$. This observation motivates Branin & Hoo to propose a simple predictor-corrector version of the explicit Euler method for computing a good approximation to the continuous Newton trajectory. Branin & Hoo also suggest a quasi-Newton version of their approach which uses (2.6) where H is an updated approximation to the true inverse Hessian.

The Branin & Hoo approach does not make any significant distinction between the cases where G (or H) is positive definite or non positive definite. This is because the authors intend it to be a method of *global* optimization which works by finding *all* stationary points of a function. The idea is for the algorithm to proceed from an initial point via the continuous Newton path until any stationary point is reached whether this is a maximum, minimum or saddle point. Branin & Hoo then propose that the existing trajectory is projected through the stationary point and that the calculation is continued with a reversal of sign on the right hand side of (2.5) (or (2.6) until another stationary point is found. Repeated application of this process can sometimes result in a trajectory that does indeed pass through all stationary points. In practice however the outcome is not always so straightforward and [14] includes a restarting procedure that is sometimes needed when a trajectory diverges to infinity. These issues are outside the scope of this thesis since our objective is simply to find a local minimum.

This reference list is by no mean exhaustive, as there are many other authors who are continuing to work in this area; but it gives some indication of research activity in minimising functions by solving system of ordinary differential equations.

Chapter 3

Approximating the CSDP

In the previous Chapter we introduced the idea of solving a minimization problem by following the solution path of the continuous steepest descent equation. In particular we have described an algorithm proposed by Behrman [7] which uses an analytic solution to the CSDP equation. Many other trajectory-following methods have also been proposed but most of them involve numerical solutions of the gradient equation. In this Chapter we shall outline an approach of this kind that we propose to develop further in this thesis. A review of some other methods of a broadly similar nature appeared in the previous Chapter.

In this Chapter we seek to develop practical algorithms for solving (1.1), by approximating the CSDP. As has been mentioned already, iterative methods for unconstrained minimization can experience difficulties, when the search enters a non-convex region of the objective function $f(x)$, i.e. when the Hessian matrix of the function is not positive definite. In this situation, methods based on minimising a convex quadratic model of the objective function become inappropriate. In the Newton method, for instance, the search direction

$$p = -G^{-1}g$$

may lead to a saddle point or a local maximum rather than a local minimum. Quasi-Newton methods, which rely on a positive definite estimate of the Hessian matrix cannot make a suitable update in the non-convex regions and so progress may be no better than the steepest descent approach.

One optimization technique which makes intuitive sense in non-convex regions is the *trust region* method. This uses a step which minimises a local quadratic model subject to some restriction on the two-norm of the step size. This gives a well defined search direction, whether or not the Hessian is positive definite. This method can be related to the *Continuous Steepest Descent Path* (CSDP) method, as we show in the next section.

3.1 Solving the continuous gradient equation

The Continuous Steepest Descent Path is defined as the solution of the initial value problem (2.4) - i.e.

$$\frac{dx}{dt} = -g(x(t)), \quad x(0) = x_0.$$

Suppose that the point x_k is an estimate of the local minimum obtained after k-iterations of the algorithm. We have already mentioned that solving this ODE can lead to the classical Newton step when the function is locally convex [7]. However it can also be a logical choice of search away from this point when x_k is in the non-convex region of the function. In practice there are key questions that need to be addressed, regarding the accuracy with which the solution path $x(t)$ needs to be obtained and the computational cost of doing so. Useful surveys of these questions have been given in the PhD theses of Brown [15] and Behrman [7].

In this thesis, we will consider a number of minimisation algorithms which use search directions based on the solution of (2.4). We shall also present some possible novel variations involving search along curvilinear directions.

Applying *Euler's method* to (2.4), a new estimate of a point on the gradient trajectory corresponding to $t = h$ can be given by

$$x_{k+1} = x_k - hg_k \tag{3.1}$$

where $g_k = g(x_k)$. This of course gives the steepest descent method.

On the other hand by using the *Implicit Euler method* we get a different estimate

$$x_{k+1} = x_k - hg_{k+1}. \tag{3.2}$$

If G_k, denotes $G(x_k)$, then (3.2) can be approximated by

$$x_{k+1} = x_k - h\left(g_k + G_k(x_{k+1} - x_k)\right).$$

Hence $x_{k+1} = x_k + p_k$ where p_k can be found by solving the following system of equations

$$(I + hG_k)p_k = -hg_k. \tag{3.3}$$

In this case, even when G_k is non-positive definite, (3.3) gives a step p_k which decreases f, so long as h is sufficiently small.

We can also consider calculating x_{k+1} by a second order method which combines (3.1) and (3.3) in mixed explicit/implicit Euler step so that

$$x_{k+1} = x_k - \frac{h}{2}\left[\ g_k + (I + hG_k)^{-1}g_k\ \right]. \tag{3.4}$$

This can be written as

$$x_{k+1} = \frac{1}{2}(x^E_{k+1} + x^I_{k+1})$$

where x^E_{k+1} comes from (3.1) and x^I_{k+1} comes from (3.3).

It is important to observe that equation (3.3) also gives a step similar to that of the *trust region methods* which calculate search directions by solving

$$(\mu I + G_k)p_k = -g_k \tag{3.5}$$

where μ in (3.5) is effectively the reciprocal of the step length h in (3.3). Equation (3.3) gives the Newton step as $h \to \infty$ while (3.5) gives the Newton step when $\mu = 0$. Also equation (3.3) makes p_k parallel to $-g_k$ when $h = 0$ while (3.5) makes p_k tend to a steepest descent step as $\mu \to \infty$. The equation (3.5) will always give a descent direction p_k, even when G_k is not positive definite, as long as μ is chosen sufficiently large.

The derivation of (3.5) is that it gives a p_k as the step that minimises the quadratic model function i.e.

$$\underset{p}{\text{minimise}} \quad \frac{1}{2}p^T G_k p + p^T g_k + f_k \qquad \text{subject to} \quad \|p\|_2 \leq \Delta \tag{3.6}$$

where the step size limit Δ is a function of μ. In the trust region method, Δ is chosen on each iteration and the parameter μ has to be adjusted so that the solution to (3.5) satisfies $\|p\|_2 \leq \Delta$. At the next iteration the value Δ is increased if the new point $x_{k+1} = x_k + p_k$ gives an acceptable function decrease. On the other hand if $f(x_{k+1}) \geq f(x_k)$ then Δ is reduced and the sub-problem is solved again to give a better step p_k. One of the difficulties of solving the sub-problem (3.6) by varying the parameter μ in (3.5) is that the relationship between Δ and μ is highly nonlinear and iterative adjustment to find an appropriate μ may not be easy. Practical trust region algorithms may therefore deal with (3.6) in a different way – for instance by treating it directly as a constrained optimization calculation. For a major survey of trust region methods see [23].

There is a strong connection between equations (3.3) and (3.5) and this will allow us to state that, whether Hessian matrix G_k is positive definite or not, (3.3) will give us a step p_k such that $f(x_k + p_k) < f(x_k)$ as long as h is sufficiently small. (3.5) will give us a step p_k such that $f(x_k + p_k) < f(x_k)$ as long as μ is sufficiently large.

We shall be concerned with tracing out a curved path away from x_k by solving (3.5) for various values of μ. The value of μ is *not* adjusted in order to cause $||p_k||$ to be less than or equal to some pre-set limit Δ; instead μ is used like a search parameter to obtain a satisfactory decrease in the objective function. When G_k is non-positive definite we can trace out a path away from x_k by using a sequence of μ values chosen in a range $0 < \mu_{min} < \mu < \infty$, where μ_{min} is the absolute value of the most negative eigenvalue of G_k. In the case when G_k is positive definite it is natural to use $\mu = 0$ and obtain p_k as the classical Newton step. This gives the quadratic convergence near the minimum. However, we shall also show that we can sometimes use negative values of μ to extrapolate beyond a Newton step

when x_k is not sufficiently near a minimum for for a quadratic convergence to occur.

It is important to note that the points x_k+p_k obtained by solving (3.5) for varying μ will in general lie on a curved path. Hence a feature of the methods we are discussing is that they involve *curvilinear* searches rather than the more usual line search. When solving (3.5) for several different values of μ, we can

(i) use a fresh Cholesky factorisation of the coefficient matrix for each μ, or

(ii) simplify the solution of (3.5) for second and subsequent values of μ by determining the eigensystem of G_k via the orthogonal factorisation $G = RDR^T$.

In case (ii) since $RR^T = I$, the system (3.5) can be written

$$R(\mu I + D)R^T p_k = -g_k$$

and to solve for each value of μ we may use

$$\begin{aligned} \hat{g}_k &= R^T g_k; \\ \hat{p}_{k,i} &= \frac{\hat{g}_{k,i}}{\mu + d_i}, \qquad i = 1, ..., n; \\ p_k &= -R\hat{p}_k. \end{aligned} \tag{3.7}$$

This calculation has something in common with the correction step used by Behrman's gradient flow method outlined in equation (2.8) of the previous Chapter. It is important to emphasise that (3.7) uses the diagonal matrix D which contains the eigenvalues of G_k while Behrman's method uses the matrix Λ defined in (2.9) which involves exponentials of these eigenvalues. Behrman's method is derived from the *analytical* solution to a linearisation of the gradient equation (2.4) whereas the correction step (3.7) is based on an implicit Euler *approximate* solution to a linearised form of (2.4).

For a single solution of (3.5), the eigenvalue calculation is more expensive than a Cholesky factorisation. But if several values of μ are tried then subsequent solutions via (3.7) or (2.9) may be cheaper than re-factorisation. Therefore the

practical merit of using (ii) in the CSDP method depends on how far and how accurately we want to pursue a curved path solution of (2.4).

When G_k is positive definite we have a choice between using (3.5) with a curvilinear search or reverting to a standard Newton step coupled with a line search. The latter will work well on functions that are nearly quadratic; but in certain cases the former may be more effective.

3.2 Searching along the curved path

We consider first the case when G_k is non positive definite. The Newton step with $\mu = 0$ in (3.5) is not appropriate because it is likely to lead towards a maximum or saddle point. Hence we try a sequence of values for $\mu > \mu_{min}$ where $\mu_{min} = |d_p|$, the most negative element in D.

To trace out an approximate CSDP from x_k we may first select a suitably large initial value of μ, - i.e. one which gives a quite small step in a near steepest descent direction. We continue to try μ values decreasing towards μ_{min} so long as (3.5) yields a new point $x_k + p_k$ *close enough* to the CSDP. Our intention is to make significant progress along this path to reach an acceptable new point $x_k + p_k$ where the Hessian will be recomputed.

Our aim is to determine μ in (3.5) so as to ensure that p is a downhill step which produces an "acceptable reduction" in the objective function. We can do this by imitating the Wolfe condition for a conventional line search [33] - i.e. we want μ to imply that $f(x_k + p_k) < f(x_k)$ and also both $|f(x_k + p_k) - f(x_k)|$ and $\|p_k\|$ are bounded away from zero by a multiple of $\|g_k\|$. This might be done by comparing the actual change in F with a first or second-order predicted change.

Suppose we choose

$$\mu = \alpha\mu_{min} \quad \text{for some} \quad \alpha > 1, \tag{3.8}$$

and compute the corresponding p from (3.5). Then Evaluate $f^+ = f(x + p)$,

$g^+ = g(x+p)$ and calculate one or more of the test ratios.

$$D_1 = \frac{f^+ - f}{p^T g}, \tag{3.9}$$

$$D_2 = \frac{|f^+ - (f + p^T g + \frac{1}{2} p^T G p)|}{|p^T g + \frac{1}{2} p^T G p|}, \tag{3.10}$$

$$D_3 = \frac{(g + Gp)^T g^+}{\|g + Gp\| \|g^+\|}. \tag{3.11}$$

Note that subscript (k) has been suppressed on the right hand side of the above equations.

D_1 compares the actual change in f with a first order prediction.
Now if,

- $D_1 \approx 1$ this suggests that the step is too short.
- $D_1 < 0$ indicates the search has gone too far past the one-dimensional minimum.
- $D_1 = 0.5$ corresponds to the one dimensional minimum along p if f is quadratic.

D_2 compares the actual reduction in f with the quadratic prediction and if the difference is relatively small (i.e. $D_2 \approx 1$) then we can suppose that $x+p$ is quite close to the true solution path of (2.4). Hence it seems reasonable to continue to extrapolate along the CSDP (so long as D_1 exceeds some positive threshold).

D_3 compares the actual gradient with the quadratic model gradient (in terms of cosine of the angle between them). Thus it is reasonable to keep extrapolating if D_3 is close to 1 (again provided $D_1 > 0$). In this case we may deduce that there is progress to be made with decreasing values of μ. Once we have computed the test ratios then we shall find either

(i) $x + p$ is acceptable as a stopping point for the iteration

(ii) $x + p$ is acceptable but it is worth extrapolating further by decreasing μ

(iii) $x + p$ is unacceptable and we must interpolate by increasing μ.

3.3 Algorithm for searching along CSDP

The algorithm for a single iterative step can now be formalised.

Algorithm 3.3.1 *(Outline CSDP algorithm for non-convex regions)*

Step 1 *Given the parameters are:* $\alpha > 1$, $\beta < 1$, $\gamma < 1$, D_1^{min}, D_1^{max}, D_2^{max} *and* D_3^{max}.
If G_k *is positive definite go to* ***Step 7****. Otherwise ...*

Step 2 *Set* $\mu = \alpha\mu_{min}$

Step 3 *Compute* p *from (3.5) and hence get* $x + p$, f^+, g^+, D_1, D_2, D_3.

Step 4 *If* $D_1 < D_1^{min}$ *set* $\mu = \mu + \gamma(\mu - \mu_{min})$ *(to interpolate) and go to step 3*

Step 5 *If* $D_1 > D_1^{max}$ *and* $|1 - D_2| < D_2^{max}$ *and* $|1 - D_3| < D_3^{max}$
set $\mu = \mu - \beta(\mu - \mu_{min})$ *(to extrapolate) and go to step 3*

Step 6 *Otherwise* $x + p$ *is acceptable and the iteration is compete.*

Step 7 *Set an initial value of* $\mu = 0$ *and then compute* p, $x+p$ *and* $f = f(x + p)$.

Step 8 *Compute the test ratio* D_1.

Step 9 *If* D_1 *is too small or negative then* $x+p$ *is unacceptable and* μ *is replaced by*

$$\mu \leftarrow \mu + \gamma(\mu - \mu_{min})$$

where in this case μ_{min} *is the negative of the smallest eigenvalue of* G.

Step 10 *If* D_1 *is close to 1, extrapolate by decreasing* μ *below zero, and this is done by replacing* μ *by*

$$\mu \leftarrow \mu - \beta(\mu - \mu_{min}) \tag{3.12}$$

The curvilinear search algorithm sketched above can be combined with several different ways of calculating p_k.

- the calculation of p from (3.3) and (3.7) gives the algorithm Nimp1 or
- the calculation of p from (3.4) gives the algorithm Nimp2 or
- the calculation of p from (2.8) and (2.9) gives the algorithm UNMIN.

3.3.1 Nimp1 and Highams trust-region method

We have mentioned previously that using (3.5) to solve the trust region subproblem (3.6) can be difficult because of the nonlinear relationship between μ and the trust region radius Δ. Higham [41] has proposed a trust-region method which works with μ directly and does not use Δ at all. It simply uses the fact that increasing μ implies a decrease in $||p_k||$ while a decrease in μ implies an increase in $||p_k||$.

Because Higham's method works only with μ it resembles Nimp1 more closely than most other trust-region methods. In order that we can discuss the similarities and differences between these two methods we first state Higham's algorithm using the notation of the present Chapter.

Algorithm 3.3.2 *(Outline of Higham's trust region algorithm)*
Compute $f_k = f(x_k)$, $g_k = \nabla f(x_k)$ *and* $G_k = \nabla^2 f(x_k)$
If $\lambda_{min}(G_k + \mu_k I) \geq \epsilon$
    Solve $(G_k + \mu_k I)p_k = -g_k$
    Compute $\delta f_k = f_k - f(x_k + p_k)$
    Compute $\delta q_k = f_k - q_k(p_k)$
    Compute $r_k = \delta f_k / \delta q_k$
    Set $\mu_{k+1} = V(r_k, \mu_k)$ *using (3.13)*
else
    set $r_k = -1, \mu_{k+1} = 2\mu_k$ *(and regard* p_k *as zero)*
end if
If $r_k \leq 0$
    set $x_{k+1} = x_k$
else
    set $x_{k+1} = x_k + p_k$
end if

The algorithm involves the function

$$V(r, \mu) = \begin{cases} 2\mu, & r < \frac{1}{4} \\ \mu, & \frac{1}{4} \leq r \leq \frac{3}{4} \\ \frac{1}{2}\mu, & \frac{1}{4} < r. \end{cases} \tag{3.13}$$

An important difference between the above algorithm and Nimp1 is that the Higham method *usually* employs only one value of μ per iteration because adjustment of μ takes place between iterations. Thus, for instance, if the step p_k produces a 'good' reduction in objective function there is no attempt at extrapolation by using a reduced μ with the current values of g_k and G_k. Instead a reduced value of μ is employed on the *next* iteration. Similarly, if the reduction in objective function is merely 'acceptable', the value of μ is left unchanged by Higham's method. In contrast, Nimp1 typically uses a different value of μ on each iteration and can therefore be thought of as being more flexible. The situation in which Higham's method most resembles Nimp1 is when the current value of μ produces an increase (or an unacceptably small decrease) in the objective function. In this case both methods would repeat the solution of (3.5) with the same g_k and G_k but with an increased value of μ.

Higham [41] does not quote any numerical experience with his method and so we cannot comment directly on performance differences between his approach and ours. He is able to prove some convergence properties for his algorithm and we shall consider these, and their implications for Nimp1 in Chapter 8 of this thesis.

3.4 Initial trials

We have written three MATLAB codes Nimp1, Nimp2 and UNMIN to implement the algorithms mentioned at the end of the previous section. UMINH is a version of Behrman's gradient flow method but using a different curvilinear search from [7].

3.4.1 Numerical results for test problem $T1$

As a simple test problem we consider the function $T1$ given by

$$f(x_1, x_2) = x_1 x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$$

with the initial condition $x_0 = (2.05, 1.6)^T$. Contours of this function are shown in Figure 3.1. We look at the CSDP solutions using the following values for the parameters in the algorithm given in section 3.3,

$$\alpha = 2, \qquad \beta = 0.75, \qquad \gamma = 0.5, \qquad D_1^{min} = 0.1,$$
$$D_1^{max} = 0.6, \qquad D_2^{max} = 0.1 \qquad \text{and } D_3^{max} = 0.75.$$

The results in Table 3.1 were obtained using MATLAB implementations of Nimp1, Nimp2 and UNMIN where the execution of these algorithms is terminated once $\|g(x_{k+1}\| < 10^{-6}$. The trust region and quasi-Newton methods were implemented using *fminunc* in the MATLAB optimization toolbox [46]. We denote these by TR and QN respectively.

The method in *fminunc* is described in [13], [19] and it is comparable with Nimp1 in that it uses the exact Hessian of the objective function and works with a trust region sub-problem on every iteration. The approach differs from Nimp1 however in not obtaining an exact solution to the trust region sub-problem but rather to restricting itself to a two-dimensional subspace. This subspace is defined by the negative gradient $-g_k$ together with *either* an approximate Newton direction, $p \approx -G^{-1}g_k$ *or* a direction of negative curvature d, such that $d^T G_k d < 0$. Obtaining the Newton direction or a direction of negative curvature could involve the solution of (3.5) with $\mu = 0$ or the calculation of the eigensystem of G_k. However the method in *fminunc* seeks to avoid doing as much work as Nimp1 on each iteration and hence it finds p or d, by applying a preconditioned conjugate gradient (PCG) method see [18] to the system $G_k p = -g_k$. When the search is far from the optimum the PCG method may be terminated with quite a low accuracy approximation to the Newton direction; and, in particular, if G_k is found to be non positive definite the PCG method returns a direction of negative curvature, rather than an approximate Newton direction.

Method	No of Its	No of fcn calls
Nimp1	7	10
Nimp2	7	13
UNMIN	18	25
TR	8	9
QN	12	19

Table 3.1: Results for the function $x_1x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$

We can observe that Nimp1 and Nimp2 need fewer iterations than other methods and -perhaps more significantly- they appear to be more efficient than UNMIN. We also note of course that the CSDP methods use more function evaluations than the trust region method approach. This is plainly due to the step size used in tracing out the CSDP - and this, in turn, depends on the rules for adjusting μ. We can surmise that a smaller value of α in step 2 of algorithm 3.3.1 or a larger value of β in step 5 would have given the same solution in fewer function calls. The important point to be drawn from this first example is that the use of curvilinear searches can reduce the number of iterations *and hence reduce the associated cost of computing second derivatives*. We recall that, once the cost of calculating the eigensystem has been accepted, each new point in the curvilinear search is relatively inexpensive.

Clearly the results in Table 3.1 correspond to a single set of parameter values in the outline CSDP algorithm. We shall consider the variation of some of these parameters on a wider selection of problems in the next Chapter. The CSDP convergence paths for Nimp1, Nimp2 and UNMIN are shown in Figures 3.1 - 3.3. The circled points mark the starts and ends of iterations and the dots indicate points obtained with different values of μ. It is clear from the Figures that the solution by all the three methods follows a different curvilinear path through the non-convex region.

Our experience with problem $T1$ encourages us to look further at the algorithm Nimp1, Nimp2 and UMINH to consider how they perform on a wider set of test problems and to explore their sensitivity to various parameter choices in the curvilinear search.

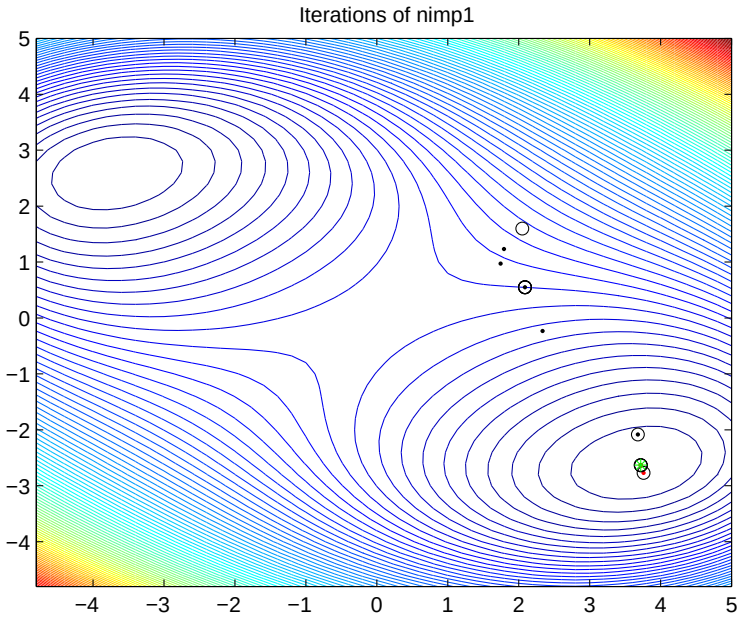

Figure 3.1: The solution path for Nimp1 on problem $T1$.

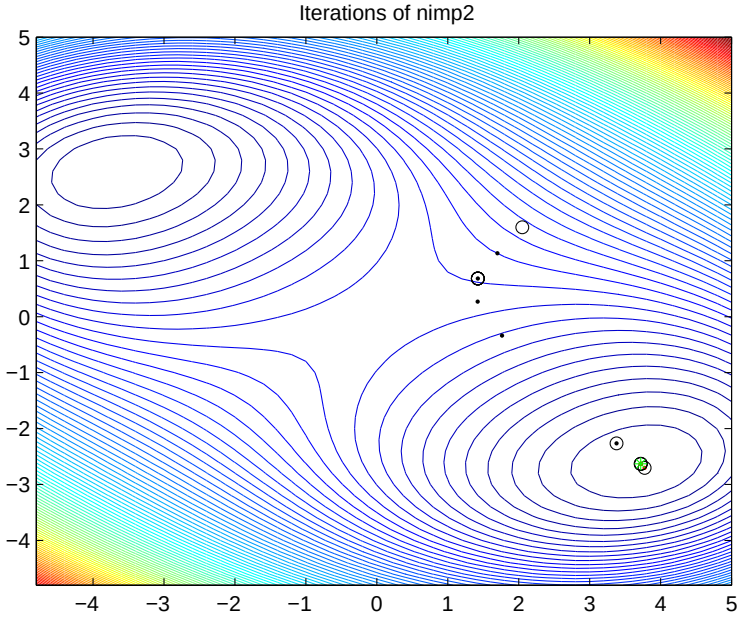

Figure 3.2: The solution path for Nimp2 on problem $T1$.

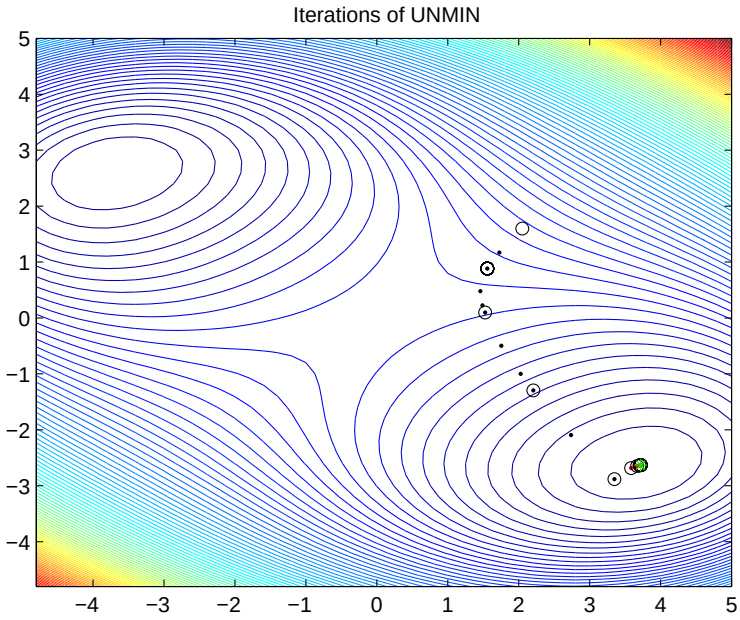

Figure 3.3: The solution path for UNMIN on problem $T1$.

Chapter 4

Further Algorithmic Investigation

4.1 Preliminaries

Results from problem $T1$ given in Chapter 3 suggest that the CSDP techniques are worth further investigation. The algorithm stated in section 3.3 involves several parameters and certain features within it are somewhat tentative, for instance the decision to base interpolation and extrapolation only on the ratio D_1 in section 3.2 when G is positive definite is based purely on observation of numerical results rather than any form of theoretical analysis. Similarly the formula for adjusting μ does not take into account whether D_1 and/or D_3 are close to their limits. In the next sections we shall consider the sensitivity of the methods to certain parameter values in the algorithm and how performance can be affected by different choices for some of them.

4.1.1 Choosing an initial μ for each iteration

We first look at the value of α in (3.8) which is involved in determining an initial value and the subsequent adjustment of μ on each iteration in order to ensure that the function value is "sufficiently less" than the previous one. As in trust region methods we could relate this to an estimate of the size of the step p_k.

Suppose $\delta_k = \|x_k - x_{k-1}\|_2$ is the size of the step taken to reach the current point x_k (δ_0 must be set arbitrarily). Suppose also that the Hessian G_k at the current point has factors given by

$$G_k = RDR^T \qquad \text{where } RR^T = I \text{ and } D \text{ is a diagonal matrix.}$$

The elements of D are the eigenvalues of G_k and the system (3.5) can be written

$$R(\mu I + D)R^T p_k = -g_k.$$

Hence the solution of (3.5) for each value of μ can be found via

$$\hat{g}_k = R^T g_k; \quad \hat{p}_k = \frac{\hat{g}_k}{\mu + d_i}, \quad i = 1, ..., n; \quad p_k = -R\hat{p}_k.$$

Because of the orthogonality of the matrix R, we deduce that

$$\|p_k\|_2 \leq \frac{1}{\mu + \lambda_{min}} \|g_k\|_2$$

and in order to give $\|p_k\|_2 < \delta_k$, we require

$$\frac{1}{\mu + \lambda_{min}} \leq \frac{\delta_k}{\|g\|_2} \text{ and so } \mu \geq \frac{\|g_k\|_2}{\delta_k} - \lambda_{min}.$$

Since we must have $\mu > \mu_{min}$, an initial μ for iteration can be obtained from the safeguarded formula

$$\mu = \text{maximum} \left(\alpha \mu_{min}, \frac{\|g_k\|_2}{\delta_k} - \lambda_{min} \right) \tag{4.1}$$

for some $\alpha > 1$ when the Hessian G is non-positive definite and

$$\mu = \text{maximum} \left(0, \frac{\|g_k\|_2}{\delta_k} - \lambda_{min} \right)$$

when G is positive definite - i.e. we put an initial restriction on the Newton step too. So now we have a modified algorithm for a CSDP step in which δ_0 is defined arbitrarily on the first iteration and is the last successful step length on all other

iterations. Acceptability of the new point $x_k + p_k$ is also based on the values of one or more of the test ratios as described in section 3.2.

Some results with this safeguarded formula are given in Table (4.1). The first rows of the table show what happens when the CSDP methods are applied to problem $T1$ using $\mu = \alpha\mu_{min}$ for a fixed value of α on each iteration while the last row uses the formula (4.1) with $\alpha = 2$. The other parameter values in the algorithm (3.3.1) are

$$\delta_0 = 1, \qquad \beta = 0.5, \qquad \gamma = 0.25, \qquad D_1^{min} = 0.1$$

$$D_1^{max} = 0.6, \qquad D_2^{max} = 0.1 \qquad \text{and} \qquad D_3^{max} = 0.5$$

$\alpha =$	Nimp1	Nimp2	UNMIN
	Its/Fcs	Its/Fcs	Its/Fcs
1.5	7/10	5/9	17/33
2	7/12	7/13	13/30
3	7/14	7/15	18/37
4	7/16	5/14	17/35
5	7/16	7/17	16/36
10	7/18	6/19	18/37
15	5/18	6/19	17/40
20	6/20	6/20	18/41
30	6/21	5/20	17/43
50	7/24	5/22	18/45
100	7/25	5/24	17/48
200	7/27	5/26	17/52
1000	6/31	7/33	18/61
$(4.1)\alpha$	6/13	6/13	18/30

Table 4.1: Results using different values of α in CSDP methods applied to problem $T1$

It is clear that varying α has an appreciable effect on the numbers of function evaluations and, to a lesser extent, on the numbers of iterations. It is also clear that we cannot use $\alpha = 1$ since this would make (3.5) a singular system. How-

ever it is interesting that we can take α fairly close to 1 without encountering difficulties. On the other hand, large values of α may reduce the number of iterations but also imply that more steps are taken along the curvilinear path at each iteration, giving a corresponding increase in function calls. The automatic choice (4.1) seems to yield a good compromise.

4.2 Varying the parameter D_3^{max}.

We can also consider varying the threshold on the accuracy parameter D_3 which controls how far the search is pursued along the approximate CSDP. Using the parameter value $D_3^{max} = 0.75$ on test problem $T1$, the same results as in Table 4.1 were obtained.

Figures 4.1 – 4.3 relate to problem $T1$ and show how the Nimp1 path varies as D_3^{max} changes.

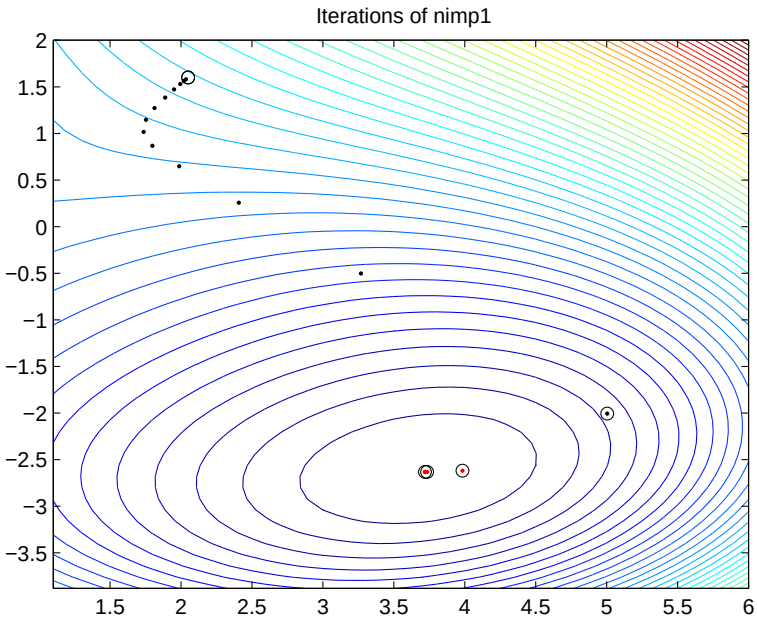

Figure 4.1: The solution path for Nimp1 on problem $T1$ using $D_3^{max} = 0.5$

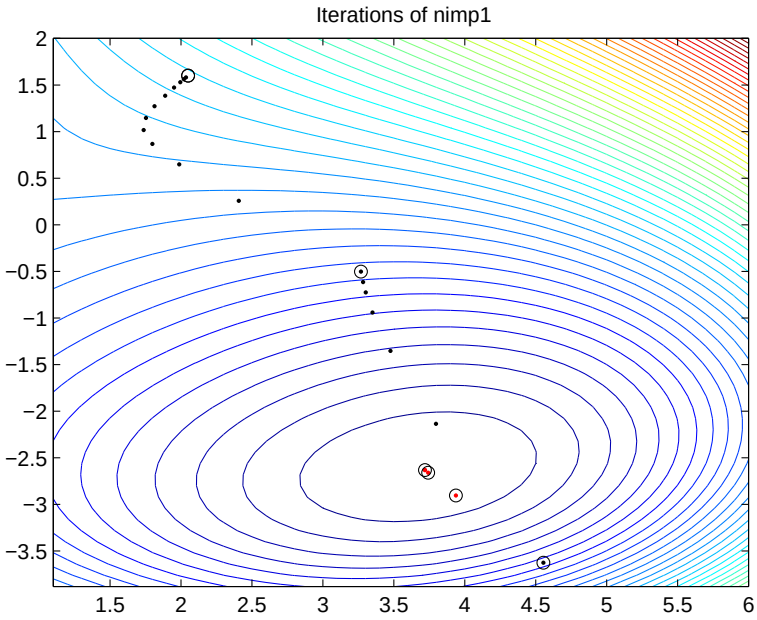

Figure 4.2: The solution path for Nimp1 on problem $T1$ using $D_3^{max} = 0.05$

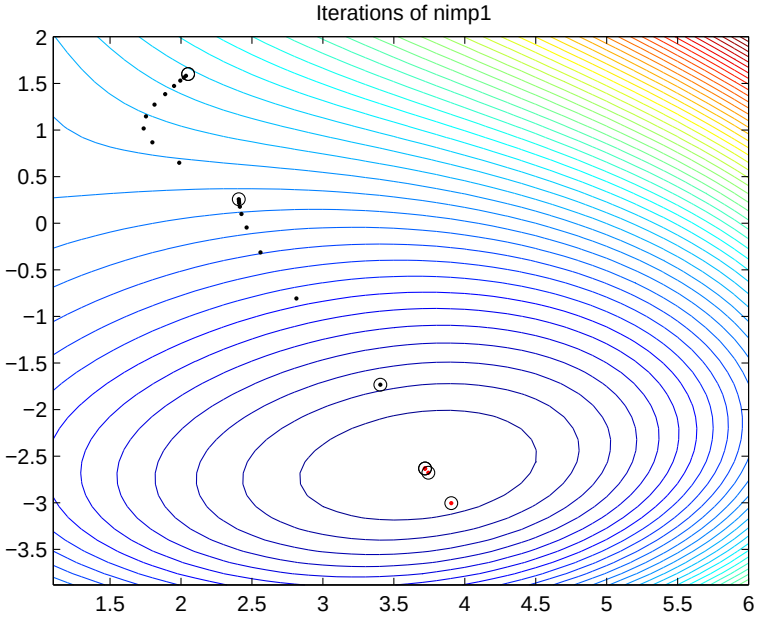

Figure 4.3: The solution path for Nimp1 on problem $T1$ using $D_3^{max} = 0.01$

Figure 4.1 shows that, in some sense, the test $|1 - D_3| < 0.5$ lets the first iteration go "too far" and obtains a point x_1 lying some way off a direct route to the minimum. On the other hand, insisting that $|1 - D_3| < 0.05$ or $|1 - D_3| < 0.01$ (Figure 4.2 and 4.3) may not let the first search go far enough, leaving the second iteration with some work still to do to escape from the non-convex region. In fact, if we count the dots, we find that Figure 4.1 represents the solution using the fewest function evaluations.

4.3 Behaviour close to the the saddle point

We now consider how Nimp1 behaves on problem $T1$,

$$f(x_1, x_2) = x_1 x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$$

as starting point x_0 moves closer and closer to the saddle point, which in this case is $x_s = [0, 0]^T$. Evaluating the function, its gradient and Hessian at this point we get,

$$f = 1, \qquad g = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad G = \begin{bmatrix} -0.4 & 1.0 \\ 1.0 & -0.8 \end{bmatrix}$$

The eigenvalues of G, at this point are $d_1 = -1.6198$ and $d_2 = 0.4198$ with corresponding eigenvectors

$$v_1 = \begin{bmatrix} -0.6340 \\ 0.7733 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} 0.7733 \\ 0.6340 \end{bmatrix}$$

respectively. Now we choose starting values in the non-convex region that lie on the first eigenvector, e.g.

$$x_0 = \begin{bmatrix} 1 \\ 0.8199 \end{bmatrix}, \qquad \begin{bmatrix} 0.1 \\ 0.0819 \end{bmatrix}, \qquad \begin{bmatrix} 0.01 \\ 0.0081 \end{bmatrix}, \qquad \begin{bmatrix} 0.001 \\ 0.0008 \end{bmatrix}.$$

Figures 4.4-4.7 show the corresponding CSDP solutions.

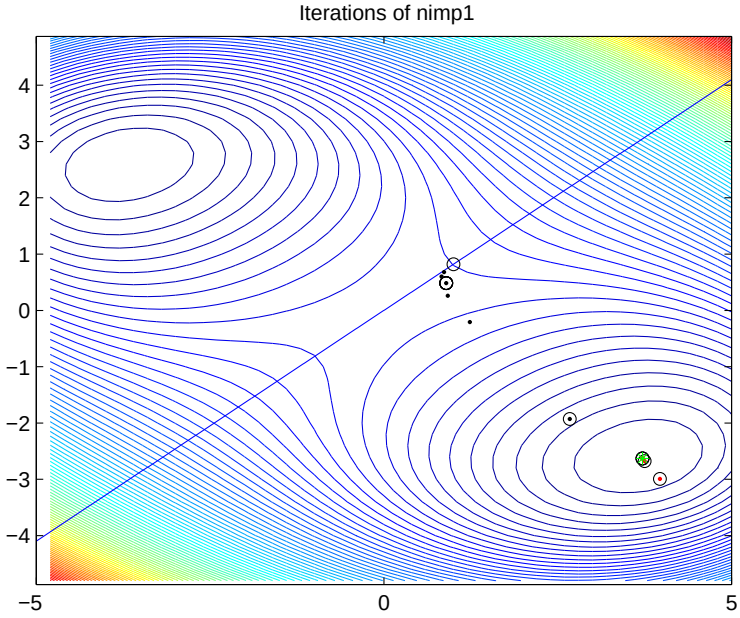

Figure 4.4: The solution path for Nimp1 on $T1$ using $x_0 = [1, 0.8199]^T$

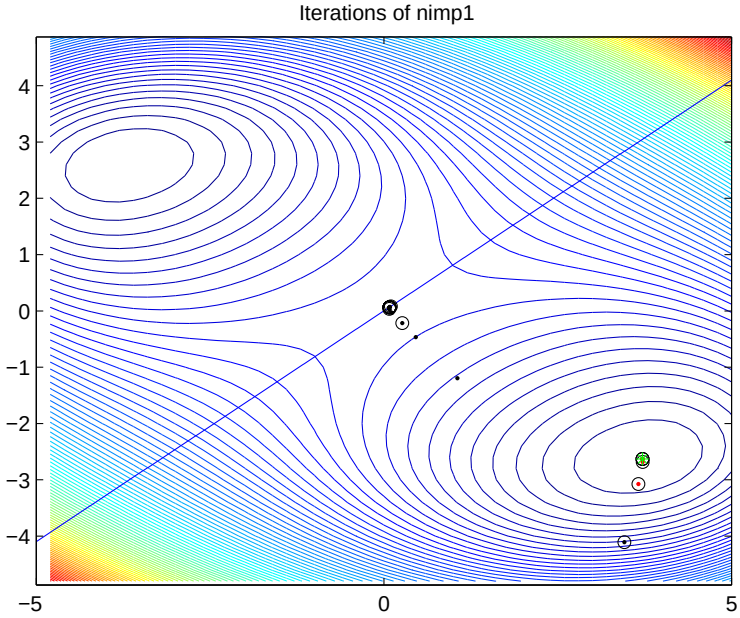

Figure 4.5: The solution path for Nimp1 on $T1$ using $x_0 = [0.1, 0.0819]^T$

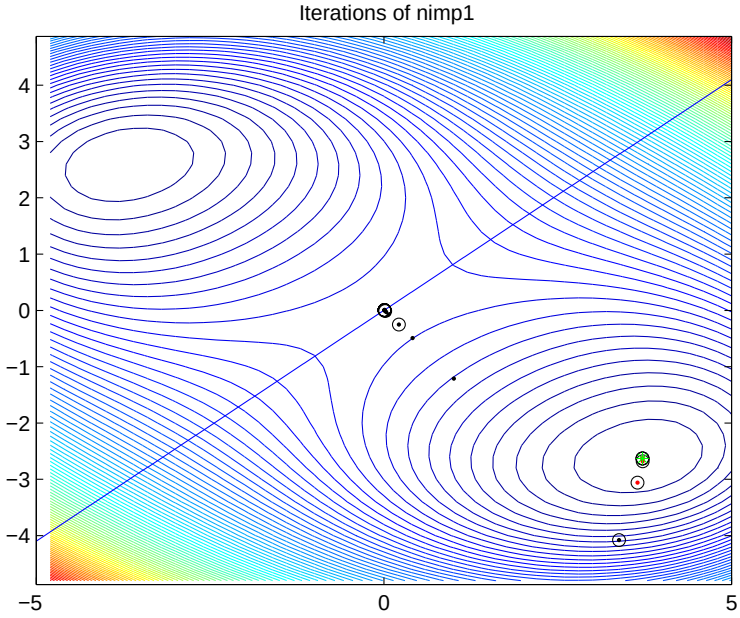

Figure 4.6: The solution path for Nimp1 on $T1$ using $x_0 = [0.01, 0.0081]^T$

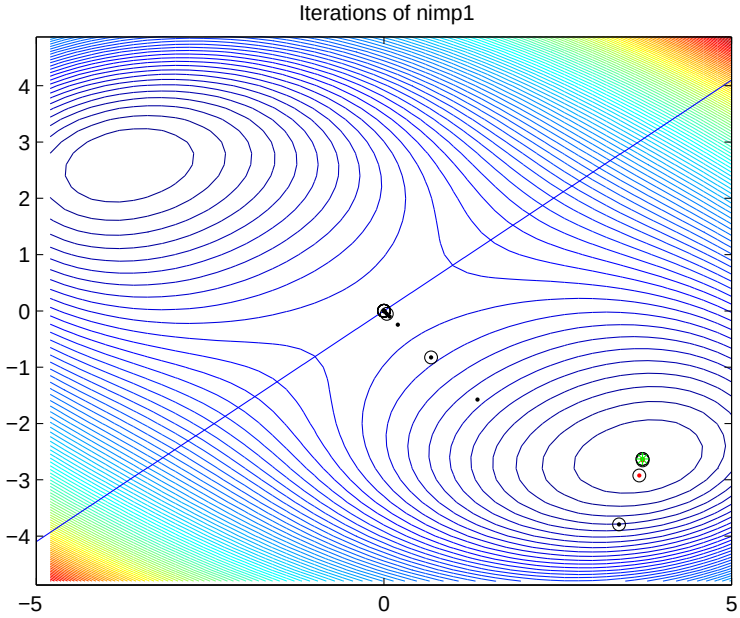

Figure 4.7: The solution path for Nimp1 on $T1$ using $x_0 = [0.001, 0.0008]^T$

x_0	Its	Fcs
$(1, 0.8199)^T$	7	13
$(0.1, 0.0.819)^T$	9	18
$(0.01, 0.0081)^T$	9	18
$(0.001, 0.0008)^T$	9	19

Table 4.2: Results using Nimp1 on problem $T1$ with different initial conditions

It can be seen from Figures (4.4) - (4.7) that even when starting very close to the saddle point (i.e. in the non-convex region where the gradient of the function ∇f is very close to zero) Nimp1 can find the minimum of the function and avoid the saddle point. This may be at a cost of more function evaluations the nearer x_0 is closer to the saddle, see Table (4.2). This was also noticed when using the other methods (see Table (4.3)).

$\alpha =$	Nimp2	UNMIN
x_0	Its/Fcs	Its/Fcs
$(1, 0.8199)^T$	5/12	20/32
$(0.1, 0.0819)^T$	6/18	26/50
$(0.01, 0.0081)^T$	6/18	27/51
$(0.001, 0.0008)^T$	6/18	28/55

Table 4.3: Results using Nimp2 and UNMIN in problem $T1$ with different initial conditions

4.4 Numerical results on specially constructed test problems

We now consider the performance of CSDP methods on a wider range of problems. These have been specially chosen to test the features of the CSDP methods and hence they involve functions with large non-convex regions - and sometimes saddle points - which are close to local minima. The problems are given in Appendix D. The functions of the form $F(x) = -1/(1+\phi(x))$ are suggested by the shape of the famous Runge function used to show the inadequacies of polynomial interpolation. $F(x)$ will have a local minimum at the same point as $\phi(x)$, but as x moves

away from this minimum the function can be expected to become non-convex and to flatten out. Figure 4.8 illustrates this behaviour by showing the surface corresponding to problem $T4\_2$. If an optimization search is started in a flattened non-convex region of the kind shown in Figure (4.8) then a significant test of the CSDP approach will be to consider how effectively it is able to make progress towards the convex area near the minimum.

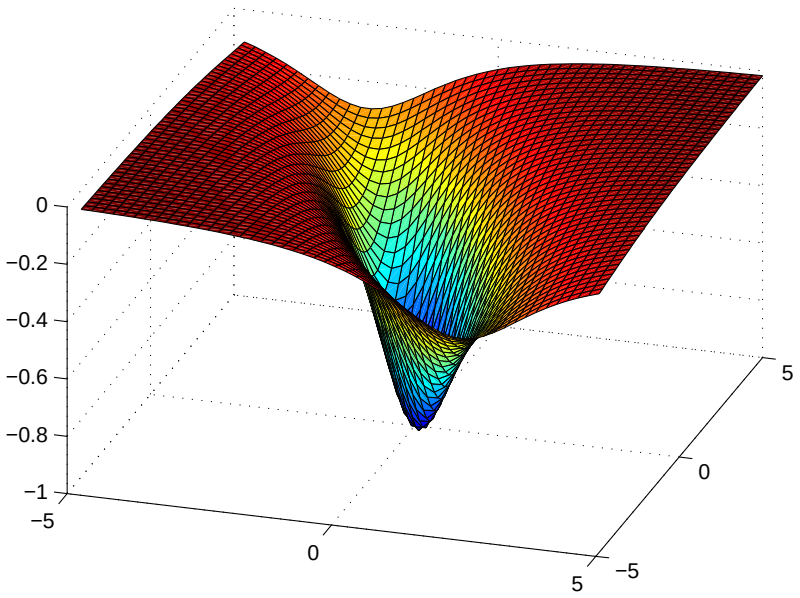

Figure 4.8: Surface Plot for Problem $T4\_2$.

Fctns	Methods	Nimp1	Nimp2	UNMIN	TR	QN	UMINH
	$\alpha =$	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
T1	2	7/10	7/13	18/25			
	100	**6/16**	**6/19**	17/33	*8/9*	12/19	8/49
	a	**6/10**	**6/10**	18/26			
T1r	2	*5/8*	6/10	15/22			
	100	6/16	7/21	14/37	9/10	11/28	F
	a	7/14	6/11	16/24			
T1r2	2	9/12	*6/10*	14/21			
	100	**5/12**	7/21	14/37	9/10	11/32	10/31
	a	8/14	7/12	15/23			
T1a	2	*5/8*	5/11	15/20			
	100	5/12	**4/12**	17/32	9/10	10/13	8/30
	a	5/10	***4/8***	17/24			
T1b	2	***5/10***	5/12	19/29			
	100	6/19	5/14	20/42	*9/10*	10/22	F
	a	7/11	5/12	19/29			
T1ar	2	7/14	**6/11**	16/32			
	100	7/24	7/20	16/43	*9/10*	11/31	11/32
	a	8/14	**6/11**	16/26			
T2	2	8/13	8/15	18/23			
	100	8/21	**7/17**	18/30	*9/10*	14/27	F
	a	8/13	**7/12**	19/27			
T2r	2	*6/9*	7/13	15/21			
	100	8/20	8/19	15/30	9/10	12/23	F
	a	7/15	7/10	13/18			
T3	2	9/17	7/16	24/38			
	100	8/28	**6/23**	21/54	*14/15*	*14/15*	F
	a	9/17	7/17	24/38			

Table 4.4: Results on special test problems from nimp1, nimp2, UNMIN, TR, QN and UMINH

Fctns	Methods	Nimp1	Nimp2	UNMIN	TR	QN	UMINH
	$\alpha =$	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
T4_2	2	**5/6**	6/11	14/17			
	100	11/25	12/31	19/33	8/9	7/18	6/31
	a	7/10	8/10	17/19			
T4_4	2	**7/24**	*8/15*	11/18			
	100	12/31	13/32	15/33	22/23	17/30	F
	a	12/16	10/14	13/15			
T4_10	2	**7/15**	9/15	11/14			
	100	14/36	14/39	20/43	*12/13*	16/31	7/28
	a	15/19	15/18	18/19			
T4_20	2	12/24	**9/15**	13/22			
	100	17/43	17/45	21/48	*12/13*	17/44	F
	a	**9/15**	19/26	21/24			
T4_50	2	11/25	12/39	14/26			
	100	18/48	19/51	24/54	15/16	23/61	F
	a	***10/13***	21/27	27/33			
T4_100	2	**12/35**	16/45	18/33			
	100	21/56	22/62	25/60	17/18	26/74	F
	a	*14/17*	24/37	30/40			
T4r_20	2	**9/16**	**9/17**	10/16			
	100	17/47	18/50	19/47	12/13	14/41	F
	a	***9/15***	19/26	18/21			
T5	2	7/11	9/14	20/25			
	100	8/21	10/26	20/36	*9/10*	**6/11**	8/36
	a	7/11	8/15	20/26			
T5a	2	***9/14***	14/44	22/29			
	100	10/27	17/44	22/44	18/19	***9/14***	F
	a	10/20	9/21	22/32			

Table 4.5: Further results on special test problems from nimp1, nimp2, UNMIN, TR, QN

In Tables 4.4 and 4.5 we summarise some results for these test problems using the parameter values $\delta_0 = 1$, $\beta = 0.5$, $\gamma = 0.25$, $D_1^{min} = 0.1$, $D_1^{max} = 0.6$, $D_2^{max} = 0.1$ and $D_3^{max} = 0.5$.

(It is worth noting, in view of the comments in section 4.2, that the results were very little changed when D_3^{max} was set to 0.25.)

The second column of Tables 4.4 and 4.5 shows the value of α used to choose an initial $\mu = \alpha\mu_{min}$ on each iteration. The symbol 'a' denotes the use of formula (4.1) with $\alpha = 2$. The tables also show the numbers of iterations and function calls needed by the truncated Newton/trust region method from the MATLAB optimization toolbox. For each problem in these tables we highlight in bold the entry which gives the best performance measured primarily in terms of the numbers of iterations. Whenever the entry which represents the best performance in terms of function evaluations is different from the one marked in bold, we distinguish it by italics. Finally, to reflect the fact that we are usually interested in both these measures, we underline the entry which gives the smallest sum of iterations and function calls. (We recognize of course, that these are rather unsophisticated ways of assessing performance which overlook the overall algorithmic costs in computing search directions etc.)

The results in Tables 4.4 and 4.5 show that Nimp1 consistently does better than Nimp2 and UNMIN. In particular, it seems that UNMIN is rarely competitive. Nimp1 also usually outperforms the Trust Region method in terms of iteration count - appreciably so on problems $T1$ group and $T2$. The choice of α does not greatly affect the numbers of iterations needed by CSDP methods. We may also note that Nimp2 and Nimp1 behave in a rather similar way on $T1$ problems while Nimp2 does slightly better on problems $T2$ and $T3$ on all other examples the performance of Nimp2 is not as competitive. Nimp1 is also less competitive with the quasi-Newton method on $T5$ problems.

It is not particularly surprising to observe that Nimp1 can often outperform the quasi-Newton routine (QN) from the MATLAB optimization toolbox [46] since this method does not use exact second derivatives. It is probably more significant to note that the pilot version of Nimp1 appears quite competitive with the trust region routine TR (also from [46]).

The code UNIMH in the final column of the tables represents an implementation of Behrman's gradient flow method [7] which uses his step control procedure

rather than our curvilinear search. Unfortunately, during the testing, we had to guess values for some of the search parameters which were not given in his thesis [7] and we have not been able to obtain very good performance. As can be seen from the results in Tables 4.4 and 4.5, UMINH failed on most of the problem and this may be due to the fact that the algorithm does not perform adequately in non-convex region for these specially selected test problems.

In the next section we consider the performance CSDP algorithms on a wider set of test problems.

4.5 Using the CUTEr test problems

We have tried our algorithms on 139 unconstrained problems taken from the Constrained and Unconstrained Test Environment revisited version (CUTEr). This offers a large set of optimization test problems and is created and maintained by Gould, Orban and Toint [38]. The algorithms and the test problems have been implemented using MATLAB for Linux with the starting point x_0 provided by CUTEr.

An important aspect of any algorithm is the convergence test for terminating the iterations. We have used the gradient norm $\|\nabla f\|_2 \leq 10^{-6}$ as the stopping criterion, and taken 10000 as the maximum permitted number of iterations. The experiments compare Nimp1, Nimp2, UNMIN (this is identical to Behrman's algorithm but using our own curvilinear search), UMINH, TR and QN the MATLAB trust region and quasi-Newton methods respectively. Note that QN method here does not use the exact Hessian and so the under performance is to be expected. QN will be looked at in more detail when compared with the similar CSDP methods in Chapter 5.

The CUTEr results are summarised in Appendix A. It can be seen that Nimp1 matches or improves on the performance of most other methods, in over 25% of cases. It beats the TR method on over half the problems. TR fails to reach the minimum on 9 problems, but Nimp1 finds a solution in all cases. Nimp2 also

does well on around 30% of the examples. However it fails on 6 problems.UNMIN wins in 20% of cases and it did really well on problems such as the PALMER*C set, SENSOR and SPARSINE. But it fails on 11 problems which include the QUARTC and DENSCHNB problems. This may be due to some small curvilinear steps being taken or even to cycling within the iterations.

TR did best on 28% of the problems. However it often seems to terminate prematurely at points that do not satisfy the first order optimality conditions. The MATLAB optimization toolbox uses a stopping rule based on the size of change in the objective function and this seems to be activated too soon on problems HEART8LS, HYDC20LS and LOGHAIRY for example.

Overall therefore the performance of CSDP methods is quite encouraging. One negative aspect of the results however is that all the methods, including Nimp1, can sometimes do badly compared with the best result. There is a degree of inconsistency in behaviour that still needs investigation. One way of trying to counteract such inconsistencies is to compare methods using *performance profiles* as proposed by Dolan & Moré [30] and we shall now use this approach based on two performance metrics, the number of iterations needed to attain the desired accuracy and the corresponding number of function evaluations. These both give information on a solver's robustness and efficiency.

As in Dolan and Moré [30], given a set of problems P and a set of solvers S , a performance profile of a solver, $s \in S$, is a plot of the probability distribution function

$$\rho_s(\tau) = \frac{1}{n_p} size \{p \in P : r_{p,s} \leq \tau\}$$

for a given performance metric, where $n_p \in P$ is the number of problems, $r_{p,s}$ is the performance ratio

$$r_{p,s} = \frac{t_{p.s}}{\min\{t_{p,s} : s \in S\}}$$

and $t_{p,s}$ denotes the number of iterations (number of function evaluations) to solve problem $p \in P$ with solver s to the required accuracy of convergence. It is easily seen that the fastest solver has $r_{p,s} = 1$ and if the solver fails then $r_{p,s} = \infty$. It

was suggested by Dolan and Moré [30] that if a method does not solve a problem, r_p, s is simply given a large value.

$\rho_s(\tau)$ represents the overall probability that the performance of solver s on any problem will be within a factor τ of the best possible performance. If we plot $\rho_s(\tau)$ for a number of solvers then the most efficient method will be the one for which ρ_s is highest on the left hand side of the plot. On the other hand the method for which ρ_s is highest on the right-hand side of the plot may be considered the most reliable.

Performance profiles will be now used to compare the solvers Nimp1, Nimp2, UNMIN, UMINH, TR and QN on 139 test problems from the CUTEr collection. Figures 4.9 and 4.10 correspond to the use of iteration count as performance metric while Figures 4.11 and 4.12 show performance profiles based on numbers of function evaluations.

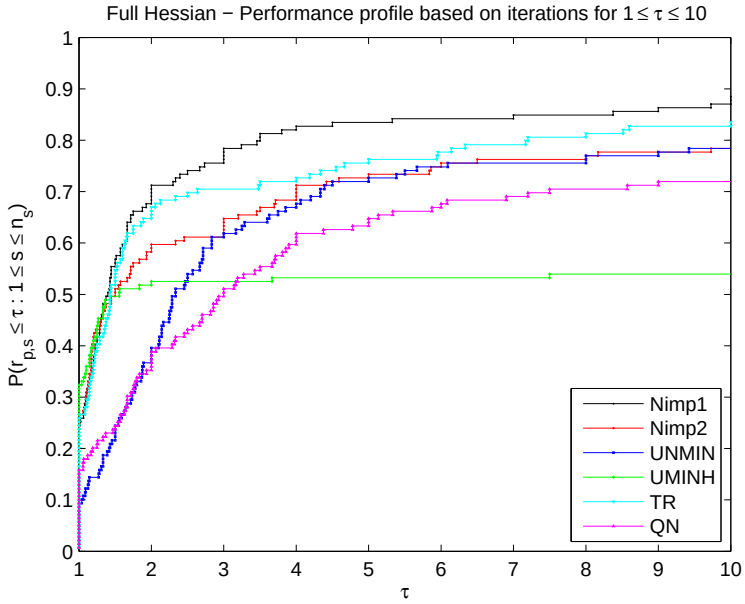

Figure 4.9: Full Hessian: Comparison of the methods based on iterations for $1 \leq \tau \leq 10$.

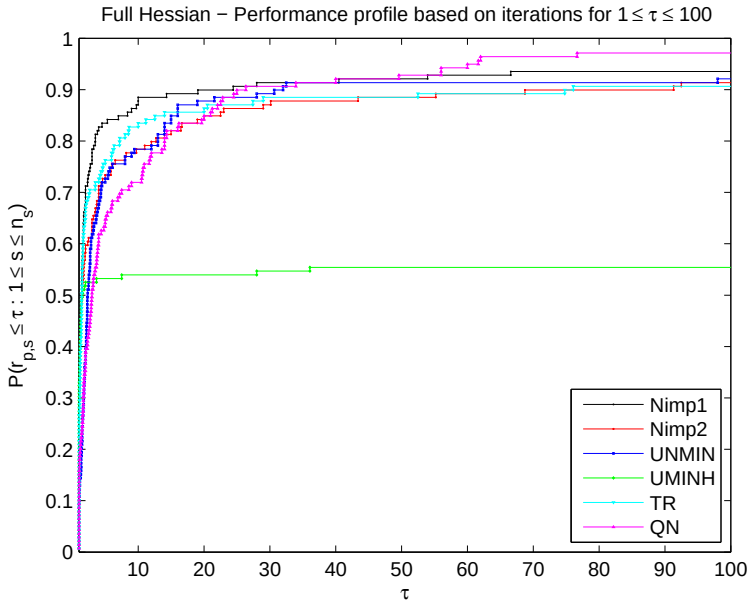

Figure 4.10: Full Hessian: Comparison of the methods based on iterations for $1 \leq \tau \leq 100$.

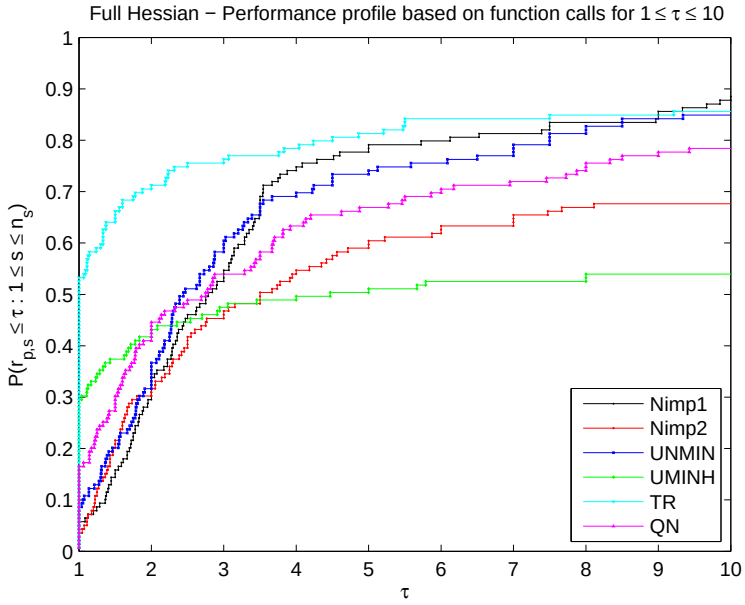

Figure 4.11: Full Hessian: Comparison of the methods based on function calls for $1 \leq \tau \leq 10$.

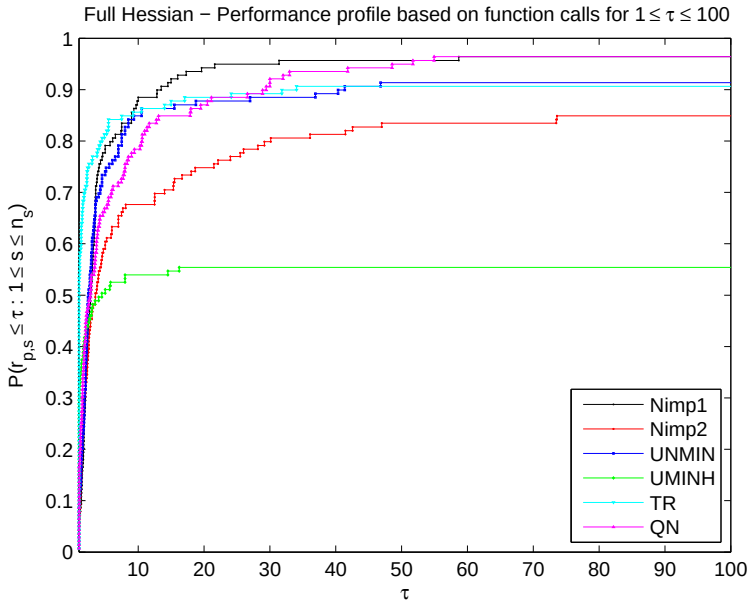

Figure 4.12: Full Hessian: Comparison of the methods based on function calls for $1 \leq \tau \leq 100$.

For small to moderate values of τ, Nimp1 and TR appear to outperform the other methods. When iteration count is the metric then Nimp1 seems to do better, but TR is more successful when performance is measured in terms of function calls. This suggests that TR often gets a good step at first attempt on each iteration while Nimp1 needs several trial steps of its curvilinear search.

For larger values of τ the picture becomes more confused and all methods - with the clear exception of UMINH - appear quite competitive in terms of its iteration count. UMINH also performs quite poorly in terms of function calls (except rather surprisingly when $1 < \tau < 2$). The fact that UNMIN usually outperforms UMINH suggests that our form of curvilinear search is an improvement on Behrman's step length strategy in [7].

QN on the other hand does unexpectedly well when $\tau > 50$; but Nimp2 seems usually to be inferior to Nimp1 and we shall not develop this approach in the work that follows.

We note that the codes TR and QN are MATLAB library routines and we can neither fine-tune nor comment on any details of their calculations.

Chapter 5

Approximating the Hessian

The algorithm Nimp1 described in the previous chapter can be viewed as a version of Newton's method in which the correction step δ_k on the k-th iteration is given by

$$\delta_k = -(\mu I + G_k)^{-1} g_k \tag{5.1}$$

where $\mu \geq 0$ is chosen so as to ensure an acceptable decrease in the objective function via a condition such as

$$f(x_k + \delta_k) < f(x_k) + \nu \delta_k^T g_k \tag{5.2}$$

for some positive constant ν. The correction step (5.1) is a variation on the classical Newton correction

$$\delta_k = -\alpha G_k^{-1} g_k \tag{5.3}$$

where $\alpha > 0$ is chosen in order to satisfy a condition such as (5.2). As is well known, the basic Newton step (5.3) is only guaranteed to be a descent direction when G_k is positive definite; but equation (5.1) will yield a downhill step even when G_k is not positive definite (providing μ is chosen sufficiently large).

One of the drawbacks with any form of Newton method is the need to calculate second derivatives, which may be a non-trivial process for large-scale practical problems. Since the 1960s, therefore, much work has been done to develop

quasi-Newton methods which calculate a correction step of the form

$$\delta_k = -\alpha B_k^{-1} g_k \tag{5.4}$$

where B_k is a positive definite approximation to the actual Hessian G_k. The calculation of each B_k is intended to be computationally cheaper than the calculation of the second derivatives in G_k. And by keeping B_k positive definite we avoid the difficulties encountered by Newton's method when the true Hessian loses positive definiteness. The next section gives a brief outline of some current methods of calculating suitable approximating matrices B_k for use in (5.4).

5.1 Quasi-Newton methods

Quasi-Newton methods are based on Newton's method but use an updated symmetric matrix B_k to approximate the Hessian matrix of the objective function on each iteration. The initial matrix B_0 is chosen arbitrarily to be positive definite e.g. as the unit matrix. In this section we look at some existing quasi-Newton (QN) methods for solving problem (1.1). We suppose it involves a differentiable objective function $f(x)$ with the gradient vector $g(x) = \nabla f(x)$.

Quasi-Newton methods are motivated by some drawbacks in the Newton method which is extremely fast near the minimum, but is also expensive and can be ineffective away from the solution. A typical quasi-Newton iteration calculates a correction step from (5.4) with α chosen to satisfy (5.2). The iteration is then completed by an updating process for obtaining a new Hessian approximation B_{k+1} for use on the next iteration. (Some quasi-Newton methods work with an approximation to the inverse of G or hold B_k in factored form in order to avoid the $O(n^3)$ operations involved in (5.4). These versions are not relevant to our present work.)

Quasi-Newton updates are designed to make B_{k+1} satisfy the secant condition

(quasi-Newton condition)

$$B_{k+1}\delta_k = \gamma_k = g(x_{k+1}) - g(x_k). \tag{5.5}$$

This gives B_{k+1} a property which would also hold for the true Hessian if the objective function were quadratic.

There are many well known updating formulae that satisfy (5.5). In this study, we focus on the three most popular updates namely the DFP, BFGS, and SR1 formulae.

The DFP updating formula was proposed by Davidon [27] and Fletcher and Powell [34] independently. Its form is

$$B_{k+1}^{DFP} = B_k + \left(1 + \frac{\delta^T B_k \delta}{\gamma^T \delta}\right) \frac{\gamma\gamma^T}{\gamma^T\delta} - \left(\frac{B_k\delta\gamma^T + \gamma\delta^T B_k}{\gamma^T\delta}\right). \tag{5.6}$$

The BFGS method was developed by Broyden [17], Fletcher [32], Goldfarb [37] and Shanno [58], and is

$$B_{k+1}^{BFGS} = B_k - \frac{B_k\delta\delta^T B_k}{\delta^T B_k \delta} + \frac{\gamma\gamma^T}{\gamma^T\delta}. \tag{5.7}$$

The symmetric Rank 1 (SR1) formula was suggested independently by many authors [33],

$$B_{k+1}^{SR1} = B_k + \frac{(\gamma - B_k\delta)(\gamma - B_k\delta)^T}{(\gamma - B_k\delta)^T\delta}. \tag{5.8}$$

(The subscript k has been omitted from δ and γ in the right hand sides of these formulae for ease of writing and reading.)

It can be shown that both the DFP and BFGS formulae will give positive definite B_{k+1} provided B_k is positive definite and $\delta_k^T\gamma_k > 0$. (If $\delta_k^T\gamma_k \leq 0$ the k-th iteration has passed through a non-convex region of the objective function, see Fletcher [33] and Denis and Schnabel [42].)

The BFGS update is the most successful of the quasi-Newton methods. Compared with the DFP method it has some good properties as far as round-off is concerned and is better behaved if the line searches are not exact [26]. It is well-liked for its robustness and for its self-correcting properties, as well as for the super-linear convergence it achieves.

The DFP method has some theoretical properties similar to BFGS and when it was first discovered, it revolutionised the field of non-linear optimization. However it has been found in practice to be less effective than BFGS at self-correcting the Hessian and it can sometimes fail to converge or else stop at saddle points. Through studying the eigenvalues of the Hessian matrix, Powell [57] found that the DFP algorithm can be highly inefficient at correcting erroneously large eigenvalues of matrices, and this is probably one reason why it does not perform as well as the BFGS and other methods. More analysis on this update can be found in Ding [28].

The SR1 formula has one major drawback that it may not preserve positive definiteness of the Hessian approximation even if $\delta_k^T \gamma_k > 0$. However it has been used successfully in the context of trust region methods (with some safeguards) when it is not essential to have the Hessian approximation positive definite. Indeed there is some evidence, see Conn, Gould and Toint [24], that the matrices B_k can sometimes converge more rapidly to $\nabla^2 f$ when the SR1 update is used.

5.2 Modifying Nimp1 to use an approximate Hessian

In this chapter we want to investigate a quasi-Newton method that is based on (5.1) rather than (5.3). That is, we consider an algorithm whose correction step is of the form

$$\delta_k = -(\mu I + B_k)^{-1} g_k \tag{5.9}$$

where B_k is an approximation to the Hessian calculated by methods similar to those outlined in section 5.1.

Conventional quasi-Newton methods based on (5.4) depend upon B_k being positive definite on each iteration, otherwise δ_k may not be a descent step. This means that the BFGS or DFP updates can only be applied when $\delta_k^T \gamma_k$ is positive. Whenever this condition does not hold the Hessian approximation will not be updated; and hence it is possible that several successive iterations may go by without any new second derivative information being incorporated into B_k. If the SR1 update is used then it cannot be guaranteed that B_{k+1} will be positive definite even if $\delta_k^T \gamma_k > 0$.

Bearing the above remarks in mind we now consider whether there could be some benefit in updating B_k on every iteration (rather than sometimes leaving it unchanged) even if this results in the Hessian approximation losing positive-definiteness. Just as Nimp1 is able to use (5.1) to make progress whether or not the exact Hessian is positive-definite so we can use correction (5.9) in a quasi-Newton context to yield a descent step whether or not the matrix B_k is positive definite. (Indefiniteness of B_k need not of course imply that the k-th iteration is in a non-convex region of the objective function; it only means that the current quadratic model is non-convex.)

5.3 A Quasi-Newton Algorithm involving a non-convex quadratic model

We now propose an algorithm which uses correction step (5.9) and which updates B_k on every iteration without attempting to ensure positive definiteness is retained. Such an algorithm is particularly suitable for use with the SR1 update which can allow B_k to become indefinite even when the simple test condition $\delta_k^T \gamma_k > 0$ is satisfied.

It is important to say that, at this stage, we make no claim for the convergence properties of such an algorithm. We shall simply implement and test it in order to look for experimental evidence which might shed light on two important ques-

tions:
(a) is there any practical advantage (particularly when minimizing non-convex functions) in updating B_k on every iteration even if this causes our quasi-Newton Hessian estimate to lose positive-definiteness?
(b) if a quasi-Newton Hessian estimate becomes indefinite can we expect it to regain positive-definiteness when standard updates like BFGS are used in the convex region round the solution?

Question (a) might be judged to have been adequately answered if our algorithm based on (5.9) can outperform a conventional quasi-Newton method on a suitably large number of examples. More importantly, a negative answer to question (b) would make it unlikely that we could claim similar convergence results to those established for existing quasi-Newton methods. Properly to establish the answer 'yes' to question (b) would involve theoretical arguments; but our purpose in this chapter is *only* to see if there is any computational evidence to that would encourage the belief that such arguments exist.

The top-level outline of the algorithm is as follows

Algorithm 5.3.1 *(a quasi-Newton CSDP technique)*

Step 1 *Set $k = 0$, and make an initial guess x_0 of the minimum*
Initialize $B_0 = I$, the $n \times n$ identity matrix (or some other positive definite choice)

Step 2 *for $k = 0, 1, 2, 3...$ until convergence*
Calculate a correction δ_k from (5.9) using a curvilinear search in terms of μ so that $x_{k+1} = x_k + \delta_k$ produces a satisfactory reduction in the objective function f.

Step 3 *If convergence has not occurred, use δ_k and $\gamma_k = g_{k+1} - g_k$ to generate an updated matrix B_{k+1}*

In the next section we shall say more about how the curvilinear search is performed in Step 2 and about the updating formulae that we can use in Step 3.

5.4 The curvilinear step and matrix updating strategies for Algorithm 5.3.1

The curvilinear search algorithm below is very similar to that used in Nimp1 and it involves the calculation of the eigensystem of the approximate Hessian B_k. We are not however assuming that the eigensystem of B_k tells us anything about the eigensystem of the true Hessian. We perform this calculation partly to simplify the repeated calculations (5.9) for varying μ and partly in order to obtain a suitable initial value of μ. We recognize that this is probably an expensive and inefficient approach! We justify its use here on the grounds that the algorithm we are considering is merely a test-bed for exploring the effectiveness of allowing Hessian approximations to become indefinite.

We have given reasons why we believe the quasi-Newton approach described in this chapter is worth investigation; but for completeness, we should also mention other ways in which use of explicit second derivatives can be avoided. For instance, a finite-difference approximation can replace the exact Hessian within the framework of Nimp1; and this should perform well provided the finite-difference estimates are accurate enough. On the other hand it is also possible to generate a descent direction using first-order methods that do not involve the quasi-Newton Hessian approximation, or its expensively computed eigenvalues. In this chapter we choose not to consider such first-order techniques but rather to pursue the quasi-Newton approach in order to maintain consistency with the ideas introduced in previous chapters. We shall, however, propose an updating strategy which makes it easy to compute the eigensystem of B_k. (We do, in fact, consider first-order methods later in this thesis when we discuss a matrix-free approach for constructing an approximate CSDP see chapter 7.)

Algorithm 5.4.1 *(Curvilinear search within Algorithm 5.3.1)*

Step 1 *Choose parameters* $\alpha > 1$, β *and* $\gamma < 1$, D_1^{min} *and* D_i^{max} *for* $i = 1, 2, 3$

Step 2 *Determine the eigenvalues of* B_k *as* $\lambda_i, ..., \lambda_n$ *in descending order.*
If $\lambda_n > 0$ *set* $\mu_{min} = 0$*; otherwise* $\mu_{min} = -\lambda_n$
Set $\mu = 0$ *if* $\lambda_n > 0$*;*

otherwise set $\mu = max(\alpha\mu_{min},\ ||g_k||/\delta_k - \lambda_n)$ *as in section 4.1.1*

Step 3 *Compute* p *by solving* $(\mu I + B_k)p = -g_k$ *and hence calculate* $x^+ = x_k + p, f^+, g^+$ *and the test ratios* $D_i, i = 1, 2, 3$ *using (3.9)-(3.11)*

Step 4 *If* $D_1 < D_1^{min}$ *set* $\mu = \mu + \gamma(\mu - \mu_{min})$ *(to interpolate) and return to step 3*

Step 5 *If* $D_1 > D_1^{max}$ *and EITHER* $\lambda_n > 0$ *or* $(\lambda_n < 0$ *and* $|1 - D_i| < D_i^{max}, i = 2, 3)$ *set* $\mu = \mu - \beta(\mu - \mu_{min})$ *(to extrapolate) return to step 3*

Step 6 *Otherwise* $x_k + p$ *is acceptable and the search is complete.*

We shall implement Algorithm 5.3.1/ 5.4.1 using the DFP, BFGS and SR1 updating formulae described in Section 5.2; and we shall denote these implementations by the names DFPimp, BFGSimp and SR1imp. One drawback of these three methods is that they must go to the expense of computing the eigensystem of an approximate Hessian. This expense can be much reduced if B_k is forced to have some special form for which eigenvalues are easy to obtain for instance, to be a diagonal matrix. Prescribing the form of B_k in this way is likely to restrict the accuracy of the Hessian approximation but in the context of Algorithm 5.4.1 it may have the compensating advantage of significantly reducing computational costs. We can obtain a diagonal form Hessian approximation to satisfy the secant condition (5.5) by setting

$$B_{ii} = \frac{1}{x_{k+1,i} - x_{k,i}} \left[\frac{\partial f(x_{k+1})}{\partial x_i} - \frac{\partial f(x_k)}{\partial x_i} \right] \quad \text{for} \quad i = 1, ..., n. \tag{5.10}$$

When (5.10) is used in algorithms 5.3.1/ 5.4.1 we denote it by **ApproxHess**. As a further comparison we denote by **DiagHess** the use of an approximate Hessian containing only the diagonal terms of the true Hessian.

5.5 Numerical results on test problem $T1$

Consider the function,

$$f(x_1, x_2) = x_1 x_2 + (x_1^2 + 2x_2^2 - 10)^2/100,$$

with the initial guess $x_0 = (2.05, 1.6)^T$. We use the CSDP algorithm parameters

$$\alpha = 2, \beta = 0.75, \gamma = 0.5, D_1^{min} = 0.1, D_1^{max} = 0.6, D_2^{max} = 0.1 \text{ and } D_3^{max} = 0.75.$$

The initial Hessian $B_0 = I$ and initial step $p_{max} = 1$, after which it will be defined as $p_{max} = norm(x_{k+1} - x_k)$.

The following Table 5.1 gives the results obtained by applying the approach described in section 5.3 and 5.4. All methods managed to solve this problem successfully to an accuracy of $\|g\|_2 < 10^{-6}$. On comparing BFGSimp, DFPimp, SR1imp, DiagHess and ApproxHess with the MATLAB toolbox quasi-Newton method QN we observe that SR1imp and BFGSimp do quite well as regards number of iterations. We also observe that DiagHess, DFPimp and ApproxHess are a little less successful.

Method	No of Its	No of fcn calls
DiagHess	15	19
ApproxHess	12	24
BFGSimp	11	22
DFPimp	14	34
SR1imp	11	34
QN	12	19

Table 5.1: Results on problem T1 from DiagHess, Approxhess, BFGSimp, DFPimp, SR1imp, QN

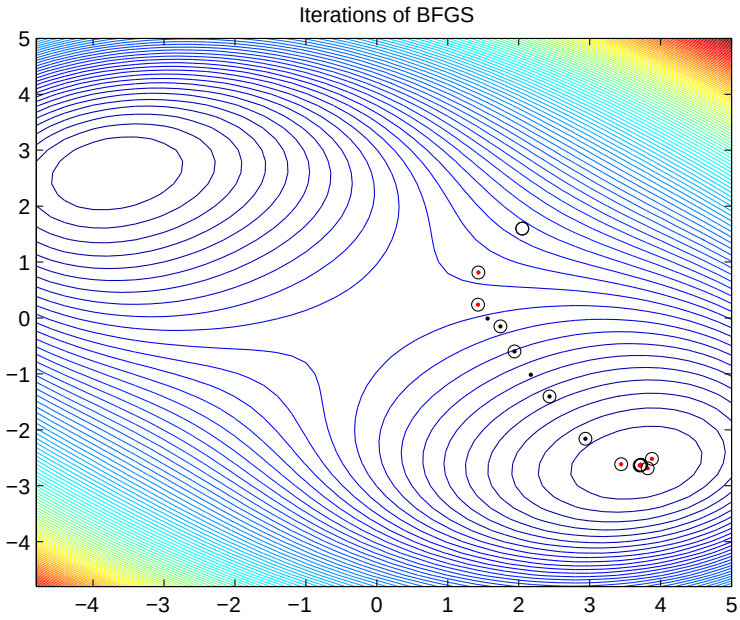

Figure 5.1: Solution path for problem T1 by BFGSimp from $x_0 = (2.05, 1.6)^T$

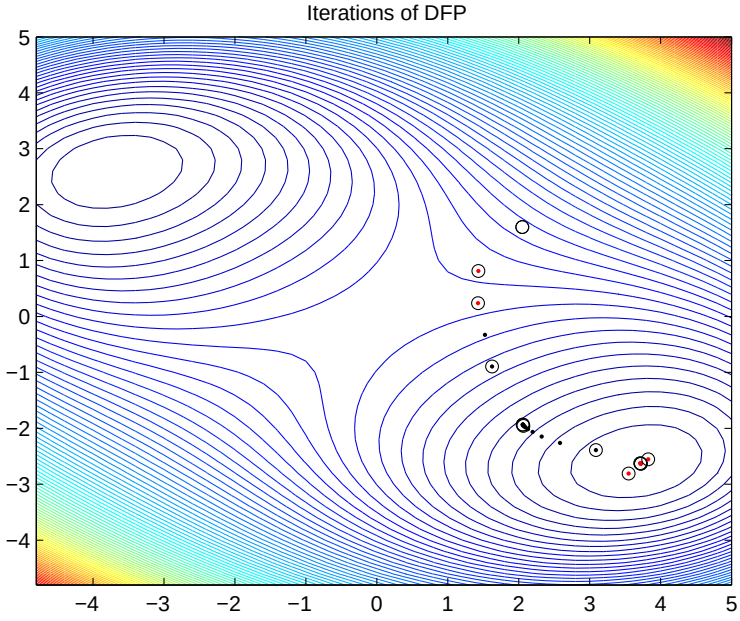

Figure 5.2: Solution path for problem T1 by DFPim from $x_0 = (2.05, 1.6)^T$

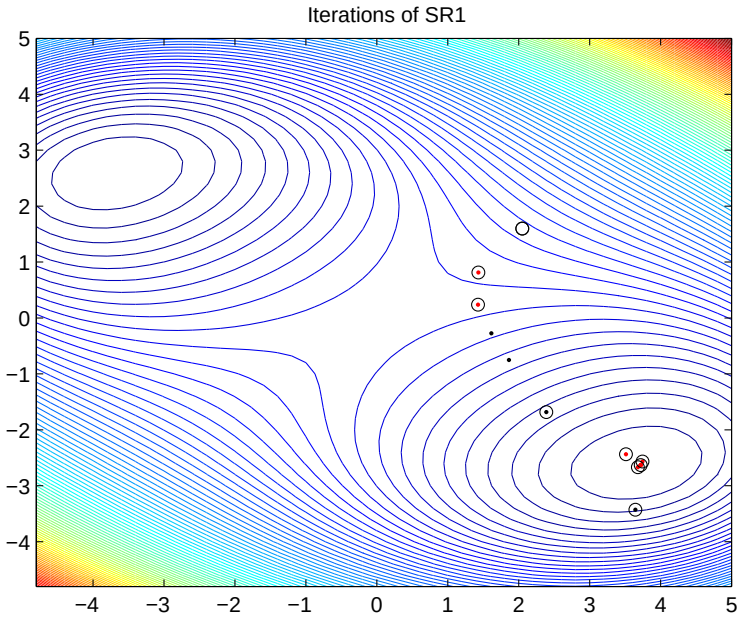

Figure 5.3: Solution path for problem T1 by SR1imp from $x_0 = (2.05, 1.6)^T$

It is clear from the figures 5.1–5.3 that the solution follows a curved path, which differs for each algorithm. Different curvilinear search parameter values could further change the paths for each of the methods, as well as the number of iterations and function evaluations.

5.6 Numerical results on specially constructed test problems

Next we show results using test problems from section 4.4. On some of the problems the performance of SR1imp and BFGSimp is comparable with the standard quasi-Newton approach in QN. However the behaviour of DFPimp and diagonalised Hessian approximations can sometimes be markedly inferior. Generally speaking therefore, it appears from this test set that following a CSDP is less successful when an approximate Hessian is used. If second derivatives are not available then a standard quasi-Newton method seems to do better.

Fctns	DiagHess	BFGSimp	DFPimp	SR1imp	ApproxHess	QN
	Its/fcs	Its/fcs	Its/fcs	Its/fcs	Its/fcs	Its/fcs
T1	15/19	**11/22**	14/34	**11/34**	12/24	*12/19*
T1r	*15/19*	21/71	2812/16727	15/36	15/32	**11/28**
T1r2	*14/21*	26/98	**	16/44	13/36	**11/32**
T1a	15/16	**10/21**	12/30	**10/15**	19/43	***10/13***
T1b	*16/20*	14/35	123/580	**9/21**	22/47	10/22
T1ar	19/27	19/82	**	**10/25**	15/39	11/31
T2	40/58	***13/26***	13/31	14/30	41/87	14/27
T2r	1/4*	13/42	4801/32192	15/49	44/110	***12/23***
T3	*16/24*	15/25	23/66	16/38	25/49	**14/27**
T4_2	66/108	12/29	21/71	11/29	70/184	***7/18***
T4_3	268/361	18/50	87/379	16/49	100/219	***10/24***
T4_4	182/268	26/101	155/695	18/49	674/1763	***17/30***
T4_10	385/500	39/125	346/1760	26/60	130/375	***16/31***
T4_20	493/632	55/184	1253/7760	28/73	51/167	**17/44**
T4r_20	251/318	84/375	**	***1/10***	122/426	14/41
T5	11/21	9/18	9/17	12/19	8/14	***6/11***
T5a	16/31	10/16	11/24	10/23	14/27	***9/14***

Table 5.2: Results for special test problems by DiagHess, BFGSimp, DFPimp, SR1Imp and ApproxHess

5.7 Numerical results using the CUTEr test problems

In this section, we describe the results obtained on the CUTEr test problems. All problems are solved to an accuracy of $\|g\|_2 < 10^{-6}$. Appendix B gives the full results.

We use Dolan-Moré performance profiles [30] to compare the methods on the CUTEr test set. Figures 5.4 and 5.5 relate to iterations as the performance metric while Figures 5.6 and 5.7 are based on numbers of function evaluations.

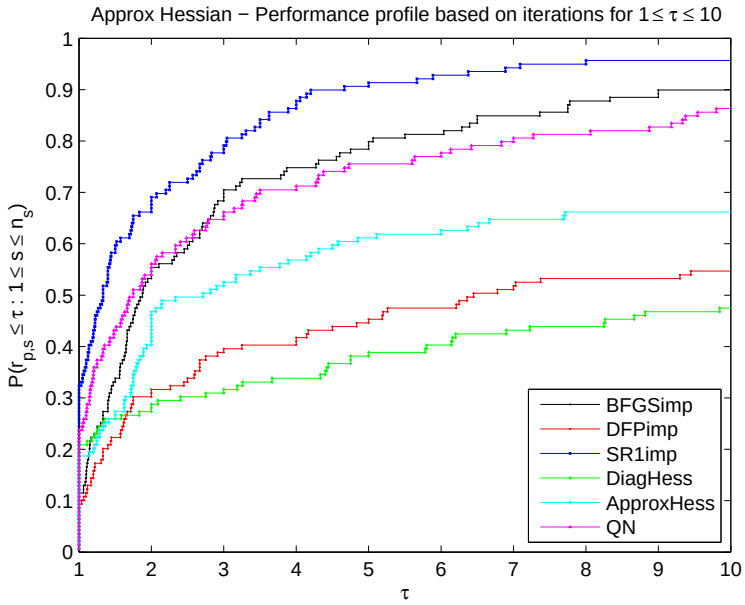

Figure 5.4: Approx Hessian: Comparison of the methods based on iterations for $1 \leq \tau \leq 10$.

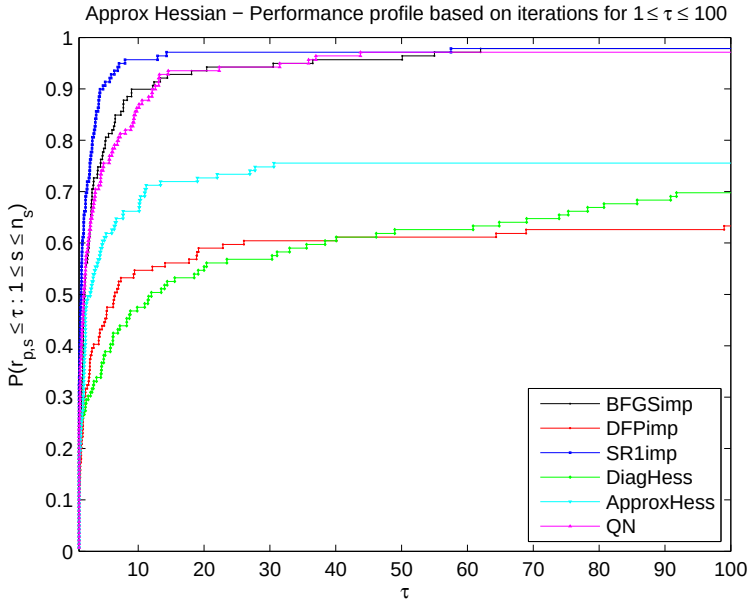

Figure 5.5: Approx Hessian: Comparison of the methods based on iterations for $1 \leq \tau \leq 100$.

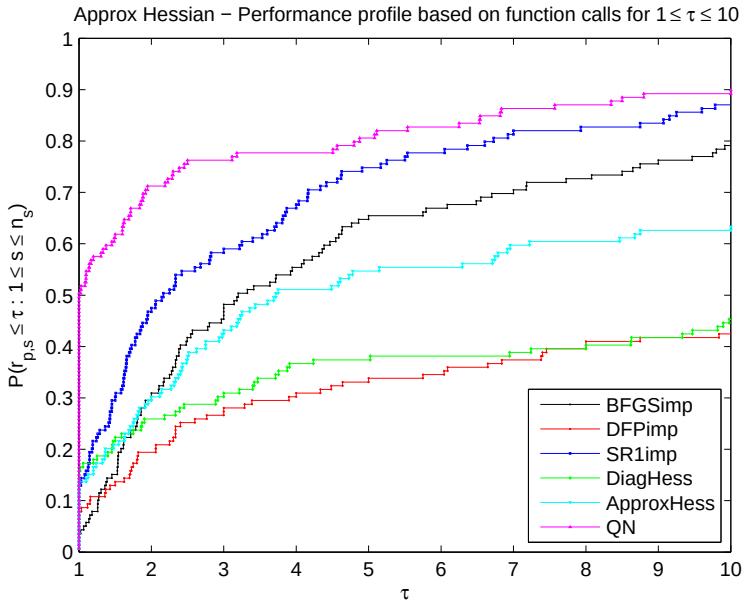

Figure 5.6: Approx Hessian: Comparison of the methods based on function calls for $1 \leq \tau \leq 10$.

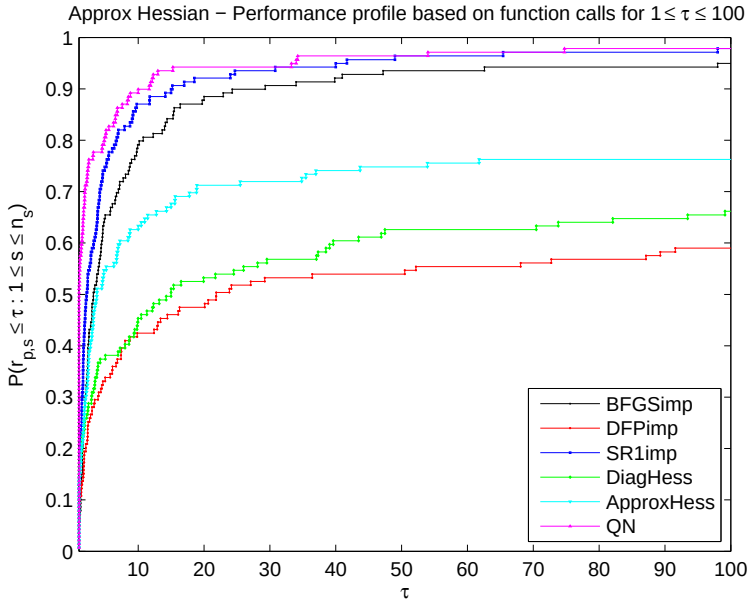

Figure 5.7: Approx Hessian: Comparison of the methods based on function calls for $1 \leq \tau \leq 100$.

These figures show that DFPimp, DiagHess and ApproxHess are always inferior to BFGSimp, SR1imp and QN. This is consistent with what we have already said about properties of the DFP update. (We shall leave our comments on the Diagonal approximations until a little later in this section.)

It is interesting to note that, for small and moderate values of τ, SR1imp can outperform both BFGSimp and QN, when iteration count is the measure of performance. For $\tau > 40$, however, this advantage is lost and all three methods behave in a very similar way.

When the methods are compared in terms of function calls QN is consistently the best, which suggests that the conventional line search is more economical than the curvilinear search used in CSDP methods.

It is worth noting that DiagHess did really well on a number of problems - especially those with a sparse Hessian such as DIXMAAN*. We also observe that ApproxHess does quite well on the same problems when compared with QN. However, it also has a large number of failures. This may be due to the steps becoming very small near the minimum. To some extent this is to be expected since B_k is obtained (5.10) which may become close to a " $0/0$" calculation.

It must of course be accepted that while DiagHess and ApproxHess may give useful Hessian approximations when the search is far from a solution they cannot in general be expected to do well as the BFGS update in the convex neighbourhood of a minimum.

5.8 Conclusion

We have carried out an experimental investigation of the benefits of following a CSDP via the implicit Euler method based on an approximation to the Hessian matrix. This means the correction steps are based on solving

$$(\mu I + B)p = -g$$

for a range of values for μ where B is a (not necessary positive definite) approximation to the true Hessian matrix. This is done in the same way as in Nimp1, i.e. by getting the eigenvalues of B by an RDR^T decomposition. The aim, as before, is to make good and efficient progress along the curved path $p(\mu)$.

We have shown that calculating the matrix B by the popular BFGS and SR1 quasi-Newton updates can sometimes be quite successful. However competitiveness in terms of function calls still seems to depend strongly on the choice of parameters in the curvilinear search. It is worth noting however that, although our methods are rather inconsistent, there are cases where they really do well when QN struggles to get the minimum.

Our main conclusion based on closer observation of the results is that BFGSimp and SR1imp can do quite well far from the solution but are not particularly efficient in the convex neighbourhood around the solution. But this is the region where a standard quasi-Newton method can be expected to do well; and hence it could be worth exploring a two-part strategy which uses a CSDP approach based on (5.9) when far from a solution but reverts to a more conventional search direction/line search method when close to the solution. This idea will be explored in the next chapter, where it will be implemented in such a way that we do not rely on any assumption that an updated Hessian approximation will automatically regain positive definiteness in the convex neighbourhood of a minimum. (In other words we shall not need to seek an answer to the interesting question (b) from section 5.3.)

Chapter 6

Hybrid Methods

6.1 Introduction

Hybrid methods seek to combine the best features of two distinct approaches in order to solve certain problems more quickly. They are well known in optimization and many authors use them. For example, Powell [56] used a hybrid method for nonlinear equations, Hald and Madsen [39] combined LP and quasiNewton methods for minmax optimization, Al-Baali and Fletcher [2] used a hybrid method in variational methods for nonlinear least squares, Morales and Nocedal [49] developed hybrid methods for large-scale unconstrained optimization that combine iterations of the limited-memory BFGS method (L-BFGS) and the Hessian free Newton method and more recently Mohr [47] developed a hybrid Runge-Kutta and quasi-Newton Method for unconstrained nonlinear optimisations.

The goal in this Chapter is to show that different approaches within one algorithm can be better suited for problems that have mixed convex and non-convex region, and that hybrid techniques which include mechanisms from different algorithms can benefit from their advantages. Ideally one of our methods is to be used in the non-convex region while an other in the convex region, where there are many well known techniques. We want the information gathered by one method to help the performance of the other without increasing the computational cost.

6.2 Hybrid method using the exact Hessian

We have seen in our initial experiments of Chapters 3 - 4, that there is an economical and logical way to minimise non-convex functions by approximately following the CSDP through the non-convex-regions. Up till now we have continued to use the same iterative scheme in convex regions as well as non-convex. But the idea was suggested in Chapter 5 that it might be more effective to use more conventional iterative schemes in a convex neighbourhood of the minimum. We, therefore, propose a new hybrid algorithm and we have chosen to use *Nimp1* as the basis for our investigation, combining it with another method that also uses the full Hessian namely the Newton method. The latter is chosen as it well-studied, and is well-known for its good performance when the starting point is chosen appropriately. We call the hybrid algorithm by **HNNimp1**. This hybrid method could be applicable to minimise any function regardless of whether it is convex or not and would guarantee to find a point which is a minimal of such function. Numerical results show that this method is effective in practice.

The particular interest given to the Newton method in the convex region is due to the fact it is computationally efficient. The method itself uses successive quadratic approximations to the objective function $f(x)$ based on the Taylor series expansion about x_k,

$$f(x_k + p) \approx f(x_k) + g_k^T p + \frac{1}{2} p^T G_k p \tag{6.1}$$

where $g_k = \nabla f(x_k)$ and $G_k = \nabla^2 f(x_k)$. When the gradient of this quadratic approximation of $f(x)$ is zero, we can form a sequence of iterations $x_{k+1} = x_k + p_k$, based on finding p_k to satisfy

$$g_k + G_k p_k = 0 \tag{6.2}$$

or

$$p_k = -G_k^{-1} g_k \tag{6.3}$$

Thus the following iterative scheme defines produce the Newton method

$$x_{k+1} = x_k - G_k^{-1} g_k.$$

Note in this case the direction (6.3) is a descent direction as we are dealing with functions in the convex region and so the Hessian is always positive definite i.e. the descent direction condition

$$p_k^T g_k < 0, \tag{6.4}$$

is always satisfied.

If the Hessian matrix G_k is positive definite, Newton's method will minimize a quadratic function, from any starting point, in exactly one step. Therefore, we expect good convergence from the method when the quadratic model (6.1) is accurate. Fletcher [33] has shown, under mild regularity conditions on $f(x)$, that Newton's method exhibits a quadratic rate of convergence.

6.3 Hybrid algorithm using the exact Hessian

An iteration of a hybrid algorithm HNNimp1 is as follows. The overall algorithm and some details of the iteration are not given since they are the same as the algorithm Nimp1 described previously.

Algorithm 6.3.1 *(Algorithm with exact Hessian)*
Given the parameters are: $\alpha > 1$, $\beta < 1$, $\gamma < 1$, D_1^{min}, D_1^{max}, D_2^{max} *and* D_3^{max}.
If the Hessian G_k *is positive definite then*

- *Compute the search direction* $p_k = -G_k^{-1} g_k$
- *Determine step size* h_k *to ensure* $f(x_k + h_k p_k) < f(x_k)$ *by means of bracketing say.*
- *Set* $x_{k+1} = x_k + h_k p_k$

else

- *Set* $\mu = maximum\left(\alpha\mu_{min}, \frac{\|g_k\|_2}{\delta_k} - \lambda_{min}\right)$
- *Compute* p_k *from (3.5) and hence get* $x_k + p_k$, f_{k+1}, g_{k+1}, D_1, D_2, D_3, *using defintions (3.9) - (3.11).*

– *If* $D_1 < D_1^{min}$ *set* $\mu = \mu + \gamma(\mu - \mu_{min})$ *(to interpolate)*
– *If* $D_1 > D_1^{max}$ *and* $D_2 < D_2^{max}$ *and* $|1 - D_3| < D_3^{max}$
set $\mu = \mu - \beta(\mu - \mu_{min})$ *(to extrapolate)*
– *Otherwise* $x_k + p_k$ *is acceptable and the iteration is compete.*

end

This algorithm performs a curvilinear search along an approximate CSDP whenever the standard Newton iteration cannot be used because of negative curvature of the objective function.

6.4 Approximating the Hessian in the non-convex region

In the previous section we based our calculations on using the exact Hessian. We could of course also propose a hybrid approach based on on the use of the Hessian approximations as in Chapter 5. In particular, we can use the *BFGS* update in the convex region and a diagonal approximation to the Hessian matrix $G(x_k)$ in the non-convex region as in Chapter 5. We suppose that from the point x_k we take the step to x_{k+1} given by (3.1). We then evaluate $g(x_{k+1})$ and set

$$D_{ii} = \frac{1}{x_{k+1,i} - x_{k,i}} \left[\frac{\partial f(x_{k+1})}{\partial x_i} - \frac{\partial f(x_k)}{\partial x_i} \right]$$

where D is treated as a diagonal estimate of $G(x_k)$.

When x_k is in the non-convex region, the calculation of x_{k+1} from (3.3) then becomes

$$x_{k+1,i} = x_{k,i} - h \frac{\partial f(x_k)}{\partial x_i} \left[\frac{1}{1 + hD_{ii}} \right]$$

By increasing h from zero we can trace out an approximation to the CSDP which begins at x_k. This rather simple Hessian approximation may be of use in non-convex regions where a more conventional low-rank update would not normally be appropriate.

6.5 Outline of a composite algorithm with diagonal Hessian

We can now describe a minimisation algorithm HDBFGS based on the idea sketched above.

Algorithm 6.5.1 *(Algorithm with diagonal Hessian)*

Step 1 *Given x_0 and a positive definite matrix H_0, calculate $p_0 = -H_0 g_0$*

Step 2 *Find h so that $x_1 = x_0 + hp_0$ and $f(x_1) < f(x_0)$.*

Step 3 *Set $\delta = x_1 - x_0$, $\gamma = g_1 - g_0$*

Step 4 **For** $k = 1, 2, ...$

If $\delta^T \gamma > 0$ then

- *Use a* BFGS *update to obtain H_k from H_{k-1} and set $p_k = -H_k g_k$*
- *Find h so that $x_{k+1} = x_k + p_k(h)$ and $f(x_{k+1}) < f(x_k)$*
- *Set $\delta = x_{k+1} - x_k$,$\gamma = g_{k+1} - g_k$*

else

- *$H_k = H_{k-1}$ and $D_{ii} = diag\left(\frac{\gamma_i}{\delta_i}\right)$*
- *Perform a curvilinear search in terms of h so that $f(x_k + p_k) < f(x_k)$, where*

$$p_{k,i} = -h g_{k,i} \left[\frac{1}{1 + hD_{ii}}\right], \qquad i = 1, ..., n \tag{6.5}$$

- *Then set $x_{k+1} = x_k + p_k$, $\delta = x_{k+1} - x_k$, $\gamma = g_{k+1} - g_k$*

end if

Step 5 **until** $\|g_{k+1}\|$ *is sufficiently small.*

This algorithm performs a curvilinear search along an approximate CSDP whenever a standard quasi-Newton iteration cannot be used because of negative curvature of the objective function. Ways of adjusting the value of h in (6.5) will be discussed later.

An important feature of HDBFGS is that, in contrast to BFGSimp, it does not need to do any eigenvalue analysis. In the non-convex region it uses a diagonal matrix whose eigensystem is obtained trivially for use in (6.5). Non-convexity of the objective function is deduced without any reference to the indefiniteness of

an approximate Hessian. Whenever an iteration has given $\delta_k^T \gamma_k < 0$, the next iteration is a Nimp-like one using a locally constructed diagonal Hessian estimate; and, on the other hand, when $\delta_k^T \gamma_k > 0$ the current positive definite Hessian approximation is updated and a standard quasi-Newton step is taken.

The ultimate convergence of the algorithm will conform to established theory regarding quasi-Newton methods, since the condition $\delta^T \gamma > 0$ will hold for all k sufficiently large -i.e. once the search has reached the convex region surrounding a local minimum. It is worth mentioning here that it can be beneficial to redefine H_{k-1} if the k-th iteration is one on which $\delta^T \gamma$ returns to being positive after several steps with negative curvature. Specifically, if $\gamma_i \delta_i > 0$ for $i = 1, ..., n$ we can set

$$H_{k-1} = \text{diag}\left(\frac{\delta_i}{\gamma_i}\right). \tag{6.6}$$

However the condition $\delta^T \gamma > 0$ does not ensure that the individual product $\delta_i \gamma_i$ are all positive and so we cannot always use (6.6) to create a positive-definite H_{k-1} based on up-to-date information.

6.6 Step length control

Our aim is to determine h in (6.5) to ensure that p_k is a descent step which produces an acceptable reduction in the objective function. We can imitate the Wolfe condition [4] for a conventional line search -i.e., we want h to imply that $f(x_k + p_k) - f(x_k)$ is negative and also that both $|f(x_k + p_k) - f(x_k)|$ and $\|p_k\|$ are bounded away from zero by a multiple of $\|g_k\|$. This might be done by comparing the actual change in f with a first or second order predicted change.
Let us define

$$\Delta_a = f(x_k + p_k) - f(x_k), \quad \Delta_{p_1} = p_k^T g_k, \quad \Delta_{p_2} = p_k^T g_k + \frac{1}{2} p_k^T D p_k$$

$$\rho_1 = \frac{\Delta_a}{\Delta_{p_1}}, \quad \rho_2 = \frac{\Delta_a}{\Delta_{p_2}}$$

When $\rho_1 < 0$ we can conclude that h is too big. This mean that either the step along the continuous gradient is sufficiently large that it has left the non-convex region and crossed a valley *or* that the direction p_k is uphill. A value of $\rho_1 \approx 1$ suggests that the step p_k is too small and that a larger value of h would produce a step giving a bigger reduction in f.

If $\rho_2 \approx 1$ the actual behaviour of f is in good agreement with the approximating quadratic model. This indicates that our approximation to a point on the steepest descent trajectory is a good one which also indicates that we have an acceptable reduction in f.

6.7 Choosing the step h so (6.5) gives a descent direction

Let us assume that the diagonal elements of D, the diagonal approximation to $G(x_k)$, are rewritten as $\lambda_1, ..., \lambda_n$ where

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n.$$

In the outline algorithm from the previous section, we will only use (6.5) if there is at least one negative element D_{ii}. Hence we can assume that $\lambda_n < 0$. From (6.5), each element $p_{k,i}$ is of the form $-\mu_{k,i} g_{k,i}$ where

$$\mu_{k,i} = h \left[\frac{1}{1 + hD_{ii}} \right]$$

and $\mu_{k,i}$ is *positive* so long as

$$0 < h < h_{max} = -\frac{1}{\lambda_n}.$$

Thus, if $h < h_{max}$ then

$$p_k^T g_k = -\sum_{i=1}^{n} \mu_{k,i} g_{k,i}^2 < 0$$

and so p_k is a descent direction.

6.8 An initial trial value for the step h

In order to perform a search along the CSDP it will be helpful to have a good initial choice for h. One way of doing this is to estimate a value for h which will give a step p_k which is comparable in size to the last successful step δ. When $0 < h < h_{max}$,

$$|p_{k,i}| \leq h \left| \frac{1}{1+\lambda_n h} \right| |g_{k,i}|.$$

Hence

$$\frac{\|p_k\|_2}{\|\delta\|_2} \leq h \left[\frac{1}{1+\lambda_n h} \right] \frac{\|g_k\|_2}{\|\delta\|_2}.$$

We can choose h so that the right hand side of this inequality is equal to q - i.e. so that the next trial step cannot be more than q times the size of the previous one.

We require

$$h \left[\frac{1}{1+\lambda_n h} \right] = q \frac{\|\delta\|_2}{\|g_k\|_2} = Q \text{ (say)}.$$

Then

$$h = Q(1+\lambda_n h)$$

$$h(1-\lambda_n Q) = Q$$

so that

$$h = \frac{Q}{1-\lambda_n Q} \tag{6.7}$$

6.9 A curvilinear search algorithm

Once we have an initial choice for h a search algorithm can be proposed as follows.

Algorithm 6.9.1 *(Approximate CSDP algorithm)*

Step 1 *Set* $h_{min} = 0$
Repeat

Step 2 *Evaluate* p_k *from (6.5) for the current* h.

Step 3 *Hence compute* $f(x_k + p_k)$ *and* ρ_1, ρ_2.

Step 4 *If* $|1-\rho_1| < \eta_1$ *then* $h_{min} = h$; $h = h + \beta(h_{max} - h)$ *and go to step 2*
else if $\rho_1 < \eta_2$ *then* $h_{max} = h$; $h = h - \beta(h - h_{min})$ *and go to step 2*

else if $\rho_1 > 1 + \eta_1$ *and* $|1 - \rho_2| < \eta_3$ *then* $h_{min} = h$; $h = h + \beta(h_{max} - h)$ *and go to step 2*
else h *is acceptable because* $\rho_1 > 1 + \eta_1$ *or* $1 - \eta_1 > \rho_1 > \eta_2$.

6.10 Numerical results on test problem T1

In this section we report some results obtained with the algorithms outlined in sections 6.3 and 6.5 using the function $T1$ given by

$$f(x_1, x_2) = x_1 x_2 + (x_1^2 + 2x_2^2 - 10)^2/100,$$

with the initial condition $x_0 = (2.05, 1.6)^T$ and the algorithm parameters $\alpha = 2$, $\beta = 0.75$, $\gamma = 0.5$, $D_1^{min} = 0.1$, $D_1^{max} = 0.6$, $D_2^{max} = 0.1$ and $D_3^{max} = 0.75$ as well as with initial Hessian $H_0 = I$ and initial step equal to 1.

Method	No of Its	No of fcn calls
HNNimp1	6	10
HDBFGS	10	15
TR	8	9
QN	12	19

Table 6.1: Results from HNNimp1, HDBFGS

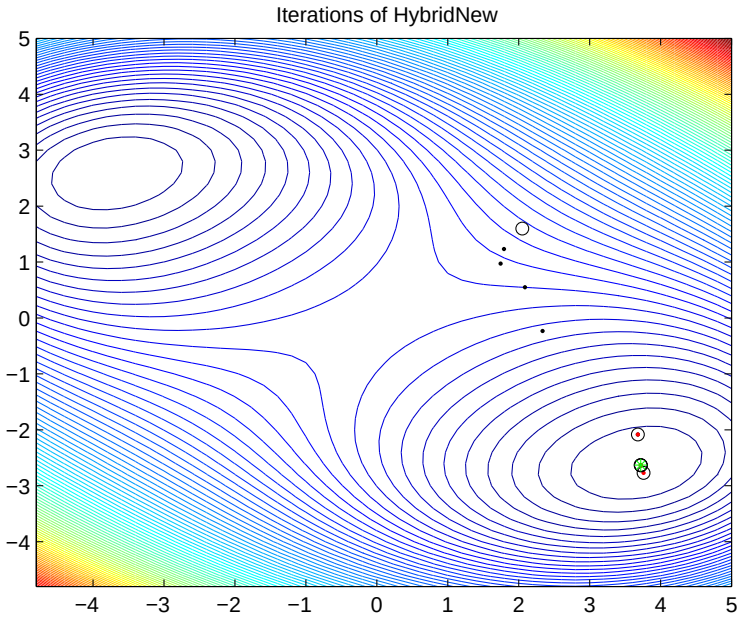

Figure 6.1: Solution path for $T1$ by HNNimp1 from $x_0 = (2.05, 1.6)^T$

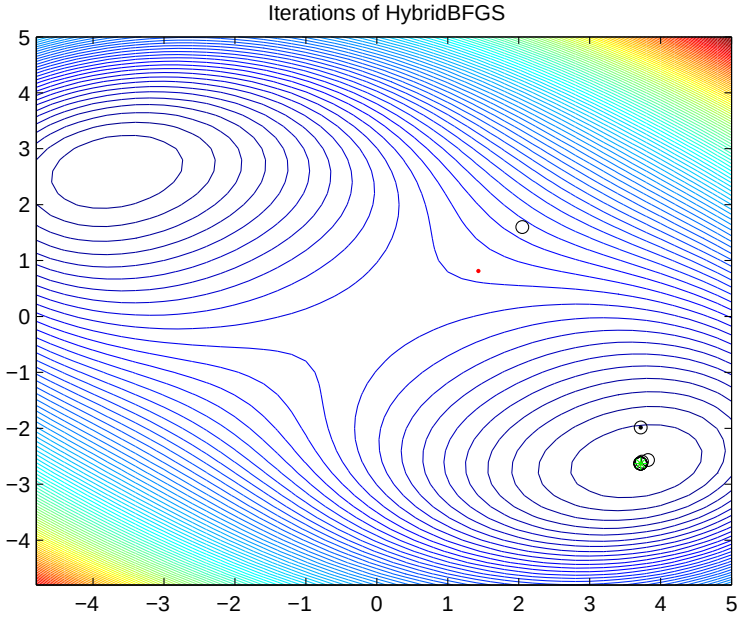

Figure 6.2: Solution path for $T1$ by HDBFGS from $x_0 = (2.05, 1.6)^T$

We can clearly see from the figures that the solution is achieved by both methods. On comparing HNNimp1 and HDBFGS with routines TR and QN from the optimisation toolbox of MATLAB, we see that they do quite well on the number of iterations but there is a difficulty with the search calculation in HDBFGS.

6.11 Numerical results on specially constructed test problems

Fctns	HNNimp1	HDBFGS	Nimp1	BFGSimp
	Its/fcs	Its/fcs	Its/fcs	Its/fcs
T1	6/10	10/15	6/10	11/22
T1r	7/14	18/27	7/14	21/71
T1r2	8/14	18/35	8/14	26/98
T1a	5/10	10/16	5/10	10/21
T1b	7/11	9/15	7/11	14/35
T1ar	8/14	13/22	8/14	19/82
T2	7/11	9/17	8/13	13/26
T2r	6/13	15/26	7/15	13/42
T3	9/17	17/32	9/17	15/25
T4_2	7/8	12/19	7/10	12/29
T4_3	9/10	17/24	8/10	18/50
T4_4	11/13	21/26	12/16	26/101
T4_10	18/21	28/37	15/19	39/125
T4_20	7/10	24/31	9/15	55/184
T4_50	10/12	29/53	10/13	67/244
T4_100	14/16	1/4*	14/17	87/340
T4r_20	7/10	36/52	9/15	84/375
T5	7/12	8/18	7/11	9/18
T5a	9/15	12/25	10/20	10/16

Table 6.2: Results from HNNimp1 and HDBFGS for specially designed test problems

We have tested both hybrid methods HNNimp1 and HDBFGS on test problems described in Appendix D. The results obtained for these problems are given in Table 6.2. We see that there is some improvement of the results when we use

these hybrid techniques. In particular, HNNimp1 generally does slightly better than the original Nimp1.

The method HDBFGS also did well on the number of iterations compared with the techniques discussed in Chapter 5. It was however, slightly less efficient than other algorithms in terms of the number of iteration when compared with the quasi-Newton method QN. It also failed on 4 test problems to locate the minimum. The high number of function evaluations in this method may be due to the fact that we are using backtracking technique in the weak line search associated with the BFGS methods and may be other line search techniques are more suited.

6.12 Numerical results using the CUTEr test problems

In this section, we test the two hybrid methods for on a total of nearly 139 CUTEr test problems that include convex and non-convex problems. The numerical results obtained are given in Appendix C.

For most of the test problems we show that both hybrid methods are efficient and robust in solving these unconstrained optimization problems. The results reported in Appendix C show that in many cases HNNimp1 gives a significant improvement on both Nimp1 and TR algorithms, since these are the two techniques that did really well in the first instance when the full Hessian was used. For instance, it can match or improve the number of iteration needed by Nimp1 in 75% of cases. It beats the trust region method TR in 65% of cases. There have also been cases however, where it did not do as well compared to both this techniques such as on test problems HIELOW and VAREIGVL. HNNimp1 matches or improves on the performance of the all the other methods used in over 45% of cases and with no failure.

Because of the considerable improvement of of the methods achieved by HNNimp1

in this thesis, it is felt that it is worthwhile comparing the performance of HDBFGS to that of a quasi-Newton method on the same test problems from CUTEr, see Appendix C.

We use Dolan-Moré performance profiles [30] to compare the hybrid methods on the CUTEr test set. Figures 6.3 and 6.4 relate to iterations as the performance metric while Figures 6.5 and 6.6 are based on numbers of function evaluations.

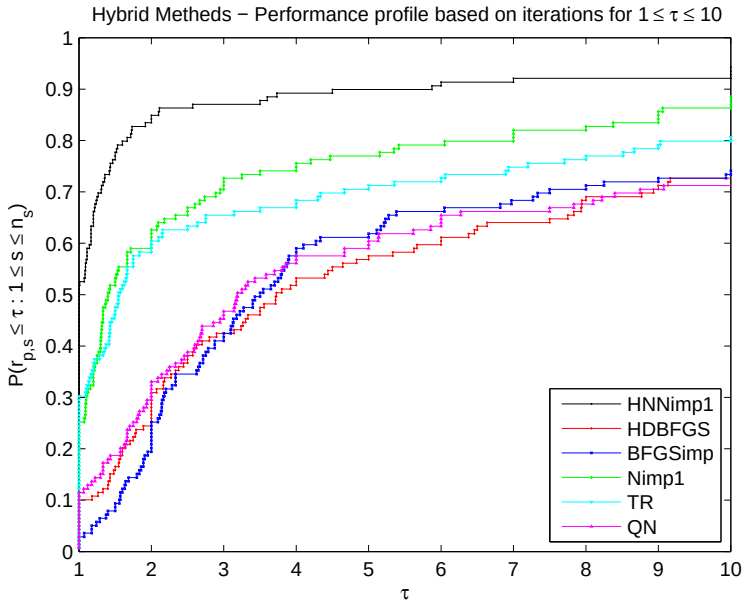

Figure 6.3: Hybrid methods: Comparison of the methods based on iterations for $1 \leq \tau \leq 10$.

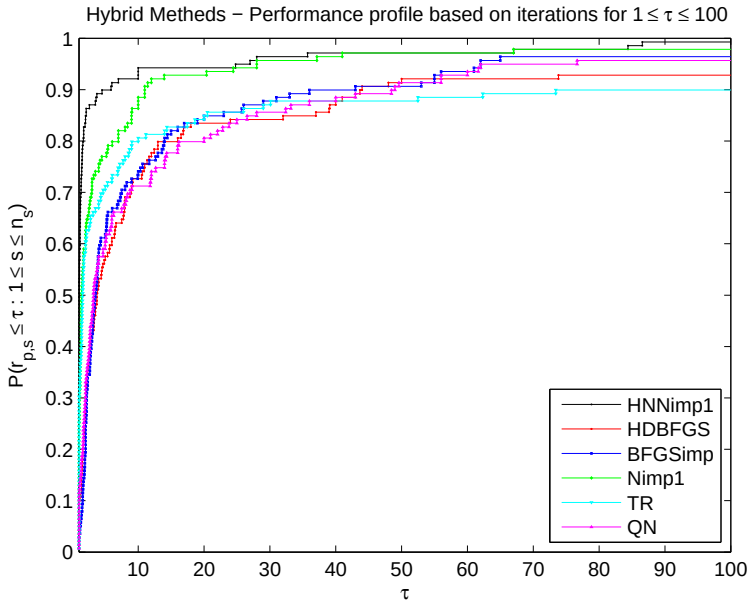

Figure 6.4: Hybrid methods: Comparison of the methods based on iterations for $1 \leq \tau \leq 100$.

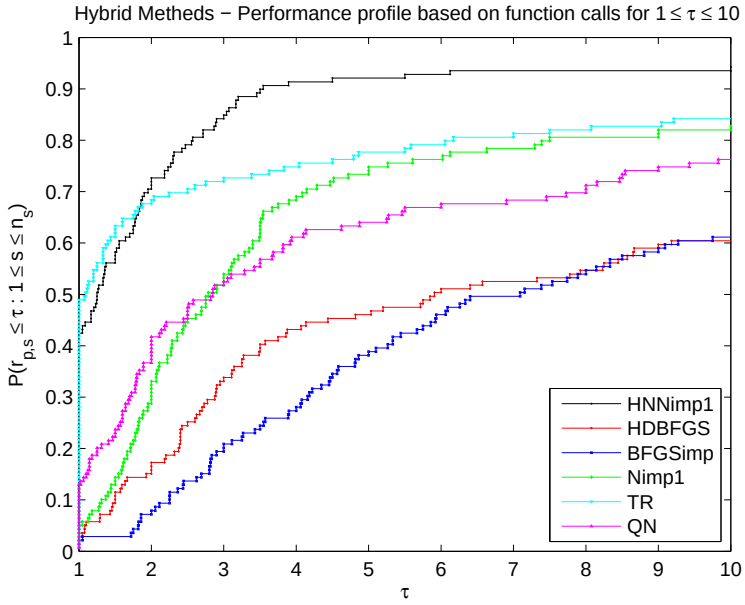

Figure 6.5: Hybrid methods: Comparison of the methods based on function calls for $1 \leq \tau \leq 10$.

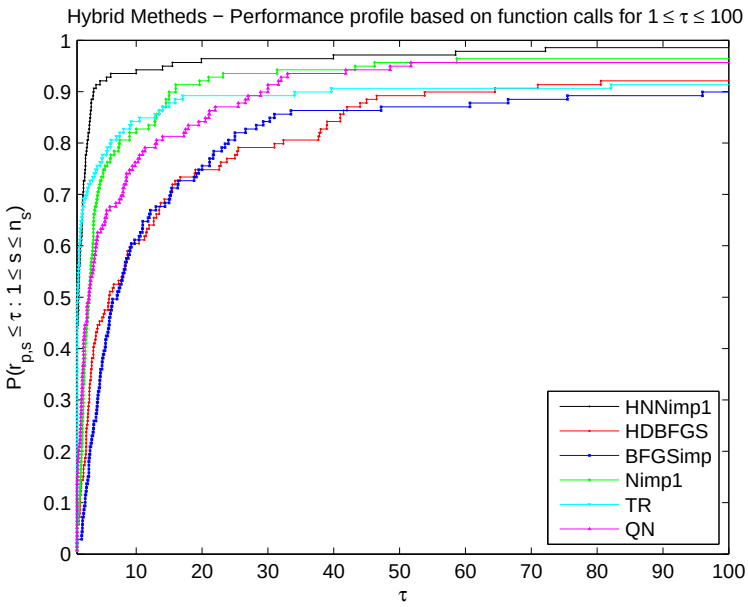

Figure 6.6: Hybrid methods: Comparison of the methods based on function calls for $1 \leq \tau \leq 100$.

Figure 6.3 shows very clearly -as we would expect- that the methods using the exact Hessian do better in terms of iteration count than methods involving Hessian approximations. This is also the case when performance is measured by function calls. The hybrid version HNNimp1 always does better than Nimp1; but it is outperformed by TR in the range $(1 < \tau < 2)$ when function calls are the basis for comparison. In terms of function calls TR also does better than Nimp1 until $\tau \approx 20$; but it is relatively less successful in terms of iteration count.

It is rather disappointing to note that the hybrid form HDBFGS often shows little or no advantage over its counterpart BFGSimp. Even when it does do better (say in the function call comparison with $\tau < 5$) it is inferior to the standard QN algorithm. Hence there is still room for improvement in the non-second derivative methods; and in the next chapter we consider a different approach.

Chapter 7

Methods Using Runge-Kutta Solvers

7.1 Introduction

So far we have based our approximate solutions of the continuous steepest descent equation on the implicit Euler approach. This requires us to calculate (or estimate) the Hessian matrix and then compute its eigensystem. In Chapter 5 we considered the possibility of reducing these expensive overhead calculations by making use of a diagonal form of Hessian approximation. In this Chapter we investigate a much more substantial reduction in overheads resulting from a matrix-free approach which is based on solving the steepest descent equation by an explicit second order Runge Kutta (RK) approach.

In the context of CSDP methods, we observe that we can use the explicit RK scheme to find a step p without using the matrix G. We use two evaluations of the gradient, namely

$$g_0 = g(x) \quad \text{and} \quad g_1 = g(x - \bar{h}g)$$

where $\bar{h}$ is a fixed reference step length. We can calculate a second order accurate

search direction p corresponding to any chosen step length h by setting

$$p = -\alpha_1 h g_0 - \alpha_2 h g_1$$

where

$$\alpha_2 = \frac{\theta}{2} \quad \text{and} \quad \alpha_1 = 1 - \frac{\theta}{2} \quad \text{where} \quad \theta = \frac{h}{\bar{h}}.$$

Substituting for α_1 and α_2 we get

$$p = -\left(h - \frac{h^2}{2\bar{h}}\right) g_0 - \frac{h^2}{2\bar{h}} g_1. \tag{7.1}$$

Hence the sensitivity of p to h is given by

$$p' = -\left(1 - \frac{h}{\bar{h}}\right) g_0 - \frac{h}{\bar{h}} g_1 = -(1-\theta) g_0 - \theta g_1. \tag{7.2}$$

From this idea we can develop an algorithm which would quite cheaply trace out an approximate CSDP.

7.2 Curvilinear search

Suppose we have taken a step from x_k to $x_{k+1} = x_k + p$ where p is obtained by (7.1). A decision on whether x_{k+1} is acceptable and/or how to improve it may involve the values of $f_{k+1} = f(x_{k+1})$ and $g_{k+1} = g(x_{k+1})$. We may also be interested in how f_{k+1} varies with h, the step size in the Runge-Kutta calculation. We let f'_k and f'_{k+1} denote df/dh at the start and the end of the step. The value of f'_{k+1} is not necessarily the same as the directional derivative $p^T g_{k+1}$ since the path traced out by this method will usually not be precisely along the direction $p = x_{k+1} - x_k$.

To estimate f'_{k+1} we can argue as follows. An average slope over the step is

$$\bar{f}' = \frac{f_{k+1} - f_k}{h} \approx \frac{1}{2}[f'_{k+1} + f'_k]$$

and we know that

$$f'_k = -g_k^T g_k$$

because the initial motion away from x_k is along the steepest descent. Therefore

$$f'_{k+1} \approx 2\bar{f}'_k - g_k^T g_k.$$

More accurately however,

$$f'_{k+1} = p_k'^T g_{k+1},$$

which can computed if we have evaluated p' by say (7.2). We may argue that we can extrapolate along the CSDP by increasing h in (7.1) if f'_{k+1} is sufficiently large and negative. Conversely we should interpolate (by reducing h in (7.1)) if f'_{k+1} is sufficiently large and positive or if $f_{k+1} \geq f_k$.

7.3 Outline algorithm

So an outline of one step of a curvilinear search CSDP method based on a Runge-Kutta method is as follows.

Algorithm 7.3.1 *(rk2)*
Step 0 *Given* $\bar{h}$, x_k, $f_k = f(x_k)$, $g_k = g(x_k)$, h_{min}, h_{max}
Step 1 *calculate* $x_1 = x_k - \bar{h}g_k$, $f_1 = f(x_1), g_1 = g(x_1)$
Step 2 *find* h, *calculate* $\theta = h/\bar{h}$; $\alpha_2 = \theta/2$; $\alpha_1 = 1 - \theta/2$;
$p = -\alpha_1 h g_0 - \alpha_2 h g_1$; $x_{k+1} = x_k + p$;
$p' = -(1-\theta)g_0 - \theta g_1$; $f'_{k+1} = p'^T g_{k+1}$;
Step 3 *If* $f_{k+1} \geq f_k$ *or* $f'_{k+1} >$ *some positive tolerance then*
reduce h *and recalculate* p *and* x_{k+1} *via step 2*
Step 4 *if* $f'_{k+1} <$ *some negative tolerance*
increase h *and recalculate* p *and* x_{k+1} *via step 2*
Step 5 *else if* f'_{k+1} *sufficiently close to zero*
accept x_{k+1} *as the end of the current step*

Instead of using f'_{k+1} we could base our algorithm on the angle Θ between p' and the local negative gradient, as a measure of when to stop exploring the curvilinear path. let

$$C = cos(\Theta) = \frac{p'^T g_{k+1}}{\|p'\| \|g_{k+1}\|} \tag{7.3}$$

then we can outline a single step of a CSDP search as

Algorithm 7.3.2 *(rk2a)*

Steps 0, 1, 2 *are as in the previous algorithm 7.3.1 with the extra calculation of C from (7.3)*

Step 3 *If $f_{k+1} \geq f$ or $C >$ some specified tolerance then reduce h and recalculate p and x_{k+1} via step 2*

Step 4 *if $C <$ some specified tolerance then increase h and recalculate p and x_{k+1} via step 2*

Step 5 *else if C lies between specified upper and lower tolerances then accept x_{k+1} as the end of the current step.*

7.4 Choosing and adjusting the step $\bar{h}$ and h

The effectiveness of a search along p given by (7.1) will clearly depend on both h and $\bar{h}$. In particular the initial choices of h and $\bar{h}$ will determine whether p is even a descent direction. Using (7.1) we have

$$p^T g_0 = -h\left(1 - \frac{\theta}{2}\right) g_0^T g_0 - h\frac{\theta}{2} g_0^T g_1$$

Hence p is a descent direction if

$$\left(\frac{\theta}{2}\right)\left(g_0^T g_0 - g_0^T g_1\right) - g_0^T g_0 < 0$$

If $g_0^T g_0 - g_0^T g_1 < 0$ then the descent condition is satisfied for all positive θ i.e. for all positive values of h. If however $g_0^T g_0 - g_0^T g_1 > 0$ then descent depends on the restriction

$$\theta < \frac{2 g_0^T g_0}{g_0^T g_0 - g_0^T g_1}.$$

Or, if we write $\beta = \dfrac{g_0^T g_1}{g_0^T g_0}$ the descent condition is equivalent to

$$h < \frac{2\bar{h}}{1-\beta} \quad \text{if} \quad \beta < 1$$

and

$$h > 0 \quad \text{if} \quad \beta \geq 1$$

To get some insight into these conditions we suppose that $\bar{h}$ is very small so that g_0 and g_1 are roughly parallel. If we are in a region of negative curvature then we expect $\|g_1\| > \|g_0\|$ and so β will typically be greater than 1 and the choice of h is unrestricted. If however we are in a region of positive curvature then β will typically be less than 1 and there is an upper bound on the choice of h. In particular, if we are in a region with a steep-sided narrow valley then even a small step $\bar{h}$ may cause both $g_1^T g_0$ and β to be negative. This potentially puts quite a strong restriction on h, since if we imagine $\bar{h}$ corresponding to a step across a valley such that $g_0^T g_1 \approx -g_0^T g_0$ then we have the limit $h < \bar{h}$.

These considerations need to be borne in mind both when choosing $\bar{h}$ and h at the beginning of an iteration and also when controlling extrapolation steps which involve increasing the current value of h. One important point that this discussion has revealed is that an approach based on the rk2 search direction may do well in a non-convex region but is likely to need more safeguards (and may perhaps be less efficient) in a convex region especially close to a minimum.

During the curvilinear search we need to beware of unhelpful interactions between increases and decreases of step size. When a step reduction occurs after we have made one or more acceptable steps we want to make sure we do not go back to a smaller value of h than one which has already been successful. Similarly if we allow an extrapolation after we have previously made a step reduction then we want to make sure that an increased value of h does not take us into a part of the curvilinear path that we have already rejected. We therefore suppose that we hold running values of h_{max} and h_{min} where h_{min} is the largest step so far

that has produced an acceptable point and h_{max} is the smallest step so far that has failed to give a function decrease. At the start of every search $h_{min} = 0$ and $h_{max} = \infty$.

A simple way of reducing h would involve two steps such as

$$h_{max} = h; \quad h = \frac{1}{2}(h_{max} + h_{min})$$

Rather more flexibly, if we have recorded a slope value f'_{min} corresponding to the step using h_{min} we could use

$$h_{max} = h; \quad f'_{max} = f'_{k+1}; \quad h = h_{min} + (h_{max} - h_{min})\frac{f'_{min}}{f'_{min} - f'_{max}}.$$

This use of linear interpolation assumes (which will often be the case) that $f'_{min} < 0$ and $f'_{max} > 0$. On non-convex functions this assumption cannot be guaranteed however. A robust step reduction strategy will probably combine both formulae to put a lower bound on the next trial to ensure it is not too close to h_{min}.

If we want to increase h then we may use a robust approach which does not take account of f'_k, namely

$$h_{min} = h; \quad h = min\left(2h_{min}, \frac{1}{2}(h_{max} + h_{min})\right).$$

A linear extrapolation approach

$$h = h_{min} + (h - h_{min})\frac{f'_{min}}{f'_{min} - f'_{k+1}}; \quad h_{min} = h; \quad f'_{min} = f'_{k+1};$$

could also be used provided $f'_{min} < f'_{k+1} < 0$, which may not hold on nonconvex function.

It may be better not to do too extensive a search, alternating extrapolation and interpolation. Thus, if $h_{min} > 0$ and if the current h is unacceptable and requires

reduction we could simply terminate the search at the point already reached with the step h_{min}. Similarly, if $h_{max} < \infty$ has already been set then we may elect not to do any extrapolation beyond the first h which is obtained.

7.5 Test results on problem $T1$

As an example, we consider the minimisation of the test problem $T1$ discussed earlier using using prototype Matlab implementations of both algorithms, rk2 and rk2a where the latter uses the angle Θ between the p' and the local negative gradient (7.3), as a measure of when to stop exploring the curvilinear path. The curvilinear searches of CSDP are shown in Figures 7.1 - 7.2. The circled points mark the starts and ends of iterations and the dots indicate trial points obtained with different values of the step h.

It is clear from the figures that both methods lead to the required minimum of the function. However, in both cases they seem to struggle in the convex region as they tend to take more iteration by taking smaller and smaller steps. This must be due to the interaction between $\bar{h}$ and h discussed in Section 7.4 and this suggests that more care must be taken about the choice of $\bar{h}$ and that it is inadequate merely to take $\bar{h}$ as a constant fraction of the step size used on the previous iteration (which is the strategy employed in pour prototype implementation).

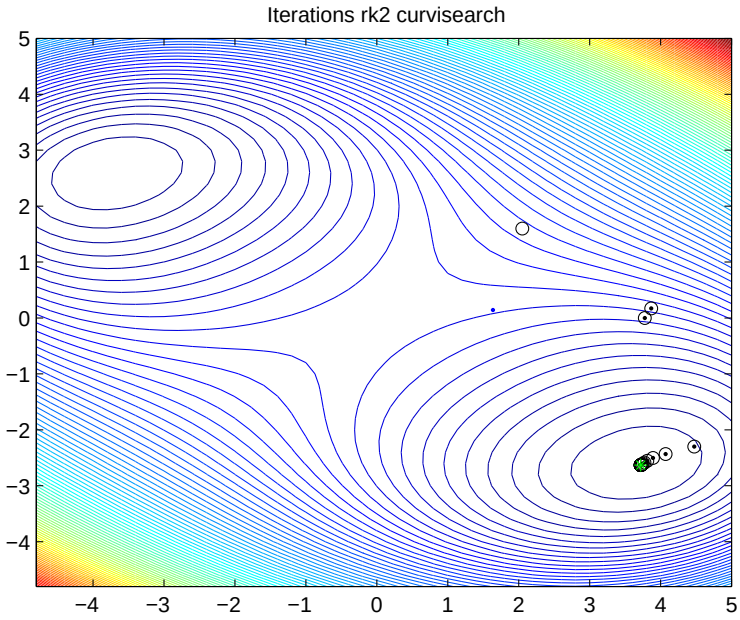

Figure 7.1: The solution path for T1 by rk2.

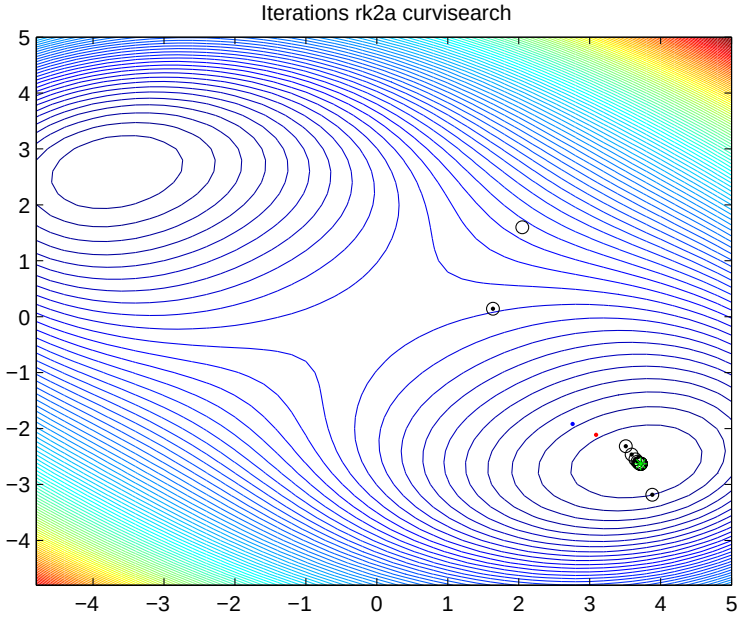

Figure 7.2: The solution path for T1 by rk2a.

7.6 Numerical results on specially constructed test problems

We now consider the performance of the both version of the algorithm of the implicit Runge-Kutta on a wider range of test problems in appendix D.

Functions/Methods	rk2	rk2a
	Its/Fcs	Its/Fcs
T1	24/51	23/50
T1r	23/56	23/53
T1r2	25/61	4/11*
T1a	24/49	24/49
T1b	25/57	24/54
T1ar	23/55	21/50
T2	59/126	59/126
T2r	60/130	63/133
T3	28/63	31/67
T4_2	108/230	108/230
T4_4	518/1054	518/1054
T4_10	653/1334	653/1334
T4_20	747/1528	749/1523
T4_50	842/1724	844/1719
T4_100	2/23	2/23
T4r_20	2/25	2/25
T5	31/79	33/75
T5a	24/63	21/54

Table 7.1: Results from rk2, rk2a

It can be seen from Table 7.1 that while rk2 and rka do solve the problems they are not competitive with other methods discussed previously. This is not completely surprising however, when we remember that they do not use any second derivative information. But in looking at the table we must also remember that the overhead costs per iteration are much smaller less than for the methods reported in previous Chapters. Thus it remains possible that these matrix-free methods may have potential advantages for dealing with non-convexities in optimization problems. To the best of our knowledge, the ideas behind rk2 and rk2a have not been previously tried; and it is to be hoped that some further refinement of this

approach will improve the results.

In view of the restrictions on h in a convex region that were identified in Section 7.4, one obvious refinement would be to consider using a hybrid approach as in the previous Chapter, reserving the CSDP strategy for those iterations which are negotiating a non-convex region.

To the best of our knowledge this is the first time that this technique is used for optimization problem and with some further refinement of this techniques will surely improve the results.

7.7 Hybrid method using the rk2

From our experiments in Section 7.6 it can be seen that rk2 might have more potential in non-convex regions than in convex ones. Hence we could consider using a hybrid approach as in the previous Chapter, reverting to a quasi-Newton technique in the convex region. We therefore propose a new hybrid algorithm where we use rk2 (or rk2a) combined with the BFGS method.

The algorithm which is outlined below attempts a quasi-Newton step on every iteration; but if it turns out that the Hessian estimate cannot be updated because negative curvature is encountered. This of course indicates that the search is in a non-convex region and so the quasi-Newton step is extended by a curvilinear search using rk2 (or rk2a). We call the hybrid algorithm by Hrk2 and we now give an outline of a single step.

Algorithm 7.7.1 *(Hybrid algorithm Hrk2)*

Step 1 *Given* $\bar{h}$, x_k, $f_k = f(x_k)$, $g_k = g(x_k)$, h_{min}, h_{max}
and a positive definite matrix H_k, *calculate* $p_k = -H_k g_k$

Step 2 *Use a standard line search to find* s *so that*
$x_1 = x_k + sp_k$ *and* $f(x_1) < f(x_k)$.

Step 3 *Set* $\delta = x_1 - x_k$, $\gamma = g_1 - g_k$, $g_k = g_1$; $f_k = f_1$; $x_k = x_1$;

Step 4 *If* $\delta^T\gamma > 0$ *then use a* BFGS *update to obtain a new* H_{k+1}
Set $x_{k+1} = x_k$, $f_{k+1} = f_k$ *and* $g_{k+1} = g_k$ *and the iteration is complete*

Step 5 *if* $\delta^T\gamma \leq 0$ *then perform an rk2 step*
(as in algorithm 7.3.1 to obtain a new point x_{k+1}

A related algorithm Hrk2a is obtained by using Algorithm 7.3.2 in Step 5.

An algorithm of this type should make good progress through non-convex regions which are, presumably, far from the optimum; and its ultimate convergence will rest on the good known properties of the BFGS method.

This algorithm performs an rk2 curvilinear search along an approximate CSDP whenever the standard quasi-Newton iteration cannot be used because of negative curvature of the objective function.

7.8 Numerical results on specially constructed test problems

We have tested MATLAB implementations of the hybrid methods Hrk2 and Hrk2a on the test problems described in Appendix D and compared their performance with that of the standard BFGS method QN from the Matlab Optimization toolbox. The results obtained for these problems are given in Table 7.2.

We can see that the hybrid methods Hrk2 and Hrk2a managed to solve all problems to the required accuracy much more efficiently than did the original algorithms rk2 and rk2a. It is even more pleasing to note that in just under half the problems one or other of the hybrid methods does better than QN in terms of iterations or function calls.

Fctns	Hrk2	Hrk2a	QN
	Its/fcs	Its/fcs	Its/fcs
T1	10/14	10/14	12/19
T1r	18/27	18/27	11/28
T1r2	21/35	16/30	11/32
T1a	8/15	8/15	10/13
T1b	8/15	8/15	10/22
T1ar	13/26	13/26	11/31
T2	12/19	12/19	14/27
T2r	14/33	13/27	12/23
T3	16/32	16/32	14/15
T4_2	9/26	9/26	7/18
T4_4	19/42	19/42	17/30
T4_10	28/62	28/62	16/31
T4_20	32/75	32/75	17/44
T4_50	36/81	36/82	23/61
T4_100	4/40	39/94	26/74
T4r_20	3/37	32/75	14/41
T5	8/18	8/18	6/11
T5a	12/25	12/25	9/14

Table 7.2: Results from Hrk2, Hrk2a and QN for specially designed test problems

As preliminary results these are quite encouraging. They suggest there is scope for further development and there are several obvious areas where this might be done. We have already mentioned the need for careful choice of search parameters $\bar{h}$ and h which will influence performance in non-convex regions. We also note that in Hrk2 the CSDP search begins after a quasi-Newton step (possibly involving a line search) has already been taken. It would however also be possible (and might be an advantage) to begin the curvilinear search from the old point x_k whenever an exploratory quasi-Newton step reveals negative curvature.

Closer examination of the results in Table 7.2 shows that rk2 is used on only quite a small number of iterations. In other words the test on $\delta^T\gamma$ at Step 4 of Hrk2 does not detect non-convexity as reliably as would be the case if exact second derivatives were available. It might therefore be worth looking for other ways of switching between the two strategies in this hybrid approach.

Chapter 8

Conclusion

8.1 Summary of the research

In this thesis, we have investigated methods of finding unconstrained local minima of non-convex functions, by following the solution curve of a system of ordinary differential equations. These ODE-based optimization algorithms can generate curvilinear paths which replace or augment the steps taken by line search and trust region methods. The motivation for this was the fact that existing methods (mainly based on Newton methods with line search) can sometimes be inefficient or even terminate at a non-stationary point when applied to problems that do not a have positive-definite Hessian. Even when they do terminate at a stationary point it could be a saddle or maximum rather than a minimum. We have shown that an efficient and logical way to traverse a non-convex region is to approximately follow the Continuous Steepest Decent Path (CSDP).

Most of algorithms we have derived are gradient based methods which require the solution of systems of equations at each step but do not use a line search in the usual sense. Progress along the CSDP is governed both by the decrease in the function value and measures of accuracy of a local quadratic model. These techniques do not require the Hessian matrix $G(x)$ to be positive definite and can minimise a general function rapidly and furthermore they are capable of quadratic convergence if they revert to the Newton method near the solution.

The basic iteration of most of our proposed techniques is based on using the implicit Euler method to solve the steepest descent equation

$$\frac{dx}{dt} = -g(x) \tag{8.1}$$

where $g(x)$ is the gradient of the objective function. This gives a correction step away from the solution estimate x_k by solving

$$(\mu I + G_k)p_k = -g_k \tag{8.2}$$

(where G_k is the local Hessian and $g_k = g(x_k)$ the local gradient) for different value of μ chosen in a range $0 < \mu_{min} < \mu < \infty$, where μ_{min} is the absolute value of the most negative eigenvalue of G_k.
Our use of (8.2) means that we must find the eigensystem of G_k but this enables us to perform a curvilinear search quite efficiently during each iteration.

8.2 Research and development of CSDP

Following on from a review of ways of dealing with difficulties caused by non-convex regions in minimisation problems, Chapters 1 and 2, introduce the idea of following an approximation to the steepest descent continuous path. This idea has been explored by many authors, but the algorithms we have chosen to implement and test are particular versions of the solution to an initial value problem which is described in Chapter 3. On each iteration, these algorithms generate a search curve which is initially tangent to the negative gradient and, regardless of whether the Hessian at the initial point is positive definite or not, will give us a step p_k such that $f(x_k + p_k) < f(x_k)$ as long as μ is sufficiently large. One important point to make here is that when G_k is positive definite, we can use $\mu = 0$ and p_k is then the classical Newton step. This gives quadratic convergence near the minimum. However, it has also been shown in [5] that we can sometimes use negative values of μ to extrapolate beyond a Newton step when x_k is not sufficiently near a minimum for quadratic convergence to occur.

In Chapters 3 and 4 we develop two algorithms called Nimp1 and Nimp2 which do a curvilinear search on each iteration by solving (8.2) for various values of μ. We have also proposed UNMIN which is a version of Behrman's gradient flow algorithm [7] which uses a similar curvilinear search in place of Behrman's own step size procedure. These three algorithms all rely on knowledge of the full eigensystem of the true Hessian G_k, which is calculated via the orthogonal factorisation $G_k = RDR^T$. For a single solution of (8.2), the eigenvalue calculation is more expensive than a Cholesky factorisation. But if several values of μ are tried then subsequent solutions of (8.2) are cheaper than re-factorisation. Hence, we propose search algorithms that make several steps along $p(\mu)$ so long as we can keep sufficient closeness to CSDP combined with sufficient decrease in the objective function. The practical merit of using the CSDP method depends on how far and how accurately we want to pursue a curved path solution of the initial value problem.

In the case when the Hessian G_k is positive definite a hybrid technique could be used which could revert to the standard well known line search version of the Newton method. However, it is also possible to use a curvilinear search by extrapolating μ below zero. We believe our work differs from previous proposals in the way we use μ as a curvilinear search parameter and also in the use of a 2^{nd} order estimate of the CSDP step in Nimp2.

Chapter 4 studies the sensitivity of our methods to certain curvilinear search parameter values. We consider the initial value and the subsequent adjustment of μ on each iteration. As in trust region methods, we can relate the initial μ to an estimate of the size of the step p_k to give a safeguarded formula to be used in subsequent iterations. This was found to have an appreciable effect on the number of function evaluations and to a lesser extent the number of iterations. Thresholds on the accuracy parameters which control how far the search is pursued along the approximate CSDP were also studied . It was also shown, even when starting very close to a saddle point, Nimp1 and Nimp2 can find the minimum of the function and avoid the saddle point, but at a cost of more function evaluations the nearer we start to the saddle.

The performance of CSDP methods was tested on some specially chosen problems, with large non-convex regions - and sometimes saddle points - which are close to local minima. They have turned out to be quite competitive with a trust region routine (TR) and a quasi-Newton routine (QN) from the MATLAB optimisation tool box. Nimp1 quite often outperforms most of the other routines although its behaviour is not as consistent as might be hoped.

We have also tried Nimp1, Nimp2 and UNMIN on about 139 unconstrained problems taken from the Constrained and Unconstrained Test Environment revisited version (CUTEr). The CSDP methods Nimp1 and Nimp2, with their curvilinear search, typically solve most problems in fewer iterations than QN or TR without failing or reaching the maximum number of iterations. Competitiveness however in terms of function calls still depends on the parameters in the curvilinear search which remains an area for ongoing research.

In Chapter 5 we described variants of Nimp1 where the exact Hessian is replaced by a cheaper quasi-Newton approximation *that is not forced to be positive definite*. This requires the evaluation of the function and its gradient only at each iteration. In particular, we propose diagonal approximations to the Hessian matrix so as to make the calculation of x_{k+1} less expensive, by using either diagonal terms of the true Hessian or a diagonal form of quasi-Newton update. We also considered using a quasi-Newton updated estimate of the full Hessian based on standard updates such as BFGS, DFP or SR1 formulae.

We compared results from these quasi-Newton CSDP algorithms with those from standard quasi-Newton and trust-region methods. On some of the test examples the performance of SR1imp and BFGSimp was comparable with the standard quasi-Newton approach in QN. However the behaviour of DFPimp and diagonalised Hessian approximations was markedly inferior when applied to the special test problems. On the CUTEr test problems, we sometimes observed rapid convergence in the case of SR1imp and to some extent with BFGSimp. Less satisfactory was the performance of DFPimp however. Diagonal approximations

did well on a few problems having sparse Hessian such as DIXMAAN*. However, these methods also experience the most failures.

In Chapter 6, hybrid approaches were introduced in which CSDP methods were used only in non-convex regions with standard line search methods being employed in convex regions. We, proposed a hybrid algorithm in our investigations with Nimp1 being used in the non-convex regions while elsewhere we use the Newton method. We called the hybrid algorithm by HNNimp1. Numerical results show that this method is effective in practice. We also proposed another hybrid approach based on the use of the Hessian approximations as in Chapter 5, where we used the BFGS update in the convex region and a diagonal approximation to the Hessian matrix $G(x_k)$ in the non-convex region. We called this hybrid method by HDBFGS. In both cases these algorithms perform a curvilinear search along an approximate CSDP whenever standard methods cannot be used because of negative curvature of the objective function.

Results obtained for the specially constructed problems using these techniques show some improvement compared with those obtained in previous Chapters. In particular, HNNimp1 generally does slightly better than the original Nimp1. HDBFGS did well on the number of iterations compared with the techniques discussed in Chapter 5. However it was not so competitive in terms of number of iterations when compared with the quasi-Newton method QN. This is may be due to backtracking technique in the weak line search we used with the BFGS methods and other line search techniques may be better suited.

When used with CUTEr test problems, both hybrid methods have shown improvement in terms of numbers of iteration. In particular HNNimp1 gave a considerable improvement on both Nimp1 and TR algorithms. Some improvement was seen when using HDBFGS when compared with QN.

In Chapter 7, we developed a matrix free approach based on solving the steepest descent equation by an explicit second order Runge Kutta method. This offers a substantial reduction in the overhead calculations on each iteration because it

does not use the Hessian at all. Of course the correction steps may be less effective because of the lack of second order information. Prototype algorithms prove capable of solving specially constructed test problems but they take rather small steps in convex regions. This suggests strongly that the approach should be used in hybrid setting like that proposed in Chapter 6. We have obtained promising results in initial tests with hybrid codes which combine an rk2 curvilinear step with a standard BFGS method.

8.3 Reflections on the Project

The CSDP path based on second derivatives seems to work well in non-convex regions but in convex regions around a solution it can be less efficient than we had hoped and was often inferior to standard line-search methods. Results with approximate Hessian information were also promising but frustratingly inconsistent. The hybrid techniques in Chapter 6 seem to be the way forward and we still hope for more success from the hybrid approach which uses a diagonal Hessian estimate in non-convex regions and a standard BFGS approach when the function appears to be convex. This combination of tactics has the potential significantly to reduce the overhead costs of the eigenvalue calculations which are a feature of Nimp1. The most innovative proposal in the thesis emerged rather late in the project and is the low-cost RK2 approximation to CSDP described in Chapter 7. This - probably in a hybrid context - does seem to deserve more study in a follow-up research project. In particular the investigation of the RK2 approach shed more light on reasons why a CSDP path may be restricted to small step sizes in the vicinity of a minimum.

Our approach throughout this thesis has been a rather practical one. We have mentioned convergence issues from time to time, especially when we are speaking of hybrid approaches where behaviour near the solution is that of conventional Newton or quasi-Newton methods whose termination properties are well-known. But we have not proved any significant properties about the guaranteed progress of the Nimp1 curvilinear search or about its complexity properties of the kind discussed in [25].

In order to remedy this lack of convergence theory it may be possible to use ideas similar to those of Higham [41] in relation to his trust-region algorithm given in section 3.3.1. This algorithm has some similarities with Nimp1 (but also some important differences). Higham's convergence results are as follows

Theorem 8.3.1 *Suppose that Algorithm 3.3.2 produces an infinite sequence such that $x_k \in B \subset \mathbb{R}^n$ and $g_k \neq 0$ for all k, where B is bounded and $f \in C^2$ on B. Then there is an accumulation point x_* that satisfies the necessary conditions for a local minimiser.*

Theorem 8.3.2 *If the accumulation point x_* of Theorem 8.3.1 also satisfies the sufficient conditions for a local minimiser, then for the main sequence $p_k \to 0$, $\mu \to 0$ and $r_k \to 1$. Further, the displacement error $e_k = \|x_k - x_*\|$ satisfies*

$$e_k \leq \frac{C}{2^{k^2/3}} \tag{8.3}$$

for some constant C, and if $e_k > 0$ for all k,

$$\frac{\tilde{C}}{2^{k^2/2}} \leq e_k \leq \frac{C}{2^{k^2/3}} \tag{8.4}$$

$$\frac{\bar{C}}{2^k} \leq \frac{e_{k+1}}{e_k} \leq \frac{\hat{C}}{2^k}, \tag{8.5}$$

for constants $\tilde{C}$, $\bar{C} > 0$ and $\hat{C}$, but the ratio e_{k+l}/e_k^2 is unbounded.

Theorem 8.3.1 establishes that Higham's method is capable of convergence to a local minimum while Theorem 8.3.2 is concerned with its ultimate *rate* of convergence. Higham's rate of convergence result is not very relevant to Nimp1 since this algorithm, unlike Higham's method, is able to choose $\mu = 0$ near the minimum when the Hessian matrix is positive definite and hence can have the same quadratic convergence rate as the Newton method. But we would like to be able to prove a result similar to theorem 8.3.1 which states that any termination point of Nimp1 will be a local minimum.

The method used by Higham to prove theorem 8.3.1 rests on the fact that each iteration of his algorithm uses a value of μ which is simply related to the value from the previous iteration. He is then able to distinguish two cases, namely that μ tends to infinity as the iterations proceed or else μ remains bounded. In each case he can show that assuming the theorem is false leads to a contradiction.

Unfortunately, this line of argument may not be applicable to Nimp1 since the value of μ used to determine a correction step on iteration k is independent of that used on iteration $k-1$. We observed in Chapter 3 that this could be viewed as an extra degree of flexibility; but it turns out to be a possible theoretical weakness that needs further consideration. This is not to say that a convergence result for Nimp1 does not exist; but it suggests that some tightening of the algorithm may be required which would put some bounds on the ratio between the μ-values used on successive iterations.

We do not, in fact, need a result as strong as theorem 8.3.1 for Nimp1 - particularly in its hybrid form - because it is expected to revert to Newton's method once the convex neighbourhood of a local minimum is reached. What we do need is a proof that the iterations performed when G_k is indefinite will guarantee that the search does indeed escape from the non-convex region. This remains a topic for further consideration.

In [25], Curtis, Robinson and Samadi propose a trust region algorithm for solving non-convex smooth optimization problems that, in the worst case, is able satisfy the stationarity condition $\|g(x_k)\| \leq \epsilon$ in at most $O(\epsilon^{-3/2})$ iterations. This algorithm follows the traditional trust region framework but uses a modified step acceptance criteria with different trust region update strategies that allow the algorithm to achieve this bound on iteration numbers. Curtis et al [25] have also proved global and fast convergence of their method under similar assumptions to other trust regions methods. This is an improvement on the $O(\epsilon^{-2})$ bound known to hold for some other trust region algorithms and matches that of another recently proposed ARC algorithm by Cartis, Gould and Toint [20], [21]. We are not yet in a position to make any such claims or comments about our own proposed

methods.

8.4 Possible Further Work

Some areas of ongoing research have been mentioned in the previous section. Other issues where improvement can be looked for concern the cost of solving the basic CSDP equation (8.1). One possible approach involves writing (8.1) as

$$(I + hG)p = -hg \tag{8.6}$$

where g and G are evaluated at the point x_k and h is the step size. Equation (8.6) can be solved by a direct method (possibly Cholesky if h is small enough to prevent the coefficient matrix having any negative eigenvalues even when G is not positive definite).

We could use an approximate solution to (8.6) for small h without any matrix factorizations or eigenvalue calculations. To do this we could use the first few terms of

$$(I + hG_k)^{-1} = I - hG_k + h^2G_k^2 - h^3G_k^3 + \cdots$$

to generate

$$p_k \approx -hg_k - \sum_{j=1}^{J} h^{j+1}\gamma_j, \quad \text{where} \quad \gamma_j = (-1)^j G_k^j g_k.$$

In this case we can estimate the solution of (8.6) when $\mu >> ||g_k||$ by using

$$p_k \approx -\frac{1}{\mu}\left[g_k + \sum_{j=1}^{J}(\frac{1}{\mu})^j\gamma_j\right]. \tag{8.7}$$

Repeated use of this expression might allow us to approximate quite cheaply the initial part of integral curve.

There is still work to be done in refining the curvilinear search. Throughout this thesis we have used a rather simple rule for stretching or shrinking the step

size by a constant factor. The results in Curtis, Robinson and Samadi [25] are based on a more complicated search using quadratic interpolation and a more sophisticated way of choosing an initial step for each iteration. The results in Bartholomew-Biggs, Beddiaf and Kane [5] also show that such refinements can produce successful results and the version of Nimp1 which is tested in [5] (on a more limited set of examples than in this thesis) compares favourably not only with the Matlab routine TR but also with the trust region method NMTR due to More and Sorenson [50].

This paper [5] incidentally provides some justification of the remark we have made several times that the Nimp1 search direction can usefully be computed with negative values of μ in (8.2) when the search is in a convex region. The coefficient matrix in (8.2) will be positive definite if $\mu > \mu_{min} = \lambda_{min}$ where λ_{min} (> 0) is the smallest eigenvalue of G_k. More work could be done on considering if and when it is worth extrapolating beyond a trial point $x_k = x_k + p_k$ by reducing μ below zero. Such decisions can be made on the basis of the ratios D_1 and D_2 as defined in Section 3.2 and on quadratic interpolations through the three most recent trial points.

Another interesting issue is the consideration of alternative ways of solving the trust region problem which defines steps along the curvilinear path. We have already made some comments on this but we can also include the iterative approach outlined by Curtis, Robinson and Samadi [25].

Yet another possibility for defining steps along the curvilinear path is to consider the constrained quadratic minimization problem which underlies it. Hamaker [40] has proposed an iterative approach to problems of this type which appears to be very efficient, under certain conditions. This approach could be applied to our algorithms for choosing the direction p_k.

In relation to the rk2 and Hrk2 methods in Chapter 7, we note that explicit Runge-Kutta methods are considered unsuitable for the solution of stiff equations because their region of absolute stability is small. It is this instability of ex-

plicit Runge-Kutta methods that motivates the development of implicit methods. Hence we might also consider the use of Implicit Runge-Kutta (IRK) methods [16] for solving

$$\frac{dx}{dt} = F(x) = -g(x).$$

Mohr [47] and Bartholomew-Biggs & Brown [16] give more details of this.

The problem with the IRK methods when solving differential equations is that they require the solution to possibly a system of non-linear equations at each iteration. Solving this can be very expensive and also requires the evaluation of the second derivatives. Quasi-Newton methods could be used to estimate the exact Hessian, since they have lower cost per iteration. Zghier [63] suggests a search direction calculation based on the application of a generalised Trapezoidal rule namely

$$(I + \theta hG)p = -hg. \tag{8.8}$$

where θ is a parameter. It can easily be seen that when $\theta = 0$ this reduces to the steepest descent (or forward Euler) method while $\theta = 1$ yields the implicit Euler method. The choice $\theta = 1/2$ gives the Trapezoidal rule for integration and this might be an easy-to-apply and useful variation on Nimp1.

The list of partly-open questions in this final section may help to explain why ODE-based methods of optimization have long been and still continue to be a fruitful research topic.

Appendices

Appendix A

CUTEr Test Results for Nimp1, Nimp2, UMINH, UMINH, TR and QN

The following table presents the numerical results obtained for the different algorithms. The stopping criterion used are given by the inequality $\|g(x_k)\|_2 < 10^{-6}$ or $\|x_{k+1}\| < 10^{-6}(1 + \|x_k\|)$. All computations were performed on Intel 2.60 GHz processors running Linux and Matlab R2010a. The test problems range from 2 to 1000 variables. The following notation were used:

- n : The number of variables in the problem .
- Its/Fcs: The number of iterations / function calls.
- "*": Signify a possible minimum has been reached but cannot be certain because the first order optimality condition measure is not fully satisfied.
- "**": Means that the maximum number of iterations has been reached.
- "F" :The routine has failed to solve the problem.
- We highlight in bold the best performance measured in terms of the numbers of iterations. The best performance in terms of function evaluations by italics and we underline the entry which gives the smallest sum of iterations and function calls.

	Methods		Nimp1	Nimp2	UNMIN	UMINH	TR	QN
	Problem	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
1	AKIVA	2	9/22	7/10	15/16	***6/7***	***6/7***	19/23*
2	ALLINITU	4	**7/14**	11/20	15/21	***7/8***	10/11	11/13
3	ARGLINA	200	2/7	5/14	15/16	***1/2***	2/3	2/3
4	ARWHEAD	100	**6/16**	**6/15**	13/14	***6/7***	***6/7***	**6/8**
5	BARD	3	12/31	12/20	15/16	***7/8***	192/193*	20/22
6	BDQRTIC	100	12/31	**10/26**	*19/20*	F	9/10*	129/131*
7	BEALE	2	***6/9***	8/11	17/19	8/51	9/10	15/16
8	BIGGS6	6	53/180	6247/82092	92/189	F	***41/42***	46/48
9	BOX	10	***3/4***	***3/4***	13/14	***3/4***	***3/4***	5/8
10	BOX3	3	11/34	**7/10**	12/14	*8/9*	13/14	24/25
11	BRKMCC	2	5/8	**3/5**	13/14	***3/4***	***3/4***	5/8
12	BROWNAL	200	7/11	7/12	**2/7**	15/16	7/8	***2/4***
13	BROWNBS	2	***8/36***	F	16/18*	7/53*	7075/7076	14/25
14	BROWNDEN	4	**8/20**	9/33	15/16	***8/9***	11/12*	32/33*
15	BROYDN7D	10	***14/31***	**14/36**	20/28	F	17/18*	23/27
16	BRYBND	10	*11/24*	11/25	20/24	**10/40**	16/17*	27/30*
17	CHAINWOO	4	56/144	2406/431457	***51/89***	F	208/209*	35/43*
18	CHNROSNB	50	48/170	572/4784	***45/83***	F	64/65*	240/248*
19	CLIFF	2	28/266	29/419	***27/28***	***27/28***	26/27*	F
20	COSINE	10	11/20	12/23	***7/13***	8/27	9/10*	12/17
21	CRAGGLVY	4	17/69	18/46	F	F	***14/15***	54/62
22	CUBE	2	31/93	**27/47**	36/58	F	*31/32*	34/40
23	CURLY10	100	***12/29***	98/650	24/41	F	17/18*	920/931
24	CURLY20	100	***14/32***	322/3055	26/43	19/55	17/18*	863/874
25	CURLY30	100	***15/34***	651/7409	26/43	F	17/18*	744/753

	Methods		Nimp1	Nimp2	UNMIN	UMINH	TR	QN
	Problem	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
26	DECONVU	61	**21/43**	**	F	F	***21/22***	80/81
27	DENSCHNA	2	6/9	6/9	15/16	6/7	***5/6***	10/11
28	DENSCHNB	2	7/12	7/11	F	F	***6/7***	7/8
29	DENSCHNC	2	**9/22**	13/46	18/19	11/32	*10/11*	21/24
30	DENSCHND	3	30/78	30/132	43/59	***29/31***	257/258	23/28*
31	DENSCHNE	3	20/39	925/12013	27/39	11/38	***10/11***	37/58
32	DENSCHNF	2	**6/9**	7/10	15/16	***6/7***	***6/7***	10/11
33	DIXMAANA	15	7/15	**5/10**	18/21	***5/10***	8/9	15/16
34	DIXMAANB	15	**7/16**	**7/13**	19/23	***7/8***	***7/8***	22/23
35	DIXMAANC	15	9/19	8/22	19/23	***7/8***	8/9	26/27
36	DIXMAAND	15	10/21	**6/11**	19/24	*8/9*	9/10	30/31
37	DIXMAANE	15	8/17	7/16	16/20	7/9	***6/7***	71/72
38	DIXMAANF	15	9/20	**7/14**	18/24	*8/9*	9/10	74/75
39	DIXMAANG	15	11/22	**9/18**	19/25	F	*10/11*	81/82
40	DIXMAANH	15	11/25	**8/15**	18/25	*10/11*	*10/11*	84/85
41	DIXMAANI	5	10/23	**6/14**	16/22	8/12	***6/7***	97/98
42	DIXMAANJ	5	13/35	**9/18**	15/23	***9/10***	12/13	105/106
43	DIXMAANK	15	13/32	**11/22**	17/25	F	*13/14*	118/119
44	DIXMAANL	15	**12/33**	13/20	17/25	F	*13/14*	131/132
45	DIXON3DQ	10	3/7	3/8	13/14	***1/2***	20/21	20/21
46	DJTL	2	5558/54577	127/260	135/12051	F	***103/104***	2189/5715
47	DQDRTIC	10	4/10	4/12	16/17	***1/2***	***1/2***	14/16
48	DQRTIC	10	23/48	**10/35**	F	F	*15/16*	39/40*
49	EDENSCH	36	11/27	13/41	*18/19*	**10/33**	12/13*	69/71*
50	EG2	1000	5/7	4/6	***3/4***	**3/32**	***3/4***	4/5

	Methods		Nimp1	Nimp2	UNMIN	UMINH	TR	QN
	Problem	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
51	ENGVAL1	2	**6/9**	**6/9**	17/18	*7/8*	*7/8*	11/12
52	ENGVAL2	3	18/40	**15/29**	*26/33*	18/48	108/109*	29/34
53	ERRINROS	50	134/372	257/2679	***106/187***	F	56/57*	329/352*
54	EXPFIT	2	**8/18**	9/16	15/22	F	*11/12*	12/16
55	EXTROSNB	10	469/1895	***393/1295***	452/768	F	330/331*	325/403
56	FLETCBV2	10	***2/3***	**2/3**	11/12	***2/3***	3/4*	15/18
57	FLETCBV3	10	855/1530	788/2350	1137/3651	1263/1266	298/299	***35/78***
58	FLETCHBV	10	***1374/2594***	1598/4377	2072/6053	F	334/335*	96/150*
59	FLETCHCR	10	25/71	***22/57***	35/57	F	131/132*	51/70
60	FMINSRF2	16	13/25	17/77	15/30	F	***10/11***	27/42
61	FMINSURF	16	13/22	17/92	12/24	F	***11/12***	27/47
62	GENROSE	500	***299/889***	**	**	**	614/615	876/1431
63	GROWTHLS	3	799/1703	1350/22456	***368/538***	F	**	12/13*
64	GULF	3	**42/126**	**	63/104	F	301/302	*50/58*
65	HAIRY	2	76/144	64/107	84/137	F	91/92	***20/42***
66	HATFLDD	3	22/66	19/35	*19/21*	**18/42**	114/115*	19/27
67	HATFLDE	3	21/49	18/27	21/24	18/25	15/16*	***9/12***
68	HEART8LS	8	88/185	**	**86/160**	F	*131/132*	2921/3815
69	HELIX	3	*12/22*	**10/23**	21/25	11/37	748/749*	29/31
70	HIELOW	3	***7/17***	211/3074	15/17	F	10/11*	28/31*
71	HILBERTA	2	4/8	4/10	12/13	***1/2***	***1/2***	5/7
72	HILBERTB	10	2/5	3/7	16/17	***1/2***	4/5	6/7
73	HIMMELBB	2	21/49	**7/18**	17/18	*11/13*	13/14*	3/8*
74	HIMMELBF	4	1943/2934	212/2074*	***912/938***	F	201/202*	48/50*
75	HIMMELBG	2	***4/5***	6/7	15/16	5/40	7/8	8/10

	Methods		Nimp1	Nimp2	UNMIN	UMINH	TR	QN
	Problem	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
76	HIMMELBH	2	***4/5***	**4/8**	F	F	7/8	7/8
77	HUMPS	2	871/1445	607/5780	560/790	F	5459/5460	***104/320***
78	JENSMP	2	12/34	10/16	16/17	F	***9/10***	16/41*
79	KOWOSB	4	10/23	822/15632	***9/10***	F	21/22	33/37
80	LIARWHD	36	10/20	**7/14**	23/24	*10/11*	*10/11*	18/19
81	LOGHAIRY	2	782/1506	676/1176	4019/6201	F	514/515*	***41/168***
82	MANCINO	100	**10/28**	11/47	17/38*	*11/12*	15/16*	6/9*
83	MARATOSB	2	7/8*	4/42*	748/2212*	F	779/780*	5/7*
84	MEXHAT	2	12/44*	*38/94*	**25/32**	F	29/30*	11/16*
85	MODBEALE	10	10/21	26/123	19/24	***7/8***	43/44	60/63
86	MOREBV	10	3/5	3/5	10/11	***2/3***	41/42	32/34
87	NONCVXU2	10	15/32	14/71	9/19	**8/32**	*17/18*	35/38
88	NONCVXUN	10	15/26	27/142	**11/26**	15/110	*18/19*	31/34
89	NONDIA	10	24/48	7950/269739	F	F	***14/15***	14/24*
90	NONDQUAR	100	**17/133**	**17/146**	***17/18***	**17/18**	26/27	375/380
91	NONMSQRT	9	**61/295**	*72/215*	**235/456**	F	88/89*	F
92	OSBORNEA	5	35/157	41/200	38/60	F	56/57*	***37/46***
93	OSBORNEB	11	**21/53**	2936/43098	*24/29*	F	30/31	67/78
94	OSCIGRAD	10	12/26*	15/68	**	***9/33***	17/18*	43/44*
95	OSCIPATH	10	*5/15*	**2/46**	**	F	2/3*	**
96	PALMER1C	8	**11/32**	13/53	***11/12***	F	**	36/44*
97	PALMER1D	7	9/27	11/48	13/14	***1/2***	**	62/66
98	PALMER2C	8	11/32	12/43	8/9	***1/2***	**	60/64
99	PALMER3C	8	10/30	15/52	8/9	***1/2***	**	56/60
100	PALMER4C	8	9/29	14/51	9/10	***1/2***	**	56/60

	Methods		Nimp1	Nimp2	UNMIN	UMINH	TR	QN
	Problem	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
101	PALMER5C	6	2/7	3/10	14/15	***1/2***	8/9	14/16
102	PALMER6C	8	8/30	12/50	7/8	***3/4***	**	43/51
103	PALMER7C	8	7/28	13/56	9/10	***2/3***	**	28/35
104	PALMER8C	8	9/29	12/50	8/9	***2/3***	**	49/54
105	PENALTY1	10	41/120	**29/73**	*37/56*	F	15/16	16/17*
106	PENALTY2	200	***11/42***	12/1788	14/127	F	F	290/292*
107	PENALTY3	50	***15/37***	29/71	17/41	F	18/19	40/65*
108	PFIT1LS	3	321/1350	321/1212	***311/453***	F	**	33/43
109	PFIT2LS	3	121/462	140/504	127/201	F	***35/36***	476/644
110	PFIT3LS	3	**107/437**	122/424	*134/207*	F	33/34*	648/904
111	PFIT4LS	3	**229/951**	252/974	*262/411*	F	43/44*	969/1296
112	POWELLSG	4	17/66	**16/37**	24/25	*17/18*	23/24	33/34
113	POWER	10	18/60	18/43	20/21	19/20	***15/16***	77/78*
114	QUARTC	25	27/58	**14/60**	F	F	*19/20*	56/ 57
115	ROSENBR	2	25/66	**20/34**	30/44	F	*27/28*	36/46
116	S308	2	**8/15**	**8/14**	18/21	F	*10/11*	16/17
117	SBRYBND	10	41/145	F	***25/51***	F	215/216	267/299
118	SCHMVETT	10	**4/6**	**4/6**	13/14	**4/25**	***4/5***	14/21
119	SENSORS	100	21/42	22/47	26/51	25/76	***16/17***	26/34
120	SINEVAL	2	59/184	**46/115**	59/122	F	*55/56*	63/86
121	SINQUAD	5	**7/13**	8/15	16/20	F	*11/12*	11/17
122	SISSER	2	13/39	13/27	14/15	14/15	11/12	***9/11***
123	SNAIL	2	65/178	**64/182**	157/301	F	*104/105*	94/127
124	SPARSINE	10	10/15	276/6297	*5/8*	**5/9**	*7/8*	36/37
125	SPARSQUR	10	15/46	14/31	16/17	15/16	***12/13***	36/37

	Methods		Nimp1	Nimp2	UNMIN	UMINH	TR	QN
	Problem	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
126	SROSENBR	10	10/22	***7/9***	16/17	F	8/9	16/18
127	TESTQUAD	1000	7/20	8/25	15/16	***1/2***	2/3	259/260
128	TOINTGOR	50	***6/13***	**6/18**	12/13	**6/26**	28/29	137/138
129	TOINTGSS	10	10/18	19/147	32/54	***1/2***	14/15	25/39
130	TOINTPSP	50	**14/46**	19/49	23/123	F	*28/29*	40/62
131	TOINTQOR	50	3/7	3/9	14/15	***1/2***	10/11	40/41
132	TQUARTIC	10	10/18	20/878	16/21	***1/2***	***1/2***	12/18
133	TRIDIA	10	3/6	3/7	14/15	***1/2***	5/6	21/22
134	VARDIM	200	28/292	29/437	28/31	28/29	29/30	***1/2***
135	VAREIGVL	50	39/43	**13/15**	***13/14***	F	15/16	20/21
136	WATSON	12	17/43	**	***12/15***	13/20	*13/14*	73/77
137	WOODS	100	**53/150**	9411/166331	62/105	F	*58/59*	35/43*
138	YFITU	3	44/200	**41/115**	180/332	F	3119/3120	*72/98*
139	ZANGWIL2	2	2/6	2/7	13/14	***1/2***	***1/2***	**1/2**

Appendix B

CUTEr Test Results for BFGSimp, DFPimp, SR1imp, DiagHess, ApproxHess and QN

The following table presents the numerical results obtained for the different algorithms. The stopping criterion used are given by the inequality $\|g(x_k)\|_2 < 10^{-6}$ or $\|x_{k+1}\| < 10^{-6}(1 + \|x_k\|)$. All computations were performed on Intel 2.60 GHz processors running Linux and Matlab R2010a. The test problems range from 2 to 1000 variables. The following notation were used:

- n : The number of variables in the problem .
- Its/Fcs: The number of iterations / function calls.
- "*": Signify a possible minimum has been reached but cannot be certain because the first order optimality condition measure is not fully satisfied.
- "**": Means that the maximum number of iterations has been reached.
- "F" :The routine has failed to solve the problem.
- We highlight in bold the best performance measured in terms of the numbers of iterations. The best performance in terms of function evaluations by italics and we underline the entry which gives the smallest sum of iterations and function calls.

	Methods		BFGSimp	DFPimp	SR1imp	DiagHess	ApproxHess	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
1	AKIVA	2	**12/41**	14/53	13/38	19/55	21/79	*19/23*
2	ALLINITU	4	11/20	**10/15**	F	F	F	*11/13*
3	ARGLINA	200	**2/7**	**2/7**	**2/7**	**2/7**	**2/7**	***2/3***
4	ARWHEAD	100	9/24	10/40	7/16	**5/12**	**5/9**	*6/8*
5	BARD	3	21/80	265/1917	**14/33**	3799/5486	30/71	*20/22*
6	BDQRTIC	100	81/150	83/298	28/167	***16/24***	33/112	129/131
7	BEALE	2	**13/22**	183/1164	18/45	149/319	F	*15/16*
8	BIGGS6	6	**44/202**	**	50/175	1689/4168	88/255	*46/48*
9	BOX	10	7/16	6/14	**5/13**	22/31	30/80	***5/8***
10	BOX3	3	23/80	1739/13518	**13/31**	265/927	56/93	*24/25*
11	BRKMCC	2	7/13	6/12	6/9	72/132	95/225	***5/8***
12	BROWNAL	200	10/23	10/23	10/21	22/60	4/10	***2/4***
13	BROWNBS	2	3/43*	3/16*	3/16*	15/60*	4/40*	14/25*
14	BROWNDEN	4	21/48	43/148	**19/47**	117/324	64/152	*32/33*
15	BROYDN7D	10	36/67	121/543	39/104	133/233	308/958	***23/27***
16	BRYBND	10	44/89	118/619	25/58	2635/7899	**19/49**	*27/30*
17	CHAINWOO	4	58/209	**	70/238	5374/10164	95/155	***35/43***
18	CHNROSNB	50	***211/216***	**	33/190*	2140/5614	93/309*	240/248
19	CLIFF	2	62/641	4506/63213	4/48*	**	22/51	***1/2***
20	COSINE	10	17/39	15/35	13/35	12/25	**9/23**	*12/17*
21	CRAGGLVY	4	73/287	1605/10378	**40/121**	51/181	F	*54/62*
22	CUBE	2	39/133	**	43/130	5494/9424	345/757	***34/40***
23	CURLY10	100	228/479	125/205	***76/204***	**	131/507	920/931
24	CURLY20	100	177/392	*103/172*	**94/278**	**	168/645	863/874
25	CURLY30	100	149/338	*93/164*	63/408	**	**59/224**	744/753

	Methods		BFGSimp	DFPimp	SR1imp	DiagHess	ApproxHess	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
26	DECONVU	61	127/386	**	**75/217**	6056/7570	F	*80/81*
27	DENSCHNA	2	10/17	10/20	**9/16**	40/43	46/121	*10/11*
28	DENSCHNB	2	8/13	9/14	8/14	***3/6***	7/12	7/8
29	DENSCHNC	2	17/46	23/72	18/41	**16/30**	26/66	*21/24*
30	DENSCHND	3	89/403	**	70/254	1800/5730	254/528	***23/28***
31	DENSCHNE	3	**18/53**	45/181	22/40	*20/39*	F	37/58
32	DENSCHNF	2	**9/13**	11/19	**9/13**	11/16	17/35	*10/11*
33	DIXMAANA	15	15/51	22/88	9/16	***8/11***	10/15	15/16
34	DIXMAANB	15	16/24	29/74	18/36	***7/10***	12/24	22/23
35	DIXMAANC	15	23/35	56/179	29/70	***8/11***	12/25	26/27
36	DIXMAAND	15	27/51	93/314	35/90	***10/13***	F	30/31
37	DIXMAANE	15	26/57	59/209	28/54	***8/13***	F	71/72
38	DIXMAANF	15	35/85	132/574	24/54	***7/11***	9/15	74/75
39	DIXMAANG	15	38/102	183/817	29/62	***8/12***	12/21	81/82
40	DIXMAANH	15	41/117	1025/5810	42/119	***9/13***	F	84/85
41	DIXMAANI	5	62/212	515/3058	26/51	***8/15***	F	97/98
42	DIXMAANJ	5	62/214	1819/11455	51/137	***8/14***	13/30	105/106
43	DIXMAANK	15	70/280	4057/26228	53/164	***9/14***	17/47	118/119
44	DIXMAANL	15	75/296	**	62/212	***9/15***	18/48	131/132
45	DIXON3DQ	10	23/39	39/155	**12/25**	554/832	F	*20/21*
46	DJTL	2	20/142*	7/15*	29/144*	3299/12032*	54/127*	2189/5715*
47	DQDRTIC	10	26/67	16/39	8/16	***4/10***	5/12	14/16
48	DQRTIC	10	207/971	**	76/269	**23/48**	F	*39/40*
49	EDENSCH	36	39/53	129/521	28/78	***20/42***	40/125	69/71
50	EG2	1000	7/9	7/9	7/9	5/7	F	***4/5***

	Methods		BFGSimp	DFPimp	SR1imp	DiagHess	ApproxHess	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
51	ENGVAL1	2	11/21	12/24	12/26	***7/10***	8/12	11/12
52	ENGVAL2	3	32/98	42/182	33/98	681/2875	41/104	***29/34***
53	ERRINROS	50	189/449	**	218/716	6605/19080	***77/298***	329/352
54	EXPFIT	2	**12/29**	14/33	16/37	222/228	F	***12/16***
55	EXTROSNB	10	320/1271	**	***53/177***	**	87/262	325/405
56	FLETCBV2	10	15/25	21/61	**13/32**	182/188	133/188	*15/18*
57	FLETCBV3	10	244/1189	**	***8/35***	840/1522	1138/1888	35/78
58	FLETCHBV	10	***37/60***	**	150/363	1483/2849	1133/2090	96/150
59	FLETCHCR	10	72/178	978/6410	73/259	3771/7380	F	***51/70***
60	FMINSRF2	16	39/100	183/918	35/90	516/1181	275/747	***27/42***
61	FMINSURF	16	31/76	70/284	33/78	88/163	180/600	***27/47***
62	GENROSE	500	**	**	***20/43***	7361/17879	97/373	875/1431
63	GROWTHLS	3	20/96	**	27/113*	**	**12/54**	***12/32***
64	GULF	3	37/198	**	**33/113**	1616/4087	66/180	*50/58*
65	HAIRY	2	**20/43**	**	21/50	55/103	58/155	***20/42***
66	HATFLDD	3	32/110	2030/16863	**18/48**	**	70/182	*19/27*
67	HATFLDE	3	29/117	3764/29812	35/111	7878/16328	101/187	***9/12***
68	HEART8LS	8	186/826	**	***131/457***	**	F	2921/3815
69	HELIX	3	**24/62**	64/305	32/75	1461/2545	F	*29/31*
70	HIELOW	3	15/49	16/53	**12/39**	22/59	24/91	*28/31*
71	HILBERTA	2	10/22	25/101	**4/10**	54/85	111/306	*5/7*
72	HILBERTB	10	15/26	13/23	**5/10**	10/13	10/17	*6/7*
73	HIMMELBB	2	8/18	8/18	8/18	11/14	6/21	***3/8***
74	HIMMELBF	4	36/203	**	29/129*	1859/6079*	***14/31***	48/50
75	HIMMELBG	2	8/14	8/13	9/15	***6/8***	**6/11**	8/10

	Methods		BFGSimp	DFPimp	SR1imp	DiagHess	ApproxHess	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
76	HIMMELBH	2	8/14	7/13	8/14	***4/5***	F	7/8
77	HUMPS	2	162/721	***22/71***	156/563	26/75	30/110	104/320
78	JENSMP	2	17/45	**	20/45	132/154	26/70	***16/41***
79	KOWOSB	4	35/134	**	27/78	1759/3569	***12/36***	33/37
80	LIARWHD	36	30/78	87/453	33/223	156/292	63/1174	***18/19***
81	LOGHAIRY	2	19/77	***6/11***	7/19	7288/14431	19/52	41/168
82	MANCINO	100	23/38	16/39	15/39	12/27	19/65	***6/9***
83	MARATOSB	2	7/14	7/14	7/14	***5/6***	**5/11**	**5/7**
84	MEXHAT	2	47/262	195/1740	19/88	1448/2997	**11/37**	***11/16***
85	MODBEALE	10	34/97	32/107	**30/95**	2260/7548	F	*60/63*
86	MOREBV	10	26/65	64/272	***16/34***	2372/5544	48/175	*32/34*
87	NONCVXU2	10	*23/38*	**21/44**	25/63	243/264	37/92	35/38
88	NONCVXUN	10	23/39	***19/31***	20/45	84/106	87/208	31/34
89	NONDIA	10	18/49	F	F	889/1298*	2/5*	***14/24***
90	NONDQUAR	100	2004/12555	**	***40/122***	2760/17671	F	375/380
91	NONMSQRT	9	61/213	**	88/384	**	***29/83***	F
92	OSBORNEA	5	39/211	**	38/178	145/459	**25/81**	**37/46**
93	OSBORNEB	11	65/240	625/4287	63/212	728/1792	***24/48***	67/78
94	OSCIGRAD	10	33/56	49/268	**20/60**	95/221	F	*43/44*
95	OSCIPATH	10	8/33	***5/31***	6/35	F	F	**
96	PALMER1C	8	43/192	5228/42136	***15/43***	1363/3174	29/84	36/44
97	PALMER1D	7	33/151	758/6305	***11/35***	2409/5616	114/242	62/66
98	PALMER2C	8	55/244	**	14/39	1401/3678	10/34*	60/64*
99	PALMER3C	8	53/213	**	***14/35***	8779/23111	58/157	56/60
100	PALMER4C	8	56/227	**	***13/35***	**	24/88	56/60

	Methods		BFGSimp	DFPimp	SR1imp	DiagHess	ApproxHess	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
101	PALMER5C	6	16/24	13/19	***8/14***	38/52	36/95	14/16
102	PALMER6C	8	65/294	**	15/39	**	10/34*	43/51*
103	PALMER7C	8	62/252	**	15/41	**	5/11*	28/35*
104	PALMER8C	8	61/255	**	15/40	**	5/12*	49/54*
105	PENALTY1	10	204/1064	**	143/709*	**	**10/21**	*16/17*
106	PENALTY2	200	115/1929*	159/2022*	55/295*	25/1778*	43/157*	290/292*
107	PENALTY3	50	47/100	104/432	20/105*	138/209	36/117*	40/65
108	PFIT1LS	3	F	**	93/284	5472/12233	**	33/43
109	PFIT2LS	3	***53/300***	**	36/126*	6902/12929*	F	476/644
110	PFIT3LS	3	68/413*	**	107/446*	8266/15900*	F	649/904
111	PFIT4LS	3	75/484*	**	27/111*	9849/18130	F	969/1296
112	POWELLSG	4	54/197	232/1239	34/96	3028/10294	215/486	***33/34***
113	POWER	10	160/755	**	68/272	***12/16***	21/52	77/78
114	QUARTC	25	489/2337	**	73/230	**27/58**	F	*56/57*
115	ROSENBR	2	**31/100**	377/2322	44/148	3336/7002*	836/1702	*36/46*
116	S308	2	**14/26**	22/47	14/28	19/27	17/36	*16/17*
117	SBRYBND	10	1/35*	1/35*	1/35*	3/4*	1/35*	267/299*
118	SCHMVETT	10	24/36	17/30	17/27	86/181	F	***14/21***
119	SENSORS	100	***22/31***	**22/32**	27/118*	46/58	40/142	26/34
120	SINEVAL	2	73/279	**	115/458	**	2/7*	***63/86***
121	SINQUAD	5	13/26	**10/19**	11/24	62/161	42/107	*11/17*
122	SISSER	2	*4/10*	*4/10*	*4/10*	13/34	**3/11**	9/11
123	SNAIL	2	99/317	**	137/485	653/1185	**79/231**	*94/127*
124	SPARSINE	10	24/40	17/26	***13/19***	128/188	100/297	36/37
125	SPARSQUR	10	88/367	7223/52362	56/169	**14/54**	18/49	*36/37*

	Methods		BFGSimp	DFPimp	SR1imp	DiagHess	ApproxHess	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
126	SROSENBR	10	25/71	72/393	23/75	2869/6182	28/86	***16/18***
127	TESTQUAD	1000	**	**	**	***7/20***	***7/20***	259/260
128	TOINTGOR	50	70/121	*47/75*	**42/86**	653/924	F	137/138
129	TOINTGSS	10	29/47	***20/36***	22/61	622/782	F	25/39
130	TOINTPSP	50	38/78	102/424	**35/101**	253/449	42/147	*40/62*
131	TOINTQOR	50	36/46	*28/36*	**27/37**	86/104*	F	40/41
132	TQUARTIC	10	14/50	50/241	16/37	427/703	F	***12/18***
133	TRIDIA	10	31/48	19/26	***11/16***	97/211	70/183	21/22
134	VARDIM	200	55/568*	99/1110*	8/98*	20/52*	11/30	***1/2***
135	VAREIGVL	50	33/39	32/38	84/97*	48/89	7317/7319	***20/21***
136	WATSON	12	68/350	**	**39/142**	**	F	*73/77*
137	WOODS	100	505/1260	**	453/1325	5627/10667	132/299	***35/43***
138	YFITU	3	**33/107**	274/2718	29/147	957/4364	62/178	*72/93*
139	ZANGWIL2	2	2/6	2/6	2/6	12/49	2/6	***1/2***

Appendix C

CUTEr Test Results for Hybrid Methods

The following table presents the numerical results obtained for the different algorithms. The stopping criterion used are given by the inequality $\|g(x_k)\|_2 < 10^{-6}$ or $\|x_{k+1}\| < 10^{-6}(1 + \|x_k\|)$. All computations were performed on Intel 2.60 GHz processors running Linux and Matlab R2010a. The test problems range from 2 to 1000 variables. The following notation were used:

- n : The number of variables in the problem .
- Its/Fcs: The number of iterations / function calls.
- "*": Signify a possible minimum has been reached but cannot be certain because the first order optimality condition measure is not fully satisfied.
- "**": Means that the maximum number of iterations has been reached.
- "F" :The routine has failed to solve the problem.
- We highlight in bold the best performance measured in terms of the numbers of iterations. The best performance in terms of function evaluations by italics and we underline the entry which gives the smallest sum of iterations and function calls.

	Methods		HNNimp1	HDBFGS	BFGSimp	Nimp1	TR	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
1	AKIVA	2	***6/7***	F	12/41	9/22	***6/7***	19/23
2	ALLINITU	4	**7/14**	10/18	11/20	**7/14**	*10/11*	11/13
3	ARGLINA	200	***1/2***	**1/3**	2/7	2/7	2/3	2/3
4	ARWHEAD	100	**6/19**	9/22	9/24	**6/16**	***6/7***	**6/8**
5	BARD	3	***11/15***	25/36	21/80	12/31	192/193	20/22
6	BDQRTIC	100	10/23	102/538	81/150	12/31	***9/10***	129/131
7	BEALE	2	*7/9*	15/24	13/22	***6/9***	9/10	15/16
8	BIGGS6	6	44/108	**41/60**	44/202	53/180	***41/42***	46/48
9	BOX	10	***3/4***	6/13	7/16	***3/4***	***3/4***	5/8
10	BOX3	3	***8/10***	20/24	23/80	11/34	13/14	24/25
11	BRKMCC	2	***3/4***	**3/13**	7/13	5/8	***3/4***	5/8
12	BROWNAL	200	9/14	33/61	10/23	7/11	7/8	***2/4***
13	BROWNBS	2	4/38*	131/571*	3/43*	8/36	7075/7076	14/25
14	BROWNDEN	4	***8/9***	19/76	21/48	**8/20**	11/12	32/33
15	BROYDN7D	10	***11/15***	41/96	36/67	14/31	17/18	23/27
16	BRYBND	10	*12/15*	116/190	44/89	**11/24**	16/17	27/30
17	CHAINWOO	4	41/77	27/63	58/209	56/144	208/209	***35/43***
18	CHNROSNB	50	**40/81**	160/515	211/416	48/170	*64/65*	240/248
19	CLIFF	2	27/28	4/31*	62/641	28/266	26/27	***1/2***
20	COSINE	10	11/20	18/52	17/39	11/20	***9/10***	12/17
21	CRAGGLVY	4	17/34	52/61	73/287	17/67	***14/15***	54/62
22	CUBE	2	**20/40**	31/51	39/133	31/93	*31/32*	34/40
23	CURLY10	100	13/24	110/678	228/479	**12/29**	*17/18*	920/931
24	CURLY20	100	**14/28**	112/681	177/392	**14/32**	*17/18*	863/874
25	CURLY30	100	26/57	119/701	149/338	**15/34**	*17/18*	744/753

	Methods		HNNimp1	HDBFGS	BFGSimp	Nimp1	TR	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
26	DECONVU	61	**21/43**	134/263	127/338	**21/43**	***21/22***	80/81
27	DENSCHNA	2	6/7	8/12	10/17	6/9	***5/6***	10/11
28	DENSCHNB	2	***5/7***	7/11	8/13	7/12	*6/7*	7/8
29	DENSCHNC	2	10/11	16/30	17/46	**9/22**	*10/11*	21/24
30	DENSCHND	3	33/52	101/161	89/403	30/78	257/258	***23/28***
31	DENSCHNE	3	20/39	27/53	18/53	20/39	***10/11***	37/58
32	DENSCHNF	2	***6/7***	12/27	9/13	**6/9**	***6/7***	10/11
33	DIXMAANA	15	**7/23**	*10/15*	15/51	***7/15***	8/9	15/16
34	DIXMAANB	15	**7/12**	14/19	16/24	**7/16**	***7/8***	22/23
35	DIXMAANC	15	**8/14**	45/78	23/35	9/19	***8/9***	26/27
36	DIXMAAND	15	**9/17**	17/26	27/51	10/21	***9/10***	30/31
37	DIXMAANE	15	7/13	21/25	26/57	8/17	***6/7***	71/72
38	DIXMAANF	15	10/19	22/27	35/85	**9/20**	***9/10***	74/75
39	DIXMAANG	15	11/20	65/101	38/102	11/22	***10/11***	81/82
40	DIXMAANH	15	12/24	106/175	41/117	11/25	***10/11***	84/85
41	DIXMAANI	5	9/19	36/40	62/212	10/23	***6/7***	97/98
42	DIXMAANJ	5	14/29	58/76	62/212	13/35	***12/13***	105/106
43	DIXMAANK	15	**13/28**	84/120	70/280	**13/32**	***13/14***	118/119
44	DIXMAANL	15	**11/23**	128/200	75/296	12/33	*13/14*	131/132
45	DIXON3DQ	10	***1/2***	18/27	23/39	3/7	20/21	20/21
46	DJTL	2	17/40*	135/293*	20/142*	5558/54577	103/104	2189/5715
47	DQDRTIC	10	***1/2***	12/51	26/67	4/10	***1/2***	14/16
48	DQRTIC	10	23/48	134/186	207/971	23/48	***15/16***	39/40
49	EDENSCH	36	*12/13*	73/217	39/53	**11/27**	*12/13*	69/71
50	EG2	1000	6/7	4/14	7/9	5/7	***3/4***	4/5

	Methods		HNNimp1	HDBFGS	BFGSimp	Nimp1	TR	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
51	ENGVAL1	2	*7/8*	13/21	11/21	**6/9**	*7/8*	11/12
52	ENGVAL2	3	**15/18**	30/56	32/98	18/40	108/109	29/34
53	ERRINROS	50	***26/51***	228/699	189/449	134/372	56/57	329/352
54	EXPFIT	2	***6/10***	13/24	12/29	8/18	11/12	12/16
55	EXTROSNB	10	***23/41***	23/119	320/1271	469/1895	330/331	325/403
56	FLETCBV2	10	***2/3***	15/18	15/25	***2/3***	3/4	15/18
57	FLETCBV3	10	868/1553	130/230	244/1189	855/1530	298/299	***35/78***
58	FLETCHBV	10	1322/2398	52/120	***37/60***	1374/2594	334/335	96/150
59	FLETCHCR	10	***19/39***	62/136	72/178	25/71	131/132	51/70
60	FMINSRF2	16	12/24	33/36	39/100	13/25	***10/11***	27/42
61	FMINSURF	16	**10/18**	28/30	31/76	13/22	*11/12*	27/47
62	GENROSE	500	***33/68***	1290/5479	**	299/889	614/615	876/1431
63	GROWTHLS	3	762/1529	**1/14**	20/96	799/1703	**	12/13*
64	GULF	3	36/74	***10/19***	37/198	42/126	301/302	50/58
65	HAIRY	2	23/43	***17/41***	20/43	76/144	91/92	20/42
66	HATFLDD	3	***19/24***	24/31	32/110	22/66	114/115	**19/27**
67	HATFLDE	3	19/38	32/45	29/117	21/49	15/16	***9/12***
68	HEART8LS	8	*89/179*	1487/1988	186/826	**88/185**	131/132*	2921/3815
69	HELIX	3	20/39	31/51	24/62	***12/22***	748/749	29/31
70	HIELOW	3	18/38*	F	15/49	***7/17***	10/11	28/31
71	HILBERTA	2	***1/2***	6/7	10/22	4/8	***1/2***	5/7
72	HILBERTB	10	***1/2***	13/27	15/26	2/5	4/5	6/7
73	HIMMELBB	2	21/49	7/31	8/18	21/49	13/14	3/8*
74	HIMMELBF	4	1944/2926	**29/54**	36/203	1943/2934	201/202	*48/50*
75	HIMMELBG	2	***4/5***	8/12	8/14	***4/5***	7/8	8/10

	Methods		HNNimp1	HDBFGS	BFGSimp	Nimp1	TR	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
76	HIMMELBH	2	***4/5***	7/10	8/14	***4/5***	7/8	7/8
77	HUMPS	2	611/972	154/334	162/721	871/1445	5459/5460	***104/320***
78	JENSMP	2	10/23	1/11	17/45	12/34	***9/10***	16/41
79	KOWOSB	4	21/44	29/32	35/134	**10/23**	*21/22*	33/37
80	LIARWHD	36	***10/11***	20/65	30/78	**10/20**	***10/11***	18/19
81	LOGHAIRY	2	606/938	***7/13***	19/77	782/1506	514/515*	41/168
82	MANCINO	100	9/26	101/1895	23/38	10/28	15/16	6/9*
83	MARATOSB	2	422/843	8/46	**7/14**	7/8*	779/780	5/7*
84	MEXHAT	2	19/28	13/71	47/262	12/44*	29/30	11/16*
85	MODBEALE	10	***10/16***	F	34/97	**10/21**	43/44	60/63
86	MOREBV	10	***2/3***	18/38	26/65	3/5	41/42	32/34
87	NONCVXU2	10	**11/24**	24/43	23/38	15/32	*17/18*	35/38
88	NONCVXUN	10	**14/24**	23/46	23/39	15/26	*18/19*	31/34
89	NONDIA	10	24/48	29/62	18/49	24/48	***14/15***	**14/24**
90	NONDQUAR	100	***17/18***	1255/1278	2004/12555	**17/133**	26/27	375/380
91	NONMSQRT	9	***28/49***	305/424	61/213	61/295	88/89	F
92	OSBORNEA	5	***14/28***	49/87	39/211	35/157	56/57	37/46
93	OSBORNEB	11	26/55	54/86	65/240	**21/53**	*30/31*	67/78
94	OSCIGRAD	10	13/21	1626/2603*	33/56*	12/26*	17/18*	43/44*
95	OSCIPATH	10	12/30	4/57	8/33	5/15	***2/3***	**
96	PALMER1C	8	***1/2***	32/93	43/192	11/32	**	36/44*
97	PALMER1D	7	***1/2***	24/76	33/151	9/27	**	62/66*
98	PALMER2C	8	***1/2***	41/90	55/244	11/32	**	60/64*
99	PALMER3C	8	***1/2***	40/82	53/213	10/30	**	56/60*
100	PALMER4C	8	***1/2***	39/84	56/227	9/29	**	56/60*

	Methods		HNNimp1	HDBFGS	BFGSimp	Nimp1	TR	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
101	PALMER5C	6	***1/2***	13/31	16/24	2/7	8/9	14/16
102	PALMER6C	8	***1/2***	48/82	65/294	8/30	**	43/51*
103	PALMER7C	8	***1/2***	44/83	62/252	7/28	**	28/35*
104	PALMER8C	8	***1/2***	43/78	61/255	9/29	**	49/54*
105	PENALTY1	10	24/37	137/209	204/1064	41/120	***15/16***	16/17
106	PENALTY2	200	10/11	251/5873*	115/1929*	11/42	1/2*	290/292*
107	PENALTY3	50	22/46*	75/615	47/100	***15/37***	18/19	40/65
108	PFIT1LS	3	23/45*	F	F	321/1350	**	33/43
109	PFIT2LS	3	20/42*	F	53/300	121/462	35/36	476/644
110	PFIT3LS	3	20/41*	F	68/413*	107/437	33/34*	648/904
111	PFIT4LS	3	20/41*	F	75/484*	229/951	43/44*	969/1296
112	POWELLSG	4	***17/18***	29/52	54/197	**17/66**	23/24	33/34
113	POWER	10	19/20	85/167	160/755	18/60	***15/16***	77/78*
114	QUARTC	25	27/58	236/452	489/2337	27/58	***19/20***	56/ 57*
115	ROSENBR	2	19/38*	34/54	31/100	25/66	*27/28*	36/46
116	S308	2	*9/11*	14/23/46	14/26	**8/15**	*10/11*	16/17
117	SBRYBND	10	10/38*	1/45*	1/35*	41/145*	215/216*	267/299*
118	SCHMVETT	10	***3/4***	16/45	24/36	4/6	4/5	14/21
119	SENSORS	100	21/42	71/431	22/31	21/42	***16/17***	26/34
120	SINEVAL	2	20/39*	76/117	73/279	59/184	55/56	63/86
121	SINQUAD	5	*8/11*	11/24	13/26	**7/13**	11/12	11/17
122	SISSER	2	14/15	18/22	***4/10***	13/39	11/12	9/11
123	SNAIL	2	***23/43***	12/15	99/317	65/178	104/105	94/127
124	SPARSINE	10	10/15	22/66	24/40	10/15	***7/8***	36/37
125	SPARSQUR	10	15/16	93/107	88/367	15/46	***12/13***	36/37

	Methods		HNNimp1	HDBFGS	BFGSimp	Nimp1	TR	QN
	Functions	Var	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
126	SROSENBR	10	***8/9***	21/70	25/71	10/22	***8/9***	16/18
127	TESTQUAD	1000	***1/2***	357/6152	**	7/20	2/3	259/260*
128	TOINTGOR	50	*7/8*	47/151	70/121	**6/13**	28/29	137/138
129	TOINTGSS	10	***1/2***	**1/3**	29/47	10/18	14/15	25/39
130	TOINTPSP	50	19/39*	65/146	38/78	**14/46**	*28/29*	40/62
131	TOINTQOR	50	***1/2***	50/129	36/46	3/7	10/11	40/41
132	TQUARTIC	10	***1/2***	11/23	14/50	10/18	**1/2**	12/18
133	TRIDIA	10	***1/2***	16/62	31/48	3/6	5/6	21/22
134	VARDIM	200	28/31	37/88	55/568*	28/292*	29/30	***1/2***
135	VAREIGVL	50	56/72	50/144	33/39	39/43	***15/16***	20/21
136	WATSON	12	17/43	51/350	68/350	17/43	***13/14***	73/77
137	WOODS	100	42/79*	278/1022	505/1260	53/150	58/59	***35/43***
138	YFITU	3	21/38*	68/110	33/107	44/200	3119/3120	72/98
139	ZANGWIL2	2	***1/2***	2/3	2/6	2/6	***1/2***	***1/2***

Appendix D

Specially Selected Test Problems

The following test problems are used

T1: $F = x_1x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$. Starting point $x_0 = (2.05, 1.6)^T$

T1r: $\phi = x_1x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$. $F = -(10 + \phi(x_1, x_2))^{-1}$. Starting point $x_0 = (2.05, 1.6)^T$

T1r2: $\phi = x_1x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$. $F = -(10 + \phi(x_1, x_2))^{-2}$. Starting point $x_0 = (2.05, 1.6)^T$

T1a: $F(x_1, x_2) = x_1x_2 + 0.01 \max\{0, (x_1^2 + 2x_2^2 - 10)\}^2$. Starting point $x_0 = (2.05, 1.6)^T$

T1b: $F(x_1, x_2) = x_1x_2 + 0.01 \max\{0, (x_1^2 + 2x_2^2 - 10)\}^2$. Starting point $x_0 = (0.26, 0.16)^T$

T1ar: $\phi(x_1, x_2) = x_1x_2 + 0.01 \max\{0, (x_1^2 + 2x_2^2 - 10)\}^2$. $F(x_1, x_2) = -(10 + \phi(x_1, x_2))^{-1}$. Starting point $x_0 = (0.26, 0.16)^T$

T2: $F(x_1, x_2) = x_1x_2 + 0.001(x_1^2 + 2x_2^2 - 10)^4$. Starting point $x_0 = (2.5, 1.6)^T$

T2r: $\phi(x_1, x_2) = x_1x_2 + 0.001(x_1^2 + 2x_2^2 - 10)^4$. $F(x_1, x_2) = -(10 + \phi(x_1, x_2))^{-1}$. Starting point $x_0 = (2.5, 1.6)^T$

T3: $F(x_1, x_2, x_3) = x_1x_2x_3 + 0.01(x_1^2 + 2x_2^2 + 3x_3^2 - 10)^2$. Starting point $x_0 = (0.4, 0.3, 0.2)^T$

T4(n): $F = (1 + x^TQx)^{-1}$ where $Q = H + 0.01I$ where H is the $(n \times n)$ Hilbert matrix. Using the initial condition $x_0 = (3, 3, ..., 3)^T$

T4r(n): $\phi(x) = (10 + x^T Q x)^{-1}$ where $Q = H + 0.01I$ where H is the $(n \times n)$ Hilbert matrix. $F = -(1 + \phi(x))^{-1}$ Using the initial condition $x_0 = (3, 3, ..., 3)^T$

T5: $F(x_1, x_2) = x_1^3 + (x_1^2 + 2x_2^2 - 10)^2$. Starting point $x_0 = (-1, 0.1)^T$

T5a: $F(x_1, x_2) = x_1^3 + (x_1^2 + 5x_2^2 - 10)^2$. Starting point $x_0 = (-1, 0.1)^T$

Bibliography

[1] F. S. Acton. *Numerical Methods that Usually Work*. The Mathematical Association of America, Washington, D.C., 1990 (updated from 1970 edition). [Introductory numerical methods book].

[2] M. Al-Baali and R. Fletcher. Variational methods for nonlinear least squares. *J. Oper. Res. Soc.,*, 36:405–421, 1985.

[3] Kenneth Joseph Arrow, Leonid Hurwicz, and Hirofumi Uzawa. *Studies in linear and non-linear programming*. Stanford University Press, 1972.

[4] M. Bartholomew-Biggs. *Nonlinear Optimization with Financial Applications*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2005. [Advanced text book on optimization techniques with financial Applications].

[5] M. C. Bartholomew-Biggs, S. Beddiaf, and S. J. Kane. Traversing non-convex regions. *Advanced Modeling and Optimization*, 15(2), 2013.

[6] S. Beddiaf. Continuous steepest descent path for traversing non-convex regions. *Advanced Modeling and Optimization*, 11(1), 2009.

[7] W. Behrman. *An Efficient Gradient Flow Method for Unconstrained Optimization*. PhD thesis, Stanford University, 1998.

[8] P. T. Boggs, R. H. Byrd, and R. B. Eds. Schnabel. Numerical optimization. *SIAM*, 1984. [A Collection of papers presented at SIAM optimization conference, varied and interesting approaches].

[9] C. A. Botsaris. A curvilinear optimization method based upon iterative estimation of the eigensystem of the hessian matrix. *J. Maths. Anal. Appl.*, 63(2):396–411, 1978.

[10] C. A. Botsaris. Differential gradient methods. *J. Maths. Anal. Appl.*, 63(1):177–198, 1978.

[11] C. A. Botsaris and D. H. Jacobson. A newton-type curvilinear search method for optimization. *J. Maths Anal.Appl.*, 54(1):217–229, 1976.

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University press, 2004.

[13] M. A. Branch, T. F. Coleman, and Y. Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999.

[14] F. H. Branin and S. K. Hoo. A method for finding multiple extrema of a function of n variables. *Numerical Methods*, pages 231–237, 1972.

[15] A. A. Brown. *Optimisation Methods Involving the Solution of Ordinary Differential Equations*. PhD thesis, Hatfield Polytechnic, 1986.

[16] A. A. Brown and M. C. Bartholomew-Biggs. Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations. *J. Optim. Theory Appl.*, 62(2):211–224, 1989.

[17] C. G. Broyden. The convergence of a class of double rank minimization algorithms: 2. the new algorithm. *J. Inst. Math. Appl.*, 6:222–231, 1970.

[18] C. G. Broyden and M. T. Vespucci. Krylov solvers for linear algebraic systems. *Studies in Computational Mathematics*, 11, 2004.

[19] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. Approximate solution of the trust region problem by minimization over two dimensional subspaces. *Mathematical Programming*, 40:247– 263, 1988.

[20] C. Cartis, N. I. M. Gould, and P. L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. part i: motivation, convergence and numerical results. *Mathematical Programming*, 127(2):245–295, 2011.

[21] C. Cartis, N. I. M. Gould, and P. L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. part ii: worst-case function-and derivative-evaluation complexity. *Mathematical programming*, 130(2):295–319, 2011.

[22] P. G. Ciarlet. *Introduction to Numerical Linear Algebra and Optimization*. Cambridge University press, Cambridge, Great Britain, 1989. [Advanced article describing preconditioned conjugate gradient applications].

[23] A. R. Conn, N. I. M. Gould, and P. T. Toint. Trust region methods. *MPS-SIAM Series on Optimization, Philadelphia*, 2000.

[24] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Convergence of quasi-newton matrices generated by the symmetric rank one update. *Mathematical Programming*, 50:177–196, 1991.

[25] F. E. Curtis, D. P. Robinson, and M. Samadi. A trust region algorithm with a worst-case iteration complexity $O(\epsilon^{-3/2})$ for nonconvex optimization. Technical Report COR@L Technical Report 14T-009, Lehigh University, Bethlehem, PA, USA, October 2014.

[26] Y. H. Dai. Convergence properties of the bfgs algorithm. *SIAM Journal on Optimization*, 13:3:693–701, 2002.

[27] W. C. Davidon. Variable metric methods for minimization. *SIAM J. Optim.*, 1:1–17, 1991.

[28] Yang Ding, Enkeleida Lushi, and Qingguo Li. Investigation of quasi-newton methods for unconstrained optimization. *Simon Fraser University, Canada*, 2004.

[29] L. C. W. Dixon and G. P. Zsego. *Towards Global Optimization- Theory and Application*. John Wiley, New York, (eds) 1975.

[30] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[31] J. Farlow, J. E. Hall, J. M. McDill, and B. H. West. *Differential Equations and Linear Algebra*. Pearson Prentice Hall, second edition edition, 2007.

[32] R. Fletcher. A new approach to variable metric algorithms,. *Comput. J.*, 13:317–322, 1970.

[33] R. Fletcher. *Practical Methods of Optimization*. A Wiley-Interscience Publication), John Wiley and Sons, Tiptree, Essex, Great Britain, second edition edition, 1987. [Advanced book on optimization techniques].

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

[35] A. Del Gatto. *A Subspace Method Based on a Differential Equation Approach to Solve Unconstrained Optimization Problems*. PhD thesis, Stanford University, June 2000.

[36] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, 1983. [Standard reference book on practical optimization methods].

[37] D. Goldfarb. A family of variable metric methods derived by variational means. *Math. Comp.*, 24:23–26, 1970.

[38] N. I. M. Gould, D. Orban, and Ph. L. Toint. Cuter and sifdec: A constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, December 2003.

[39] J. Hald and K. Madsen. Combined lp and quasinewton methods for minmax optimization. *Math. Programming*, 20:49–62, 1981.

[40] C. Hamaker. Optimization of a constrained quadratic function. *Studies in Informatics and Control*, 18(1), 2009.

[41] Desmond J. Higham. Trust region algorithms and timestep selection. *SIAM Journal on Numerical Analysis*, 37(1):194–210, 1999.

[42] J. E. Dennis Jr and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, volume 16. Siam, 1996.

[43] J. E. Dennis Jr and R. B. Eds Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs,, New Jersey, 1983. [Advanced optimization book, providing theoretical background for algorithms].

[44] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Massachusetts, second edition edition, 1984. [Introductory optimization book, providing easy to grasp explanations of algorithmic strategies].

[45] X. L. Luo, C. T. Kelly, L. Z. Liao, and H. M. Tam. Combining trust region techniques and rosenbrock methods. *AJ Optim Theory Appl*, 140:265–286, 2009.

[46] MATLAB. *Matlab, Optimization Toolbox, Version 7.10.0 (R2010a), www.mathworks.com*. The MathWorks Inc. Natick, Massachusetts, 2010.

[47] D. G. Mohr. *Hybrid Runge-Kutta and Quasi-Newton Methods for Unconstrained Nonlinear Optimization*. PhD thesis, Graduate College of The University of Iowa, July 2011.

[48] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[49] J. L. Morales and J. Nocedal. Enriched methods for large-scale unconstrained optimization. *Computational Optimization and Applications*, 21:143154, 2002.

[50] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.

[51] Y. Murata. *Mathematics for Stability and Optimization of Economic Systems*. Academic Press (New York), 1977.

[52] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[53] Y. Ou, Q. Zhou, and H. Lin. An ode-based trust region method for unconstrained optimization problems. *Journal of Computational and Applied mathematics*, 232:318–326, 2009.

[54] Y. G. Ou. A superlinearly convergent ode-type trust region algorithm for nonsmooth nonlinear equations. *Journal of Applied mathematics and Computing*, 22(3):371–380, 2006.

[55] J. p. Vial and I. Zang. Unconstrained optimization by approximation of the gradient path. *Maths. Oper. Res.*, 2(3):253–265, 1977.

[56] M. J. D. Powell. A hybrid method for nonlinear equations. *in Numerical Methods for Nonlinear Algebraic Equations*, 1970.

[57] M. J. D. Powell. How bad are the bfgs and dfp methods when the objective function is quadratic? *Math. Programming*, 34:118–122, 1990.

[58] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Math. Comp.*, 24:647–650, 1970.

[59] Z. J. Shi and J. Shen. A new descent algorithm with curve search rule. *Applied Mathematics and Computation*, 161, 2005.

[60] P. Wolfe. Convergence conditions for ascent methods. *SIAM Rev.,*, 11:226–235, 1969.

[61] P. Wolfe. Convergence conditions for ascent methods ii: Some corrections. *SIAM Rev.,*, 13:185–188, 1971.

[62] I. Zang. A new arc algorithm for unconstrained optimisation. *Mathematical Programming*, 15, 1998.

[63] A. K. Zghier. *The Use of Differential Equation in Optimization*. PhD thesis, Loughborough University of Technology, 1981.