A comparison of methods for traversing regions of non-convexity in optimization problems

Michael Bartholomew-Biggs*, Salah Beddiaf† and Bruce Christianson‡

September 2019

Abstract

This paper considers the well-known problem of dealing with non-convexity during the minimization of a nonlinear function $f(x)$ by Newton-like methods. The proposal made here involves a curvilinear search along an approximation to the continuous steepest descent path defined by the solution of the differential equation

$$\frac{dx}{dt} = -\nabla f(x).$$

The algorithm we develop and describe has some features in common with trust region methods; and we present some numerical experiments in which its performance is compared with other ODE-based and trust region methods.
Keywords: Non-convex optimization, ODE-methods, Continuous steepest-descent path (CSDP), trust-region, Newton-like methods, curvilinear search.

1 Introduction and overview of the paper

In this paper we revisit a familiar problem in nonlinear optimization. Many – probably most – iterative optimization methods are based on the use of an underlying convex quadratic model function which is usually very successful at approximating the behaviour of a general nonlinear function in the neighbourhood of a minimum. But difficulties can arise when such an iterative algorithm is applied to a function $f(x)$ which is not convex for all values of x. If the search enters a region where f is non-convex then the underlying assumptions of, for instance, a Newton or quasi-Newton method do not hold and subsequent iterations will not necessarily be very

*Corresponding Author: m.bartholomew-biggs@herts.ac.uk School of Physics, Astronomy and Mathematics, University of Hertfordshire, Hatfield, UK

†School of Physics, Astronomy and Mathematics, University of Hertfordshire, Hatfield, UK

‡School of Engineering and Cmputer Science, University of Hertfordshire, Hatfield, UK

effective in traversing the region where the objective function is non-convex and locating any convex basin that lies beyond it. We shall say a more about these issues in the next section.

In this paper we consider a way of traversing such regions of non-convexity by a gradient-flow algorithm which follows (some approximation to) a continuous steepest descent path (CSDP). This idea has often been considered before and its implementation usually involves the construction of an *approximate* CSDP using an iterative scheme which has much in common with a gradient-flow algorithm devised by Behrman [4] and also with a class of well-known trust-region methods. The approach that we propose is unusual, however, in that each iteration performs a *curvilinear* search along the current CSDP. This is an idea which appears not to have been widely tested before and our main purpose in this paper is to investigate whether it has any merit. Some evidence on this point can be found in the work of Higham [16] which proposes an algorithm which essentially uses a curvilinear search for interpolation. However the method we introduce in this paper can also perform *extrapolation* and this is its most significant claim to originality. In practical terms our approach also differs from Higham's method (as described in [16]) because it performs the interpolation phase of the search using only function evaluations whereas the Higham approach treats step-size reduction like a new iteration and computes a new correction "from scratch" involving the additional computational cost of solving a system of linear equations.

It is important to make clear in these introductory remarks both what we are and what we are not seeking to achieve in this paper. Our chief aim is to investigate the effectiveness of a curvilinear search along a CSDP as a means of escaping from regions where the objective function is non-convex. The methods we introduce in section 5 make use of both first and second derivatives of the objective function and furthermore they compute the eigensystem of the Hessian matrix $\nabla^2 f(x)$ on each iteration. The advantages of this for our present purposes are (a) it is very clear when the search enters or leaves a region where f is non-convex; and (b) the curvilinear search can be based on a good deal of accurate curvature information. However the algorithms in section 5 are NOT intended or expected to be useful for practical problems in large numbers of variables because obtaining the Hessian matrix and computing its eigenvalue decomposition will be too expensive. Our aim here is to investigate the behaviour of a curvilinear search under the most favourable circumstances when ample curvature information is available. We see this as a first step towards devising a similar CSDP approach which avoids the high-cost overheads of obtaining a complete matrix of second derivatives and calculating eigenvalues. We shall say more about possible future algorithm development in the final section of this paper.

In Section 2 we give a more detailed discussion of the difficulties presented to a typical optimization technique when non-convexity is encountered. In sections 3 and 4 we

give an overview of how, respectively, CSDP and trust-region approaches provide a systematic way of escaping from such regions. In section 5 we describe two forms of a new CSDP method which incorporates a curvilinear search. We then report some computational experiments in which we compare the performance of these curvilinear search CSDP methods with a pair of trust-region techniques. In section 6 we give a brief discussion of some convergence questions relating to the new CSDP methods. Our conclusions are set out in the final section.

2 Minimization methods and non-convex regions

Many successful iterative methods for solving the unconstrained minimization problem

$$\text{Minimize } f(x)$$

(where $x \in R^n$ and f is a single real valued function assumed to be twice continuously differentiable) are based on constructing a quadratic model of the objective function f. Thus, if x_k is the point reached at the start of the k-th iteration and if f_k denotes $f(x_k)$ while g_k denotes $g(x_k)$, the gradient of f, and G_k is the Hessian matrix, $G(x_k)$, then a quadratic model in the neighbourhood of x_k can be written

$$Q(p) = f_k + p^T g_k + \frac{1}{2} p^T G_k p. \tag{1}$$

A stationary point of Q can then be found by solving $\nabla Q(p) = g_k + G_k p = 0$ and hence the step

$$p = -G_k^{-1} g_k$$

can also be regarded as an estimate of the step from x to a nearby stationary point of f.

If $G(x)$ is positive definite then the stationary point of Q will be a minimum and hence the following iterative scheme can be used to seek a minimum of f.

Algorithm 1
Given an initial point x_1
Repeat for $k = 0, 1, 2...$
Calculate $f_k = f(x_k), g_k = g(x_k), G_k = G(x_k)$
Find p_k by solving $G_k p = -g_k$
Set $x_{k+1} = x_k + s^* p_k$ where s is chosen by a *perfect* or a *weak* line search.

A perfect line search is one where s^* solves the one-variable problem of minimizing $f(x_k + s p_k)$ w.r.t. s. A weak line search merely returns a value s^* which is bounded

away from zero and also yields a reduction in f which is bounded away from zero. Both these requirements on a weak line search can be expressed in terms of the ratio

$$d_k(s) = \frac{f(x_k + sp_k) - f_k}{sp_k^T g_k} \tag{2}$$

which compares the actual reduction in f with a first-order predicted reduction. Clearly $d_k(s)$ tends to 1 as s tends to zero and becomes negative if the steplength s produces an increase in the value of f. Hence a weak line search typically terminates with a value of s^* that satisfies a condition of the form

$$1 - \alpha_1 > d_k(s^*) > \alpha_2 \text{ or } d_k(s^*) > 1 + \alpha_1 \tag{3}$$

for some parameters $\alpha_1, \alpha_2 \in (0, 0.5)$

Algorithm 1 is known as the Newton method and it can be proved to have second order convergence so long as x_1 is chosen in a convex region around a minimum of f. However the method may not be successful if it begins in, or subsequently enters, a region where f is non-convex and hence encounters a point $x_k$where the Hessian G_k is not positive definite. In this situation the search direction p_k may point towards a saddle point, or even a maximum of Q; and then the correction p_k may not be a direction of descent w.r.t. f and the line search procedure may fail. Practical implementations of the Newton method have to include an alternative correction step calculation for use when G_k is not positive definite.

The previous difficulty is at least partially avoided by the so-called quasi-Newton methods which have the same form as Algorithm 1 except that the Hessian G_k is replaced by an approximate matrix B_k, say, which is revised on every iteration. In some implementations of this approach, such as the widely-used BFGS method combined with perfect line searches, the matrix B_k remains positive definite even when the objective function is locally non-convex. In this case, the quasi-Newton search direction is always a direction of descent; but the effectiveness of a quadratic model based on a positive definite estimate of an indefinite Hessian must be questionable. The resulting correction step may turn out to be little better than one which ignores second derivative information altogether – e.g. the steepest descent method which also follows the iterative scheme of Algorithm 1 except that the search direction is given simply by $p_k = -g_k$. There are also other forms of the quasi-Newton approach – such as the Symmetric Rank One method or the BFGS method combined with a weak line search – in which the updating strategy for the approximate Hessian may allow it to become indefinite. (In the case of the Symmetric Rank One method this indefiniteness can occur even when $f(x)$ is locally convex!) Once B_k becomes non-positive definite then the quasi-Newton approach is subject to the same difficulties as Algorithm 1 and needs similar safeguards.

The steepest descent method can converge to a minimum of $f(x)$ even when x_1 is in a region of non-convexity; but the rate of convergence, even close to the solution, can be

very slow. This slowness arises partly because progress to the minimum takes place in piecewise linear steps defined by a sequence of search-direction/line-search calculations. On functions whose Hessian matrix has widely spread eigenvalues (imagine contours that are very long and very narrow ellipses) the minimum is typically approached via a series of very small zig-zags. Because of this, many authors – beginning with Arrow et al [1], and also including Botsaris[5, 6] and Brown [8] – have pointed out the possible benefits of smoothing out the zig-zags by following a Continuous Steepest Descent Path (CSDP) defined as the solution of an initial value problem in which x is regarded as a function of a parameter t.

$$\frac{dx}{dt} = -g(x(t)), \qquad x(0) = x_1. \tag{4}$$

One advantage of the continuous steepest descent path is that it can be followed through both convex and non-convex regions in order to locate a minimum of $f(x)$. For this reason we shall consider some minimization algorithms based on the idea of solving (4).

3 Continuous Steepest Descent Paths

To devise a computational CSDP algorithm for minimizing a general function $f(x)$ we must use an approximate solution of (4) since analytic solutions will not be available. Behrman [4] suggests that, at a point x_k, we consider a linearised version

$$\frac{dx}{dt} = -g_k - G_k(x(t) - x_k) \text{ where } x(0) = x_k. \tag{5}$$

This linearised equation has an exact analytical solution which can be expressed in terms of the eigenvalues and eigenvectors of G_k. Suppose R_k is the matrix whose columns are the normalised eigenvectors of G_k and D_k is the diagonal matrix whose elements are the eigenvalues $\lambda_1, ..., \lambda_n$ (where $\lambda_1 \geq \lambda_2 ... \geq \lambda_n$). Then G_k can be written in decomposed form as

$$G_k = R_k D_k R_k^T$$

and the solution path away from x_k is defined by points on the trajectory

$$p_k(t) = x(t) - x_k = -R_k \Lambda R_k^T g_k \tag{6}$$

where

$$\Lambda_{ii} = \frac{exp(-\lambda_i t) - 1}{\lambda_i} \text{ if } \lambda_i \neq 0 \text{ and } \Lambda_{ii} = t \text{ if } \lambda_i = 0. \tag{7}$$

It is worth noting that if G_k is positive definite then (7) implies

$$\Lambda_{ii} \to -1/\lambda_i \text{ as } t \to \infty$$

Hence $p_k(t) \to -R_k D_k^{-1} R_k^T g_k = -G_k^{-1} g_k$ (the Newton correction) as $t \to \infty$.

The approximate steepest descent path given by (6) and (7) is the basis of Behrman's gradient flow algorithm [4] which we shall discuss again later.

Another way of approximating a solution to (4) involves applying the implicit Euler numerical method to (5). If we take a step t in parameter space away from the point $x_k = x(0)$ then the implicit Euler method gives

$$x(t) = x_k - t(g_k + G_k(x(t) - x_k))$$

which rearranges to

$$(I + tG_k)(x(t) - x_k) = -tg_k$$

and so the approximate solution trajectory is

$$x(t) = x_k + p_k(t) \text{ where } p_k(t) \text{ solves } (I + tG_k)p = -tg_k. \tag{8}$$

The solution path defined by (8) has been considered by many authors including Brown and Bartholomew-Biggs [8, 9] and will also be the basis for an algorithm which is the main object of study in this paper.

If G_k is positive definite then the coefficient matrix of the linear system in (8) is positive definite for all positive t. If G_k is indefinite, then the coefficient matrix of the linear system in (8) is positive definite for all t between zero and $-1/\lambda_n$, where λ_n is the most negative eigenvalue of G_k. This upper bound on t for the non positive definite case also puts a bound on the exponential term in (7) (which could otherwise become unbounded when $\lambda_i < 0$ and t tends to ∞).

A solution to the linear system in (8) can be written in terms of the eigenvectors and eigenvalues of G_k, as in Behrman's gradient flow method [4] If R_k and D_k are defined as in the lead-up to (6), (7) then, since $R_k R_k^T = I$ (because R_k is orthogonal), the linear system in (8) can be written $R_k(I + tD_k)R_k^T p = -tg_k$. Hence $p_k(t)$ can be obtained via the following calculations:

$$p_k(t) = -tR_k \hat{D} R_k^T g_k \text{ where } \hat{D}_{ii} = \frac{1}{1 + t\lambda_i}, i = 1, ..., n. \tag{9}$$

It is important to emphasise that this calculation is different from the calculation in (6) and (7). However when G_k is positive definite (9), like (7), does yield a correction $p_k(t)$ which tends to the Newton direction as t tends to ∞. This is easy to see because the linear system in (8) approaches $G_k p = -g_k$ as $t \to \infty$.

We note also that one would not normally set out to solve a linear system via an (expensive) eigenvalue calculation. However we shall be considering methods in which

we follow a solution trajectory by performing *several* solutions of (6) or (8) for different values of t on each iteration. The once and for all eigensystem calculation means that the second and subsequent solutions are relatively cheap. Before we consider more details of how such iterations might be carried out we need to look at another well-known approach to non-convexity in optimization calculations.

4 Trust region methods

Although a quadratic model like (1) cannot be used to predict a local minimum when the Hessian matrix G_k is not positive definite, it can be used to produce a search direction away from a point x_k as a solution to the constrained subproblem

$$\text{Minimize } Q(p) = f_k + p^T g_k + \frac{1}{2} p^T G_k p \text{ subject to a step limit } p^T p = \Delta_k. \qquad (10)$$

This problem amounts to finding the lowest point of a quadratic hyper-surface on its intersection with a hypersphere of radius $\sqrt{\Delta_k}$ and centred on x_k. Problem (10) always has a solution whether or not G_k is positive definite. Δ_k can be thought of as a *trust-region radius* around x_k – that is, it defines the boundary of a region within which we "trust" our quadratic model to be acceptably accurate. Δ_k is adjusted on each iteration, depending on how well progress on the previous iteration agreed with the quadratic model.

For completeness we note that trust-region methods usually employ a version of (10) in which the equality constraint is replaced by an upper bound on $p^T p$. This is important if (10) is being used to determine a correction step when the Hessian G_k is positive definite and the actual distance to a local minimum is less than the arbitrary step-limit Δ_k. But this situation need not concern us since we are only considering the benefits of a trust-region step in regions where $f(x)$ is non-convex, and so one of our goals is to take a step which leaves the region where the current G_k is a good approximation to the Hessian.

Algorithm 2 below gives an outline of how (10) can be used.

Algorithm 2
Choose an initial point x_1
Choose an initial value $\Delta_1 > 0$ and constants $\beta_1 > 1, \beta_2 \in (0, 1)$
Repeat for $k = 0, 1, 2...$
calculate $f_k = f(x_k), g_k = g(x_k), G_k = G(x_k)$
find p_k by solving problem (10)
set $x_{k+1} = x_k + p_k$ if $f(x_k + p_k) < f_k$
$x_{k+1} = x_k$ if $f(x_k + p_k) \geq f_k$
set $\Delta_{k+1} = \beta_1 \Delta_k$ if the reduction in $f(x)$ has been "good"

$\Delta_{k+1} = \Delta_k$ if the reduction in $f(x)$ has been "acceptable"
$\Delta_{k+1} = \beta_2 \Delta_k$ if the reduction in $f(x)$ has been "unacceptable"

The terms *good, acceptable* and *unacceptable* will be explained more fully later in this section.

A solution to (10) corresponds to a stationary point of the *Lagrangian* function

$$L(p, \mu) = Q(p) + \frac{\mu}{2}(p^T p - \Delta_k)$$

where μ is an unknown *Lagrange multiplier*. Stationarity of the Lagrangian implies

$$\nabla L = g_k + G_k p + \mu p = 0$$

and so

$$(\mu I + G_k)p = -g_k \tag{11}$$

which is equivalent to the linear system in (8) if we make the parameter substitution

$$t = \frac{1}{\mu}.$$

If we assume that $\mu \geq 0$ and $(\mu I + G_k)$ is positive definite then we can say that a step $p_k(\mu)$ which solves the system (11) for an arbitrary choice of μ will also solve (10) for *some* value of Δ_k, in fact for $\Delta_k = p_k(\mu)^T p_k(\mu)$. But the relationship between μ and Δ_k is not a simple one; and once Δ_k has been specified it may be necessary to solve a nonlinear equation in order to find a value of μ to use in (11) and hence to solve (10). It is more usual for a trust region method like Algorithm 2 to obtain p_k by constrained minimization techniques which are outside the scope of this paper. (For a fuller discussion of trust region methods see [13].)

There is, however, a trust region method proposed by Higham [16] which *does* use equation (11) and works directly with the parameter μ rather than dealing with Δ_k explicitly. It follows the spirit of Algorithm 2 by observing that an increase in μ corresponds to a decrease in Δ_k and vice versa. Hence μ is decreased after a step which is judged "good" and increased after a step which is "unacceptable". In Higham's method (as in many other trust-region methods) the quality of a step $p_k(\mu)$ is measured by a comparison with the quadratic model. If we define

$$r_k(\mu) = \frac{f(x_k + p_k(\mu)) - f_k}{Q(p_k(\mu)) - Q(0)} \tag{12}$$

then a value $r_k \approx 1$ indicates that the behaviour of f is in good agreement with the quadratic model. Similarly $r_k \approx 0$ means poor or unacceptable agreement.

We also note that (dropping explicit dependence on μ for clarity)

$$Q(p_k) - Q(0) = p_k^T g_k + \frac{1}{2} p_k^T G_k p_k = \frac{1}{2}[p_k^T g_k + p_k^T g_k + p_k^T G_k p_k]$$

Since p_k solves (11) it follows that

$$Q(p_k) - Q(0) = \frac{1}{2}[p_k^T g_k - \mu p_k^T p_k]$$

If $g_k \neq 0$ and if μ is positive and chosen sufficiently large that $(\mu I + G_k)$ is positive definite then both terms in square brackets on the right hand side are negative and hence the solution to (11) does yield a reduction in the quadratic model function.

In the next section we shall present a modified version of Higham's algorithm (the original can be found in [16]) which will be compared with implementations of the two CSDP methods discussed in Section 3. The CSDP approach based on (6),(7) is Behrman's method [4]; and our own suggested approach based on (8) will henceforth be referred to as Nimp1.

5 A Computational investigation

We now propose some computational algorithms which are developments of methods outlined in Section 2. For consistency with the notation of Higham's trust region method, it will be convenient for us to express the continuous steepest descent calculations in (6), (7) and (8) in terms of $\mu = \frac{1}{t}$. Thus (6) and (7) become

$$p_k(\mu) = -R_k \Lambda R_k^T g_k \tag{13}$$

where

$$\Lambda_{ii} = \frac{exp(-\frac{\lambda_i}{\mu}) - 1}{\lambda_i} \text{ if } \lambda_i \neq 0 \text{ and } \Lambda_{ii} = \frac{1}{\mu} \text{ if } \lambda_i = 0. \tag{14}$$

Similarly (8) becomes

$$x(t) = x_k + p_k(\mu) \text{ where } p_k(\mu) \text{ solves } (\mu I + G_k)p = -g_k \tag{15}$$

and the solution via (9) becomes

$$p_k(\mu) = -R_k \hat{D} R_k^T g_k \text{ where } \hat{D}_{ii} = \frac{1}{\mu + \lambda_i}, i = 1, ..., n. \tag{16}$$

The CSDP algorithms we now propose are designed to perform a search in terms of μ to obtain a new point of the form $x_k + p_k(\mu)$ where p_k may be obtained either from (13) and (14) or from (16). As μ varies, the trial points will trace out a *curvilinear*

path in contrast to the straight line search that usually figures in optimization methods like Algorithm 1. The use of curvilinear searches in terms of μ has been discussed in previous work by the present authors [3], [2]; and in what follows we describe one particular approach in more detail.

We note first of all that if G_k is positive definite with smallest eigenvalue λ_n then the linear system in (15) has a positive definite matrix for all $\mu > -\lambda_n$. On the other hand, if G_k is not positive definite and if λ_n denotes its most negative eigenvalue then the matrix of the linear system in (15) is again positive definite for all $\mu > -\lambda_n$. Hence, if we define μ_{min} as $-\lambda_n$, we have for any value of $\mu > \mu_{min}$ that $p_k(\mu)$ is a descent direction. The purpose of the curvilinear search is to obtain a value μ^* so that the ratio of actual change in f to the linear predicted change

$$d_k(\mu) = \frac{f(x_k + p_k(\mu)) - f_k}{p_k(\mu)^T g_k} \tag{17}$$

is bounded away from one and zero, as in the discussion following Algorithm 1. The same requirement can of course be applied to the search for μ^* when $p_k(\mu)$ comes from (13) and (14).

We now look in more detail at how to carry out a curvilinear search along the path $p_k(\mu)$. For each trial value of μ interpolate– i.e. increase μ – if the current value does not yield an acceptable decrease in f (s measured by the value of $d_k(\mu)$). Similarly we can consider extrapolating – by decreasing μ – if the reduction in f produces a value of d_k that exceeds (or is close to) 1. As a precautionary measure, the curvilinear search procedure set out below uses *both* ratios (17) and (12) in choosing to extrapolate.

The curvilinear search procedure distinguishes between the convex and non convex cases. If G_k is non-positive definite then the starting value of μ is derived from the successful value of μ at the end of the previous iteration. But if G_k is positive definite it seems much more sensible to try the initial value $\mu = 0$ (that is to attempt a Newton step) and to adjust μ only if the Newton step is unsuccessful. All of this is formalised in Algorithm 3 below, in which the step "calculate $p_k(\tilde{\mu})$" may be performed either by (13) and (14) (Behrman's method) or by(16) (Nimp1).

Algorithm 3 (embodying both Behrman's method and Nimp1)
Given an initial point x_1 and an initial trial value μ_1
Choose constants $\nu_1, \nu_2, \eta_1, \eta_2 \in (0, 1)$ with $\nu_1 < \nu_2$ and $\eta_1 < \eta_2$
Choose constants $\alpha_1, \alpha_2 \in (0, 0.5)$
Repeat for $k = 0, 1, 2...$
    calculate $f_k = f(x_k), g_k = g(x_k), G_k = G(x_k)$
    compute the eigensystem decomposition $G_k = R_k D_k R_k^T$
    set $\mu_{min} = -\lambda_n$

if $\mu_{min} > 0$ set $\tilde{\mu} = \max(\mu_k, 2\mu_{min})$
if $\mu_{min} \leq 0$ set $\tilde{\mu} = 0$
calculate $p_k(\tilde{\mu})$ and set $\tilde{x} = x_k + p_k(\tilde{\mu})$
calculate $\tilde{d}_k = d_k(\tilde{\mu})$ from (17) and $\tilde{r}_k = r_k(\tilde{\mu})$ from (12)
if $\mu_{min} > 0$
    repeat while $\tilde{d}_k > 1 - \alpha_1$ and $\tilde{r}_k > \eta_2$ and $\tilde{\mu} > 1.1\mu_{min}$
        replace $\tilde{\mu}$ by $\tilde{\mu} - \nu_2(\tilde{\mu} - \mu_{min})$
        calculate $p_k(\tilde{\mu})$ and set $\tilde{x} = x_k + p(\tilde{\mu})$
        set $\tilde{d}_k = d_k(\tilde{\mu})$ and $\tilde{r}_k = r_k(\tilde{\mu})$
    end
endif
repeat while $\tilde{d}_k < \alpha_2$
    replace $\tilde{\mu}$ by $\tilde{\mu} + \nu_1(\tilde{\mu} - \mu_{min})$
    calculate $p_k(\tilde{\mu})$ and set $\tilde{x} = x_k + p_k(\tilde{\mu})$
    set $\tilde{d}_k = d_k(\tilde{\mu})$ and $\tilde{r}_k = r_k(\tilde{\mu})$
end
set $x_{k+1} = \tilde{x}$ and $\mu_{k+1} = \tilde{\mu}$

In regions where f is non-convex the procedure optionally performs extrapolation steps if the initial move is sufficiently promising. It may also (or instead) perform some interpolation steps if the initial (or an extrapolated) move does not produce an acceptable function decrease. The extrapolation and interpolation loops are distinct; hence, once interpolation has occurred there is no possibility of more extrapolation (which might risk causing an "endless" cycle of interpolations and extrapolations). Christianson [12] has previously pointed out that such separation of the two steplength criteria (i.e. ensuring that a step is neither"too short" nor"too long") is consistent with established convergence theory.

It is worth considering whether an interpolation can 'undo' the benefits of a good extrapolation step. Suppose therefore that $\tilde{\mu}$ has defined a good move which permits further extrapolation using

$$\tilde{\mu}^+ = \tilde{\mu} - \nu_2(\tilde{\mu} - \mu_{min}).$$

Suppose further that $\tilde{\mu}^+$ yields an unacceptable move and that interpolation takes place and provides a new value

$$\tilde{\mu}^- = \tilde{\mu}^+ + \nu_1(\tilde{\mu}^+ - \mu_{min}).$$

We do not want $\tilde{\mu}^-$ to be larger than the good value $\tilde{\mu}$. We can see that

$$\tilde{\mu}^- = \tilde{\mu} - \nu_2(\tilde{\mu} - \mu_{min}) + \nu_1(\tilde{\mu} - \nu_2(\tilde{\mu} - \mu_{min}) - \mu_{min}) = \tilde{\mu} - (\nu_2 - \nu_1 + \nu_1\nu_2)(\tilde{\mu} - \mu_{min})$$

Hence it follows that $\tilde{\mu}^- < \tilde{\mu}$ if the parameters ν_1 and ν_2 are chosen so

$$\nu_2 - \nu_1 + \nu_1\nu_2 > 0.$$

In particular this holds if $\nu_1 = \nu_2$. More generally we need

$$\nu_2 > \frac{\nu_1}{1+\nu_1}.$$

We shall perform some numerical tests with implementations of Algorithm 3 in order to compare the performance of the corrections $p_k(\mu)$ corresponding to (13) and (15). We shall also include in our tests a version of Higham's method (which will be specified below as Algorithm 4). The interpolation procedure in Algorithm 3 is essentially the same as in Higham's method − i.e. it involves rejecting an unacceptable step and calculating a new correction with an increased value of μ. However the algorithm in [16] has to solve a linear system to compute a new correction for each value of μ and Algorithm 4 is able to avoid this because of the eigenvalue decomposition done once and for all on each iteration. Hence the main difference between Algorithm 3 and Algorithm 4 is that the latter does not perform extrapolation but simply carries forward a reduced value of μ for the next iteration whenever the first correction step is judged to be good. Our version of the Higham approach can therefore be stated as follows; and it should be noted that it resembles Algorithm 3 in always taking $\mu = 0$ as the initial choice in a convex region.

Algorithm 4 (modified Higham procedure)
Given an initial point x_1 and an initial trial value μ_1
Choose constants $\nu_1, \nu_2, \eta_1, \eta_2 \in (0,1)$ with $\nu_1 < \nu_2$ and $\eta_1 < \eta_2$
Choose constants $\alpha_1, \alpha_2 \in (0, 0.5)$
Repeat for $k = 0, 1, 2...$
    calculate $f_k = f(x_k), g_k = g(x_k), G_k = G(x_k)$
    compute the eigensystem decomposition $G_k = R_k D_k R_k^T$
    set $\mu_{min} = -\lambda_n$
    if $\mu_{min} > 0$ set $\tilde{\mu} = \max(\mu_k, 2\mu_{min})$
    if $\mu_{min} \leq 0$ set $\tilde{\mu} = 0$
    calculate $p_k(\tilde{\mu})$ and set $\tilde{x} = x_k + p_k(\tilde{\mu})$
    set $\tilde{d}_k = d_k(\tilde{\mu})$ and $\tilde{r}_k = r_k(\tilde{\mu})$
    if $\mu_{min} > 0$
        if $\tilde{d}_k > 1 - \alpha_1$ and $\tilde{r}_k > \eta_2$ and $\tilde{\mu} > 1.1\mu_{min}$
            replace $\tilde{\mu}$ by $\tilde{\mu} - \nu_2(\tilde{\mu} - \mu_{min})$
        end if
    end if
    repeat while $\tilde{d}_k < \alpha_2$
        replace $\tilde{\mu}$ by $\tilde{\mu} + \nu_1(\tilde{\mu} - \mu_{min})$
        calculate $p(\tilde{\mu})$ and set $\tilde{x} = x_k + p(\tilde{\mu})$
        set $\tilde{d}_k = d_k(\tilde{\mu})$ and $\tilde{r}_k = r_k(\tilde{\mu})$
    end
    set $x_{k+1} = \tilde{x}$ and $\mu_{k+1} = \tilde{\mu}$

5.1 Computational results

Using Algorithms 3 and 4 we can compare the performance of the methods Nimp1, Behrman and Higham on a set of 139 small to medium-sized problems from the well-known CUTEr test set [15]. Each of the problems in this test set is provided with a standard starting point and thus provides a basis for a uniform comparison between optimization methods.

Algorithms 3 and 4 have been coded in MATLAB (version R2010a) and computations were performed on Intel 2.60 GHz processors running under Linux. Algorithm 3 implements the two gradent-flow methods, Nimp1 and Behrman, while the version of Higham's method in Algorithm 4 can be regarded as something of a compromise between a trust-region and a gradient-flow approach. In order to extend the comparison we also consider a more conventional trust region method – denoted by TR – which is implemented within the MATLAB optimization toolbox as *fminunc* [17].

The trust region method in *fminunc* is described in [7], [11] and it is comparable with Higham in that it uses the exact Hessian of the objective function and works with a trust region subproblem on every iteration. The approach differs from Higham however in not obtaining an exact solution to the trust region subproblem but rather restricting itself to a two dimensional subspace. This subspace is defined by the negative gradient $-g_k$ together with either an approximate Newton direction, $n \approx -G_k^{-1} g_k$ or a direction of negative curvature s, such that $s^T G_k s < 0$. Obtaining the Newton direction or a direction of negative curvature could involve the solution of the linear system in (15) with $\mu = 0$ or else the calculation of the eigensystem of G_k. However the method in *fminunc* seeks to avoid doing as much work as Higham or Nimp1 on each iteration and hence it finds n or s, by applying a preconditioned conjugate gradient (PCG) method see [10] to the system $G_k n = -g_k$. 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.

In fairness we need to point out that, while *fminunc* is comparable with Algorithms 3 and 4 in using exact Hessian information, it may be at a disadvantage in that it does not employ an eigenvalue decomposition which means that its correction step calculation cannot be based on as much information as is available to Algorithms 3 and 4. Nevertheless we have chosen to use this method as our representative trust-region approach,TR, largely because its MATLAB implementation enables us easily to include it in a testing procedure alongside our own MATLAB codes. We justify this decision by reminding the reader that our purpose in this paper is not to attempt a definitive ranking among competing algorithms but rather to do a feasibility study to see whether our selective use of curvilinear CSDP searches shows any promise.

We present detailed results of our computing experiments in table form in an Appendix which also gives details of convergence and stopping criteria together with the values used for the parameters which control the curvilinear search procedures in Algorithms 3 and 4. The tables show the number of iterations needed to attain the desired accuracy and the corresponding number of function evaluations. These measures both give useful information on a solver's robustness and efficiency. A comparison of run-times might have been interesting too – although possibly misleading because we know that the implementations of Nimp1, Behrman and Higham are very similar whereas TR is a "black box" code and details of its per-iteration computations were invisible to us. Numbers of iterations and function calls are quantities which, to some extent, are independent of the overhead costs of the algorithm itself.

For Algorithms 3 and 4, one "function call" per iteration is required to compute $f(x)$ along with the gradient and the Hessian matrix. All other function calls are simply evaluations of the objective. Nimp1 and Behrman (Algorithm 3) use these extra function calls to perform extrapolation and/or interpolation along the curvilinear search path. For Higham (Algorithm 4) the additional function evaluations relate only to interpolation. It will be noted that TR, being a classical trust-region algorithm, computes only one new point (involving one function call) per iteration. This new point is then accepted or rejected and μ is adjusted appropriately for the next iteration. It may be assumed that this single function call computes the gradient and Hessian as well as the value of $f(x)$.

As we consider the tabulated results in the Appendix we need some criteria for deciding which method has performed best on each problem. Clearly, if one method uses fewer iterations and fewer function calls it can be regarded as the"winner". But how should we judge between two methods where one takes fewer iterations but requires more function calls? Suppose that on a problem involving n variables method A requires I_A iterations and f_A function calls and that the corresponding figures for method B are I_B and f_B. We know that one function call per iteration evaluates the gradient and Hessian and hence can be regarded as being n^2 times more expensive than the other function calls. Hence if $I_A < I_B$ but $f_a > f_B$ then method A can be regarded as more efficient than method B if

$$f_A - f_B < n^2(I_B - I_A)$$

– that is if the additional cheap function calls made by method A during the curvilinear searches are outweighed by the expensive function calls made by method B on its extra iterations. This way of ranking the performance of competing algorithms does of course ignore overhead costs such as the linear algebra performed during each iteration. It is however not unreasonable to ignore such costs in the present case when one of the methods, namely TR, probably has very different overheads from Nimp1, Behrman and Higham. In any case it is not the overheads of the compet-

ing algoritms that we are interested in: instead we are investigating (a) the benefits of adding a curvilinear search to a conventional trust-region framework and (b) the effectiveness of performing extrapolation as well as interpolation during such a search.

When we use the above method of ranking the algorithms we find that the classical trust region approach TR comes out best on just under 25% of the problems considered. This of course means that the use of some sort of curvilinear search is advantageous in around 75% of cases. Moreover the methods Nimp1 & Behrman – both of which use extrapolation – return the best result in 60% of the tests reported in the Appendix. This represents a fairly encouraging endorsement of the curvilinear search we have proposed in this paper. This being said, however, we note from the results in the Appendix that all the tested methods have some shortcomings. There are a number of instances where the methods based on Algorithms 3 and 4 all fail to solve a problem. There are are also cases where *only* Behrman or Higham fail but there do not appear to be any examples where Nimp1 records the only failure. And while TR does not fail or terminate at a non-optimal point there are some cases where it still has not converged within the limiting number of 10000 iterations.

Having discussed the methods in terms of a performance measure of our own devising, we now also seek to draw some overall conclusions about the behaviour of the four methods using the generally recognised yardstick of *performance profiles* as proposed by Dolan & Moré [14]. 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 cumulative 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 [14] that if a method does not solve a problem, $r_{p,s}$ is simply given an arbitrary 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 available 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.

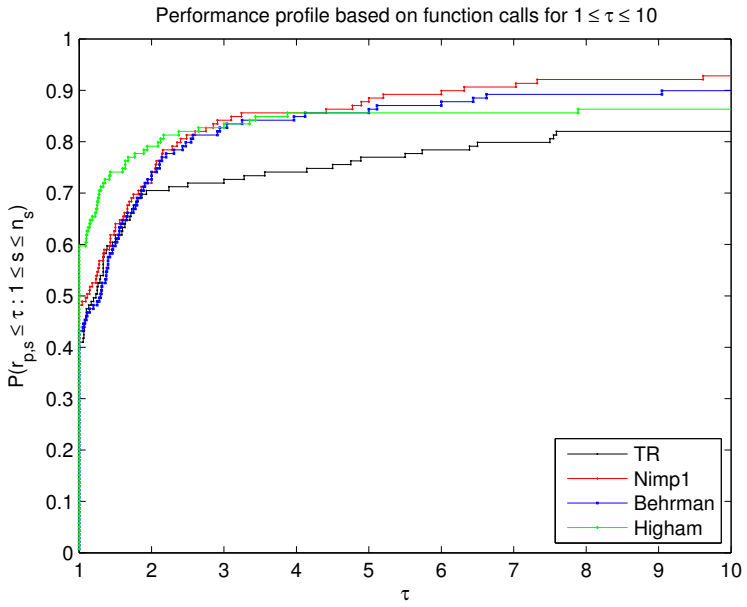

Figure 1: Functions calls for $1 \leq \tau \leq 10$.

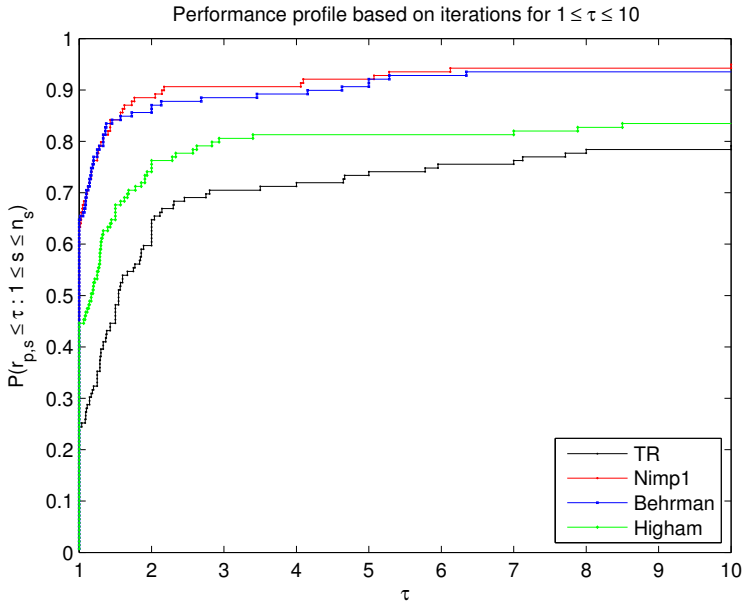

Figure 2: Iterations calls for $1 \leq \tau \leq 10$.

From these graphs we can observe that Nimp1 and Behrman behave in a very similar way, although it is also possible to say that Nimp1 seems to maintain a rather small advantage. When function calls are used as a performance measure Higham does considerably better than the other methods for smaller values of τ. This must reflect the fact that Higham uses only one function call per iteration provided the initial step is an acceptable one. On the other hand it is the absence of extrapolation that probably explains the relatively poor showing of Higham when performance is measured by numbers of iterations needed for convergence. The relative superiority of Higham in terms of function calls only occurs for values of τ less than about 5. Broadly speaking, the effort expended in the curvilinear search by Nimp1 and Behrman seems to pay off in terms of savings in iteration numbers (and the associated linear algebra overheads).

The behaviour of TR is also interesting. For small values of τ it appears as effective as Nimp1 and Behrman in terms of function calls; but it does very much worse than the other three methods in terms of iterations. This pattern persists for all values of τ as regards iterations; and it also applies to performance in terms of function calls when τ is bigger than about 2.

Overall therefore there are quite good grounds for believing that methods using a curvilinear search with the option of extrapolation often have the potential to be superior to − a conventional trust region approach.

6 Convergence of Algorithm 3 (Nimp1)

In this section we show that, under certain assumptions about f, Nimp1 will converge to a stationary point. This stationary point will − except under exceptional circumstances − be a local minimum rather than a saddle point.

Assumption 1: $f(x)$ is twice continuously differentiable for all values of x and the eigenvalues of the Hessian matrix $G(x)$ are everywhere bounded and lie in the range $[\bar{\lambda}_{min}, \bar{\lambda}_{max}]$.

In practice we only need the eigenvalue bounds to hold within an appropriate basin of convergence, say for all x such that $f(x) < f(x_0)$.

Theorem 6.1: If Assumption 1 holds the values of $\tilde{\mu}$ (and hence μ_k) remain bounded above when Algorithm 3 is implemented using (16) to calculate the search direction.

The proof depends on showing that the test condition $d_k > \alpha_2$ can be obtained for all values of μ greater than a finite threshold. Hence on any iteration *at worst* the first trial value of μ which exceeds that threshold on any iteration will yield an acceptable

point.

Dropping subscripts for convenience, (15), which is the definition of $p(\mu)$ used by Nimp1, implies

$$p^T(\mu)g = -p(\mu)^T(\mu I + G)p(\mu)$$

and hence, using Assumption 1

$$p^T(\mu)g \geq -p(\mu)^T p(\mu)[\mu + \bar{\lambda}_{max}].$$

Similarly, using the mean value theorem,

$$f(x + p(\mu)) - f(x) = p(\mu)^T g + \frac{1}{2}p(\mu)^T \bar{G} p(\mu)$$

where $\bar{G}$ is the hessian evaluated at some point between x and $x + p(\mu)$. Hence

$$f(x+p(\mu))-f(x) = -p(\mu)^T(\mu I+G)p(\mu)+\frac{1}{2}p(\mu)^T\bar{G}p(\mu) \leq -p(\mu)^T p(\mu)[\mu+\bar{\lambda}_{min}-\frac{1}{2}\bar{\lambda}_{max}].$$

Since

$$d_k = \frac{f(x + p(\mu)) - f(x)}{p^T(\mu)g}$$

it follows that

$$d_k > \frac{\mu + \bar{\lambda}_{min} - \frac{1}{2}\bar{\lambda}_{max}}{\mu + \bar{\lambda}_{max}}.$$

Thus $d_k(\mu) > \alpha_2$ if μ is sufficiently large that

$$\mu + \bar{\lambda}_{min} - \frac{1}{2}\bar{\lambda}_{max} > \alpha_2[\mu + \bar{\lambda}_{max}]$$

or in other words, if

$$\mu(1 - \alpha_2) > (\frac{1}{2} + \alpha_2)\lambda_{max} - \lambda_{min}$$

We can now conclude that there is a finite upper bound, say $\bar{\mu}$, to the values of μ encountered during the iterations of Algorithm 3.

Assumption 2: The function f is bounded below.

Theorem 6.2: If Assumptions 1 and 2 hold then the iterations of Algorithm 3 ensure that $\lim_{k\to\infty} \|g(x_k)\| = 0$, (i.e. the iterates converge to a stationary point of f) when (16) is used to calculate correction steps.

The proof is by contradiction. Suppose that $g(x_k)$ does not converge to zero as $k \to \infty$ and in particular that $\exists \hat{g}$ such that $g_k^T g_k > \hat{g} > 0$ for an infinite number of k.

Since $p_k = -(\mu_k I + G_k)^{-1} g_k$ it follows that

$$p_k^T g_k = -g_k^T (\mu_k I + G_k)^{-1} g_k < -\frac{\hat{g}}{\bar{\mu} + \bar{\lambda}_{max}}$$

where we have made use of Assumption 1 and Theorem 6.1 which states that, for all k, there is an upper bound on the values of μ_k.

The iterations of Nimp1 ensure that

$$f(x_k + p_k) - f(x_k) < \alpha_2 p_k^T g_k < -\frac{\alpha_2 \hat{g}}{\bar{\mu} + \bar{\lambda}_{max}}$$

which implies that an iteration of Nimp1 yields a decrease in objective function value which is bounded away from zero for an infinite number of $k - 1, 2, ...\infty$. But this contradicts the assumption that f is bounded below. Hence g_k must tend to zero as $k \to \infty$.

We now consider whether a stationary point to which Algorithm 3 converges must be a minimum. Unfortunately we cannot guarantee this. Consider for instance the application of Nimp1 to the function $f(x) = x_1^2 - x_2^2 + x_2^4$ starting from the point $(1, 0)^T$. There is no value of μ that will cause (15) to generate a step $p_k(\mu)$ which does not point towards the saddle point at $(0, 0)^T$. This is because the gradient at $x = (1, 0)^T$ has no component in the subspace where $f(x)$ has negative curvature. This illustrates a difficulty which could arise in a practical problem. Therefore, for the purposes of our next result, we introduce an extra assumption:

Assumption 3 The iterations of Nimp1 do not generate any points where the gradient vector g_k is orthogonal to *all* the eigenvectors of G_k which are associated with negative eigenvalues. (This of course also excludes the possibility that an iteration of Nimp1 terminates precisely at a saddle point.)

Theorem 6.3: If Assumptions 1,2 and 3 hold and if x_k is a point close to, but above, a saddle point of f then there exist values of μ in (15), the step calculation used in Algorithm 3, which can yield a new point which is below the saddle point of the quadratic model function Q.

Let n_k be the exact step from x_k to the saddle point of Q. We show that

$$Q(n_k) - Q(0) = n_k^T g_k + \frac{1}{2} n_k^T G_k n_k = -\frac{1}{2} n_k^T G_K n_k > -\frac{||g_k||^2}{2\lambda_{min}^+}$$

where λ_{min}^+ denotes the smallest strictly positive eigenvalue of G_k.

To see this, express $n_k = \sum_i a_i u_i$ as a sum of the eigenvectors u_i of G_k. Then $n_k G n_k = \sum_i \lambda_i a_i^2$, and $g_k = -\sum \lambda_i a_i u_i$ so $\|g_k\|^2 = \sum_i \lambda_i^2 a_i^2$. Since for all i we have $(\lambda_{min}^+ \lambda_i) \leq \lambda_i^2$ it follows that $\lambda_{min}^+ \; n_k G n_k < \|g_k\|^2$ as required.

The indefiniteness of G_k implies that there is a unit vector v and a positive constant β such that $v^T G_k v = -\beta$. Now suppose that $\Delta_k(\mu)$ denotes $p_k(\mu)^T p_k(\mu)$ when $p_k(\mu)$ is obtained by solving (15) and consider a step away from the point x_k which is of the form $w = \sigma\sqrt{\Delta_k(\mu)}v$ and σ is chosen as ± 1 in order that $w^T g_k \leq 0$. It follows that

$$Q(w) - Q(0) = w^T g_k + \frac{1}{2} w^T G_k w < -\frac{1}{2}\Delta_k(\mu)\beta.$$

Now the correction $p_k(\mu)$ solves (10) with $\Delta_k = \Delta_k(\mu)$ and hence the reduction in Q due to a step from x_k to $x_k + p_k(\mu)$ must be *at least* as large as that produced by the step w. Hence

$$Q(p_k(\mu)) - Q(0) < -\frac{1}{2}\Delta_k(\mu)\beta.$$

But if μ is chosen small enough that that

$$\Delta_k(\mu) > \frac{||g_k||^2}{\beta\lambda_{min}^+}$$

the step $p_k(\mu)$ must produce a bigger decrease in Q than the step n_k and hence it reaches a point *below* the saddle point of the quadratic model function. Since $||p(\mu)||$ becomes arbitrarily large as μ tends to $-\lambda_{min}^+$, it is clear that a suitable value of μ is available to be located in the extrapolation phase of Nimp1.

Note In the above Theorem we do not claim to have proved that the precise curvilinear search algorithm in Algorithm 3 will guarantee to find a suitable value of μ as identified above. As we have mentioned before, Algorithm 3 has been devised in order to test the potential benefits of a curvilinear CSDP search and is not presented as a"watertight" general purpose method.

Some of our foregoing discussion of ultimate convergence might however be said to miss the point of Nimp1 (and also of Behrman) because the main feature of these methods is to provide efficient ways of dealing with regions where f is non-convex. Both methods revert essentially to Newton's method around a local minimum where f is convex and hence ultimate convergence (and rate of convergence) properties are supported by a good deal of existing theory. Bearing that in mind, it is probably more important to consider whether the iterations of Nimp1 will provide an efficient escape from non convex to a convex region, which requires them to move well away from any local saddle point, rather than slightly below it. The counter-example preceding Theorem 6.3 shows that Assumption 3 must hold in order for such an escape to be guaranteed.

7 Conclusions

We have considered two algorithms for non-linear optimization (Nimp1 and Behrman) which can be viewed as following an approximate steepest descent path. These methods have features in common with trust region methods, but differ from most trust region methods by essentially reverting to Newton's method when the search is in a convex region. We have compared the performance of these two techniques with that of two more conventional trust-region approaches Higham and TR/*fminunc*. The results with Nimp1 and Behrman do give grounds for believing that there is some promise in the idea of using a curvilinear search continuous steepest descent path to traverse efficiently regions where the objective function is non-convex.

All the methods described in this paper use exact second derivatives; and three of them incur the unusual overhead cost of performing an eigenvalue analysis of the Hessian matrix. This makes it easy to distinguish a convex from a non-convex region; and it also has some modest advantages in cost-saving during the curvilinear search. But it would obviously be worthwhile to consider algorithms which seek to follow a similar philosophy to Nimp1 by following some sort of CSDP in regions where the objective function is non-convex while reverting to a non-second-derivative approach (e.g. quasi-Newton) when the function is convex. We intend this to be the focus of our further work in this area.

References

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

[2] M. C. Bartholomew-Biggs, S. Beddiaf, and S. J. Kane. Traversing non-convex regions. *Advanced Modeling and Optimization*, 15(2):387–407, 2013.

[3] S. Beddiaf. Continuous steepest descent path for traversing non-convex regions. *Advanced Modeling and Optimization*, 11(1):3–24, 2009.

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

[11] 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.

[12] B. Christianson. Global convergence using de-linked goldstein or wolfe linesearch conditions. *Advanced Modeling Optimization*, 11(1):25–31, 2009.

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

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

[15] 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.

[16] Desmond J. Higham. Trust region algorithms and timestep selection. *SIAM Journal on Numerical Analysis*, 37(1):194–210, 1999.

[17] MATLAB. *Matlab, Optimization Toolbox, Version 7.10.0 (R2010a), www.mathworks.com*. The MathWorks Inc. Natick, Massachusetts, 2010.

Appendix Some Results Using CUTEr test problems

The following tables present results obtained with the methods discussed in this paper. The problems are drawn from the CUTEr test set [15] and the methods compared are the trust-region approach *fminunc* – denoted by TR – along wth Nimp1 and Behrman (as implemented in Algorithm 3) and the Higham method as implemented in Algorithm 4.

The stopping criteria used by all the methods were

$$||g(x_k)||_2 < 10^{-6} \text{ and } ||x_{k+1} - x_k||_2 < 10^{-6}(1 + ||x_k||_2)$$

The values used for the controlling parameters for the iterations in Algorithms 3 and 4 are as follows

$$\alpha_1 = 0.4, \alpha_2 = 0.1, \eta_2 = 0.9, \nu_1 = 0.5, \nu_2 = 0.75.$$

It will be noted that the tables do not report on every one of the 139 test problems. We have omitted results for problems where it is clear from the counts of iterations and function calls that no interpolation or extrapolation has taken place. This allows us to observe more clearly the differences in performance in situations that are actually relevant to our investigation.

The following notation is used in the tables:

n: the number of variables

Its/Fcs: the number of iterations/function calls
(The maximum number of iterations allowed was 10000 and a table entry of the form 10000/10001 indicates this limit has been reached.)

"*" denotes the performance on each problem that is judged "best" by the criteria discussed in section 5.1

"F" indicates a method has suffered a numerical failure such as floating overflow in a function evauation, e.g. because of an exceptionally large correction step.

	Methods →		TR	Nimp1	Behrman	Higham
	Problems ↓	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
2	ALLINITU	4	10/11	7/12	7/15	9/11*
4	ARWHEAD	100	6/7	5/6	5/6	5/6
5	BARD	3	192/193	10/16	7/14	7/8*
7	BEALE	2	9/10	7/13*	6/8	17/19
8	BIGGS6	6	41/42	30/86*	64/173	88/91
12	BROWNAL	200	7/8*	15/40	11/26	16/17
13	BROWNBS	2	7075/7076	7/11*	8/11	11/12
15	BROYDN7D	10	17/18	11/13*	12/17	12/13
16	BRYBND	10	16/17	11/15	10/22*	13/15
17	CHAINWOO	4	208/209	37/63*	36/67	40/44
18	CHNROSNB	50	64/65	41/82*	40/89	50/61
20	COSINE	10	9/10	10/20	7/14*	10/11
21	CRAGGLVY	4	14/15*	F	F	F
22	CUBE	2	31/32	24/30*	26/50	31/39
23	CURLY10	100	17/18	11/31*	15/35	21/23
24	CURLY20	100	17/18	11/24*	16/38	21/22
25	CURLY30	100	17/18	11/23*	15/38	22/24
26	DECONVU	61	21/22*	37/97	F	35/36
28	DENSCHNB	2	6/7*	F	F	F
30	DENSCHND	3	257/258	28/42*	25/47	33/34
31	DENSCHNE	3	10/11	8/13*	11/21	13/14
33	DIXMAANA	15	8/9	4/10*	5/17	6/7
34	DIXMAANB	15	7/*	8/22	6/12*	7/8
35	DIXMAANC	15	8/9	8/18	7/16*	8/9
36	DIXMAAND	15	9/10	10/24	7/18*	9/10
37	DIXMAANE	15	6/7*	6/15	7/15	7/8
38	DIXMAANF	15	9/10	11/31	7/18*	14/16
39	DIXMAANG	15	10/11	8/21*	9/32	21/23
40	DIXMAANH	15	10/11*	14/32	11/28	12/13
41	DIXMAANI	15	6/7*	7/13	8/18	9/10
42	DIXMAANJ	15	12/13	9/22*	12/30	13/15
43	DIXMAANK	15	13/14	13/32	10/28*	34/42
44	DIXMAANL	15	13/14	10/21	11/26	13/14

	Methods →		TR	Nimp1	Behrman	Higham
	Problems ↓	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
46	DJTL	2	103/104*	87/731	552/2235	F
48	DQRTIC	10	15/16*	F	F	F
52	ENGVAL2	3	108/109	14/19*	16/25	17/19
53	ERRINROS	50	56/57	21/55	20/47*	25/33
54	EXPFIT	2	11/12	6/10*	8/16	9/11
55	EXTROSNB	10	330/331	360/638	355/669	71/101*
58	FLETCHBV	10	334/335*	726/1599	1388/3875	2633/634
59	FLETCHCR	10	131/132	24/31	24/35	22/27*
60	FMINSRF2	16	10/11*	16/536	12/460	F
61	FMINSURF	16	11/12*	14/625	11/369	
62	GENROSE	500	614/615	267/882*	F	338/339
63	GROWTHLS	3	10000/10001	67/140	66/144	79/87*
64	GULF	3	301/302	23/51*	22/56	37/40
65	HAIRY	2	91/92	43/96*	55/110	10000/10001
66	HATFLDD	3	114/115	19/24	16/23*	17/18
67	HATFLDE	3	15/16*	19/28	15/21	17/18
68	HEART8LS	8	131/132	81/211	72/183	71/74*
69	HELIX	3	748/749	13/30	8/14*	12/14
73	HIMMELBB	2	13/14	7/16*	14/29	18/19
74	HIMMELBF	4	201/202	49/93	8/19*	407/409
76	HIMMELBH	2	7/8*	F	F	F
77	HUMPS	2	5459/5460	4/11*	91/258	110/112
79	KOWOSB	4	21/22	6/17*	6/17*	51/57
81	LOGHAIRY	2	514/515	5/9*	6/17	70/71
82	MANCINO	100	15/16	9/20*	9/20*	13/14
83	MARATOSB	2	779/780	651/1209	6/46/1195	124/158*
84	MEXHAT	2	29/30	21/28*	21/28*	25/31
87	NONCVXU2	10	17/18	12/20	9/11*	16/18
88	NONCVXUN	10	18/19	10/16	9/13*	21/23
89	NONDIA	10	14/15*	F	F	F

	Methods →		TR	Nimp1	Behrman	Higham
	Problems ↓	n	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
91	NONMSQRT	9	88/89	348/1296	449/1647	85/110*
92	OSBORNEA	5	56/57	8/16*	40/80	56/62
93	OSBORNEB	11	30/31	13/24	13/20*	17/19
94	OSCIGRAD	10	16/17	10/52	8/10*	10000/19979
95	OSCIPATH	10	2/3*	2/44	2/44	10000/10003
107	PENALTY3	50	18/19	12/20*	14/28	16/18
108	PFIT1LS	3	10000/10001	225/406*	237/438	F
109	PFIT2LS	3	35/36*	41/74	94/184	10000/10001
110	PFIT3LS	3	33/34*	134/249	114/219	46/57
111	PFIT4LS	3	43/44*	227/423	199/398	F
114	QUARTC	25	19/20*	F	F	F
115	ROSENBR	2	27/28	19/31	19/33	11/15*
116	S308	2	10/11	8/10*	8/11	8/10*
119	SENSORS	100	16/17*	20/40	19/50	31/33
120	SINEVAL	2	55/56	41/81	40/76	20/25*
121	SINQUAD	5	11/12	7/20*	7/24	13/15
123	SNAIL	2	104/105	61/103	59/104	51/66*
124	SPARSINE	10	7/8*	7/12	7/11	9/10
126	SROSENBR	10	8/9	7/9*	7/10	7/9*
130	TOINTPSP	50	28/29	14/72*	15/115	10000/10001
134	VARDIM	200	28/29	20/23*	20/23*	28/29
135	VAREIGVL	50	15/16	11/17	7/12*	14/17
136	WATSON	12	13/14	12/28	12/24	12/13*
137	WOODS	100	58/59	37/63*	37/68	40/44
138	YFITU	3	3119/3120	37/80*	36/84	46/54

Table 1: CUTEr Test Results from TR, Nimp1, Behrman and Higham