AMO - Advanced Modeling and Optimization, Volume 11, Number 1, 2009

Continuous Steepest Descent Path for Traversing Non-convex Regions

S.Beddiaf
School of Physics Astronomy and Mathematics
University of Hertfordshire, Hatfield AL10 9AB, England
s.beddiaf@herts.ac.uk

Abstract
This paper revisits the ideas of seeking unconstrained minima by following a continuous steepest descent path (CSDP). We are especially interested in the merits of such an approach in regions where the objective function is non-convex and Newton-like methods become ineffective. The paper combines ODE-trajectory following with trust-region ideas to give an algorithm which performs curvilinear searches on each iteration. Progress along the CSDP is governed both by the decrease in function value and measures of the accuracy of a local quadratic model. Experience with a prototype implementation of the algorithm is promising and it is shown to be competitive with more conventional line search and trust region approaches. In particular, it is also shown to perform well in comparison with the, superficially similar, gradient-flow method proposed by Behrman.

1 Introduction

In this paper, we are concerned with finding a local solution of the unconstrained optimisation problem

$$\text{Minimise} \quad F(x), \quad \text{where} \quad x = (x_1, x_2, ..., x_n)^T \in R^n, \tag{1.1}$$

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, science, engineering and management. As is well known, the first order necessary condition at a local solution x_* of (1.1) is given by the system of n non-linear equations

$$\nabla F(x_*) = 0$$

and the second order condition is that the Hessian matrix $\nabla^2 F(x_*)$ is positive-definite.

There are many iterative numerical optimization techniques which can be applied to (1.1). Most of these methods use an iteration of the form

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

where p_k is a descent search direction and α_k is a step length obtained by a one-dimensional search to ensure that $F(x_{k+1}) < F(x_k)$. Sometimes these techniques enter a region where the Hessian $\nabla^2 F$ is not positive-definite and they may then exhibit slow convergence or even fail. For instance, 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. Similarly, quasi-Newton methods, whose search directions are based an approximation of $\nabla^2 F$, will be unable to use a standard updating formula to revise their Hessian estimate 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 approach which does make sense in non-convex regions is the *trust region method* [1] [2]. The strategy we discuss in this paper is related to trust region methods and is based on following the *Continuous Steepest Descent Path* (**CSDP**). This approach has already been looked at by a number of authors (e.g. [3], [4], [5],[6], [7], [8], [9], [10]) and, essentially, it uses a system of ordinary differential equations to construct a path leading to the solution of problem (1.1). Such approaches have not been as widely used as search-direction/linesearch methods, such as the Newton, quasi-Newton and conjugate gradient methods, perhaps because of the perceived difficulties inherent in accurately solving a system of *non-linear* ordinary differential equations.

The structure of this paper is as follows. In the next section we introduce the idea of Continuous Steepest Descent Path methods for unconstrained optimization. We look at ways of approximating the CSDP in order to solve problems of the form (1.1) and give an outline of some possible algorithms. In section 3, the performance of these algorithms is illustrated and compared on a small example. In section 4, we look more closely at some of the algorithmic choices involved in the CSDP method and give some numerical results in which their performance is compared with that of some other well known methods from the MATLAB optimization toolbox [11]. Conclusions and a discussion of further work are given in section 5.

2 The Continuous Steepest Gradient Path

Consider the unconstrained optimization problem (1.1). We suppose it involves a nonlinear twice continuously differentiable objective function $F(x)$. At each point the gradient vector is $\nabla F(x)$ and the Hessian is $\nabla^2 F(x)$ (which will sometimes be denoted by $G(x)$).

Three well-known techniques based on a line search are
Steepest Descent,

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

Newton,

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

and *Quasi-Newton*

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

where H denotes a positive-definite approximation of $G^{-1}(x)$ which is updated at the end of each iteration. From a given starting 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}) designed to converge to the true solution x_*.

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

$$\frac{dx}{dt} = -\nabla F(x(t)), \;\; x(0) = x_0. \tag{2.4}$$

If the solution $x(t)$ of (2.4) for $t > 0$ has a limit point such that $\lim_{t\to\infty} x(t) = x_*$, then x_* is a stationary point of $F(x)$ ([3], [4],[6], [7]). 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 $\nabla^2 F(x_*)$ is positive-definite.

A CSDP method can be outlined as follows. From $x(0) = x_0$, let $p(t)$ be a curve, with $p(0) = 0$, which is an *approximation* to the integral curve $x(t)$ which solves (2.4). The method then searches along $p(t)$ 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.) 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 the initial condition changed to $x(0) = x_1$. A search along $p(t)$ will then yield a new point x_2; and this process can be repeated until a point is found that satisfies a convergence test such as $||\nabla F(x_*)|| < \varepsilon$, where ε is some specified tolerance.

A *gradient flow method* of the kind just described has been proposed by Behrman [3]. The k-th iteration of Behrman's algorithm uses an approximation of the vector field $-\nabla F$ about $x = x_k$ involving the integral curves of the linearised CSDP

$$\frac{dx}{dt} = -\nabla F(x_k) - \nabla^2 F(x_k)(x - x_k) \tag{2.5}$$

where x_k is the starting point of the k^{th} iteration. Equation (2.5) has an analytical solution through x_k given by

$$x(t) = x_k + p_k(t)$$

where

$$p_k(t) = -R\Lambda R^T g_k \tag{2.6}$$

and $R = R(x_k)$ is the matrix whose columns are the normalised eigenvectors of $\nabla^2 F(x_k)$ while Λ is a diagonal matrix whose elements are derived from the eigenvalues $d_1, \ldots, d_n$ via

$$\Lambda_{ii} = \begin{cases} \frac{1}{d_{ii}}\left(e^{-d_{ii}t} - 1\right) & \text{for} \quad d_{ii} \neq 0 \\ t & \text{for} \quad d_{ii} = 0 \end{cases} \tag{2.7}$$

From any given point, Behrman's algorithm calculates a curve that is initially tangent 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.

Theorem [3] Let $x(t)$ be the solution to (2.5). For a fixed $t_0 \geq 0$ if $\nabla F(x(t)) \neq 0$ for all $t > t_0$, then $F(x(t))$ is strictly decreasing with respect to t, for all $t > t_0$.
Proof: We know

$$\frac{dF(x(t)}{dt} = \nabla F(x(t))^T \frac{dx(t)}{dt} = -\nabla F(x(t))^T \nabla F(x(t)) = -||\nabla F(x(t))||_2^2.$$

Since $\nabla F(x(t)) \neq 0$ when $t > t_0$, it follows that

$$\frac{dF(x(t)}{dt} < 0,$$

i.e. $F(x(t)$ is strictly decreasing for $t > t_0$.$\Diamond$

It can be seen from (2.5) that, if the only information given about F at x_k is its gradient $\nabla F(x_k)$, we can use $p_k(t) = -t\nabla F(x_k)$ as the solution to (2.5). 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 a positive-definite quadratic the algorithm finds the minimiser in one step. This is identical to the Newton step.

If $F(x)$ is a general function for which $\nabla F(x_k)$ and $\nabla^2 F(x_k)$ are known then we can use (2.6) to compute a path $p_k(t)$ corresponding to a 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 $p(t)$ we can connect the initial point x_0 with a critical point x_* of F as shown in Figure 1. (The dotted curves in Figure 1 represent the CSDP that would be obtained by solving (2.4) exactly.)

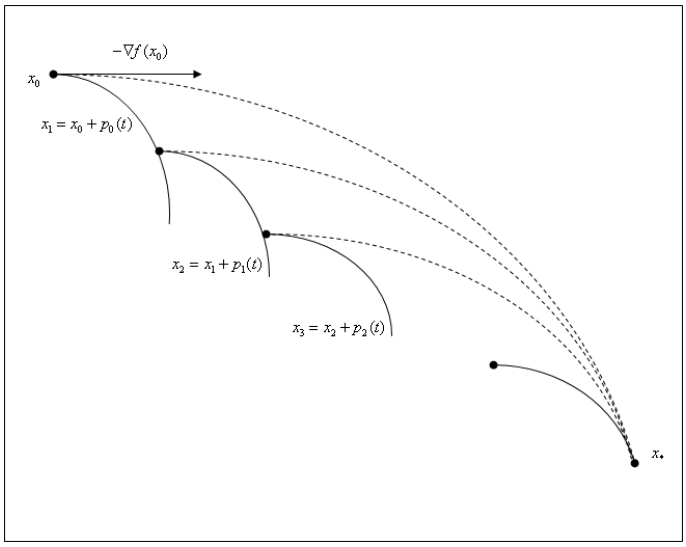

Figure 1: An approximate CSDP

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.

2.1 Approximating CSDP

In this section we consider practical algorithms for solving (1.1), which approximate the CSDP by finding a numerical solution to (2.4).

Starting from the point x_k where the parameter $t = 0$ and applying *Euler's method* to (2.4), a new estimate of the point on the gradient trajectory corresponding to $t = \delta t$ can be given by

$$x_{k+1} = x_k - \delta t g_k \tag{2.8}$$

where $g_k = \nabla F(x_k)$. On the other hand, by using the *Implicit Euler method* we get

$$x_{k+1} = x_k - \delta t g_{k+1}. \tag{2.9}$$

If G_k, denotes $\nabla^2 F(x)$, then (2.9) can be approximated by

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

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

$$(I + \delta t G_k) p_k = -\delta t g_k. \tag{2.10}$$

Even when G_k is non-positive definite, (2.10) gives a step p_k which decreases F, so long as δt is sufficiently small.

We can also consider calculating x_{k+1} by a second-order method which combines (2.9) and (2.10) in a mixed explicit/implicit Euler step so that

$$x_{k+1} = x_k - \frac{\delta t}{2}[\ g_k + (I + \delta t G_k)^{-1} g_k\ ]. \tag{2.11}$$

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 (2.9) and x^I_{k+1} comes from (2.10).

Equation (2.10) gives a step similar to that of the *trust region methods* which use

$$(\mu I + G_k) p_k = -g_k \tag{2.12}$$

where μ in (2.12) is effectively the reciprocal of the step length δt in (2.10). Equation (2.10) gives the Newton step as $\delta t \to \infty$ while (2.12) gives the Newton step when $\mu = 0$. Also equation (2.10) makes p_k parallel to $-g_k$ when $\delta t = 0$ while (2.12) makes p_k tend to a steepest descent step as $\mu \to \infty$. For a major survey of trust region methods see [2].

When G_k is non-positive definite we can trace out a path away from x_0 by using a sequence of μ values in (2.12). These must be chosen in a range $\infty > \mu > \mu_{min} > 0$, where $\mu_{min} = -d_{min}$, the most negative eigenvalue of G_k.

When solving (2.12) for several different values of μ, we can either use a fresh Cholesky factorisation of the coefficient matrix for each μ or determine the eigensystem of G_k via the orthogonal factorisation $G = RDR^T$. In the second case, since $RR^T = I$, the system (2.12) can be written

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

and to solve for each value of μ we may use

$$\hat{g}_k = R^T g_k; \quad \hat{p}_{k,i} = \frac{\hat{g}_{k,i}}{\mu + d_{ii}}, \quad i = 1, \ldots, n; \quad p_k = -R\hat{p}_k. \tag{2.13}$$

This calculation can be regarded as being comparable with those used in the Behrman correction [3] given by (2.6), (2.7).

For a single solution of (2.12), the eigenvalue calculation is more expensive than a Cholesky factorisation. But if many values of μ are tried then subsequent solutions via (2.13) may be cheaper than re-factorisation. Therefore the practical merit of using RDR^T factors in the CSDP method depends on how far and how accurately we want to pursue a curved path solution of (2.4).

We consider first the case when G_k is non positive definite (The Newton step with $\mu = 0$ in (2.12) 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 $\mu_{min} = |d_{pp}|$, and d_{pp} is the most negative element in D.

To trace out an approximate CSDP from x_k we must 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 use trials of decreasing μ values towards μ_{min} so long as (2.12) yields a an improved point $x_k + p_k$ which is *close enough* to the CSDP. Our intention is to make a significant progress along this path to reach an acceptable new point $x_k + p_k$ where the Hessian will be recomputed.

2.2 Searching along the curved path

Our aim is to determine μ in (2.12) to ensure that p is 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 (see [1] for instance). We want μ to give $F(x_k + p_k) < F(x_k)$ and also to ensure 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 prediction. If we choose μ and then compute the corresponding p we can evaluate $F^+ = F(x+p)$, $g^+ = g(x+p)$ and consider the following test ratios.

$$D_1 = \frac{F^+ - F}{p^T g}, \quad D_2 = \frac{|F^+ - (F + p^T g + \frac{1}{2} p^T G p)|}{|p^T g + \frac{1}{2} p^T G p|}$$

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

D_1 compares the actual change in F with a first order prediction. If $D_1 \approx 1$ this suggests that the step is too short. On the other hand $D_1 < 0$ indicates the search has gone past the one-dimensional minimum. If F is quadratic then $D_1 = 0.5$ at the one dimensional minimum along p.

D_2 compares the actual change in F with the quadratic predicted reduction and if the difference is relatively small then it seems reasonable to continue to extrapolate along the CSDP (so long as $D_1 > 0$).

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 $D_1 > 0$).

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

2.3 Algorithm for searching along CSDP

We can now formalise the steps of an iteration which uses the ideas discussed above. We consider first the case when G_k is not positive definite.

Outline CSDP Algorithm for nonconvex regions

Given the parameters $\alpha > 1$, $\beta < 1$, $\gamma < 1$, D_1^{min}, D_1^{max}, D_2^{max} and D_3^{max}.
1) Set $\mu = \alpha \mu_{min}$
2) Compute p from (2.12) and hence get $x+p$, F^+, g^+, D_1 ,D_2 ,D_3.
3) If $D_1 < D_1^{min}$ set $\mu = \mu + \gamma(\mu - \mu_{min})$ (to interpolate) and go to (2)
4) 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) and go to (2)
5) Otherwise $x+p$ is acceptable and the iteration is compete.

In the case when G_k is positive definite we could revert to the standard well-known linesearch version of the Newton method. However, it is still possible to use a curvilinear search in terms of μ as previously described. The strategy is to choose an initial value of $\mu = 0$ and then compute p, $x + p$ and $F = F(x + p)$. In the positive definite case, however, we simplify the search and only use the test ratio D_1. Thus,if D_1 is too close to 1, it may be reasonable to extrapolate by decreasing μ *below zero*, and this is done by replacing μ by

$$\mu \leftarrow \mu - \beta(\mu - \mu_{min}). \tag{2.14}$$

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

$$\mu \leftarrow \mu + \gamma(\mu - \mu_{min}). \tag{2.15}$$

The curvilinear search algorithm sketched above can be combined with several different ways of calculating p_k. If we use (2.13) then the resulting algorithm will be referred to as NIMP1. The algorithm using Behrman's calculation of p_k from (2.6), (2.7) will be called UMINH as in [3] (although we emphasise that the curvilinear search in our implementation is not the same as in Behrman's). Finally, if p_k is obtained from (2.11) we call the algorithm NIMP2. In practice, the step calculation in NIMP2 is done by first obtaining p_k from (2.12) then setting $\tilde{p} = p_k$ and finally defining a new p_k as

$$p_k = \frac{1}{2}(-g_k + \tilde{p}).$$

3 Numerical results for test problem T1

As a simple test example 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 $x_0 = (2.05, 1.6)^T$. We look at CSDP solutions using the following values for parameters in the algorithm of section 2.3:

$$\alpha = 2, \quad \beta = 0.5, \quad \gamma = 0.25, \quad D_1^{min} = 0.1, \quad D_1^{max} = 0.6,$$

$$D_2^{max} = 0.1 \quad \text{and} \quad D_3^{max} = 0.5$$

The results in Table 1 were obtained using MATLAB implementations of NIMP1, NIMP2 and UMINH along with the trust region method implemented as `fminunc` in the MATLAB optimization toolbox [11]. We denote this by TR.

The method in `fminunc` is described in [12], [13] and it is comparable with NIMP1 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 NIMP1 however in not obtaining an exact solution to the trust-region subproblem but rather by 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 (2.12) 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 n or s, by applying a preconditioned conjugate gradient (PCG) method (see [14]) 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 approximation to the Newton direction.

Method	No of Its	No of fcn calls
NIMP1	7	12
NIMP2	6	20
UMINH	13	49
TR	8	9

Table 1: Results for the function $x_1 x_2 + (x_1^2 + 2x_2^2 - 10)^2/100$

We can observe that NIMP1 and NIMP2 need fewer iterations than TR and – perhaps more significantly – they appear to be considerably more efficient than UMINH. We also note of course that the CSDP methods use more function evaluations than the trust-region approach. This is plainly due to 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 (1) of the outline algorithm or a larger value of β in (2.14) 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 also reduce the associated cost of computing second derivatives*.

Clearly the results in Table 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 section.

The CSDP convergence paths for NIMP1, NIMP2 and UMINH as shown in Figures

2 - 4. The circled points mark the starts and ends of iterations and the dots indicate points obtained with different values of μ in (2.12) or (2.6),(2.7). It is clear from the figures that the solution by all the three CSDP methods follows a different curvilinear path through the non-convex region.

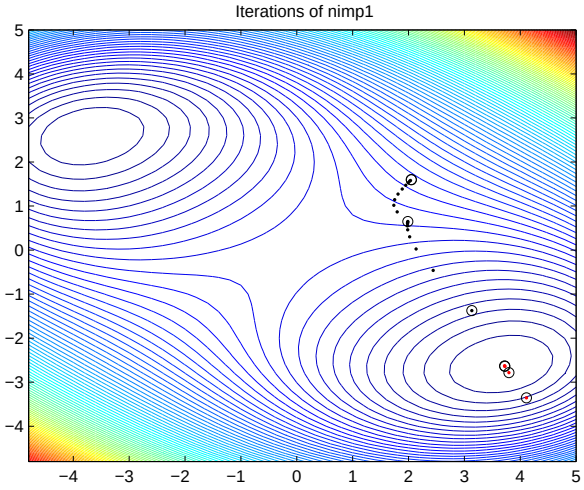

Figure 2: The solution path for NIMP1 on problem T1.

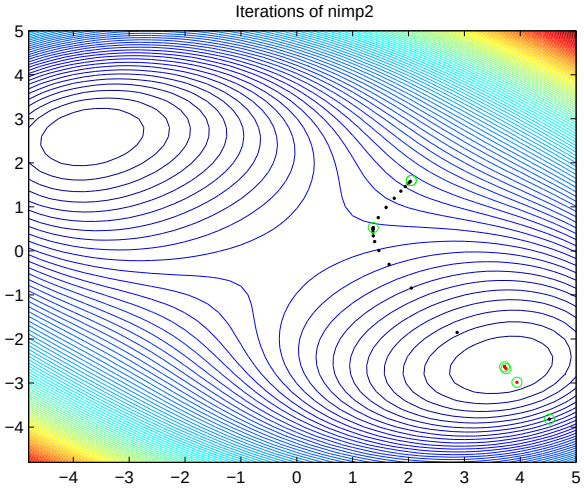

Figure 3: The solution path for NIMP2 on problem T1.

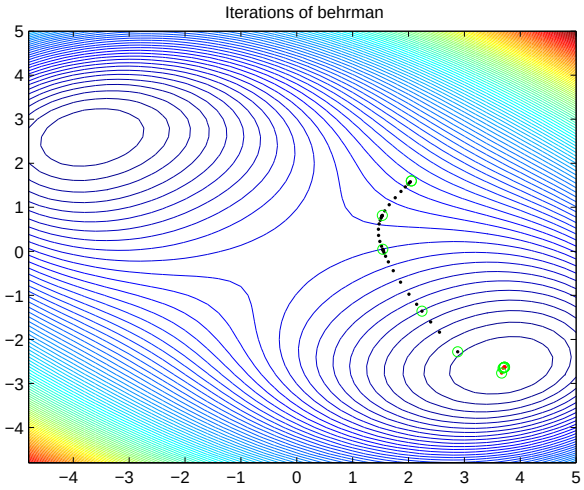

Figure 4: The solution path for UMINH on problem T1.

4 Further algorithmic investigation

Results for problem T1 suggest that the CSDP techniques used here are worth further investigation. The algorithm stated in section 2.3 involves several parameters and in this section we shall consider how performance can be affected by different choices of two of them.

4.1 Varying the parameter D_3^{max}

We can consider varying the threshold on the accuracy parameter D_3 which controls how far the search is pursued along the approximate CSDP. Figures 5 – 7 relate to problem $T1$ and show how the NIMP1 path varies as D_3^{max} changes. Figure 5 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 $|1 - D_3| < 0.01$ (Figures 6 and 7) 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 the Figure 5 represents the solution using the fewest function evaluations.

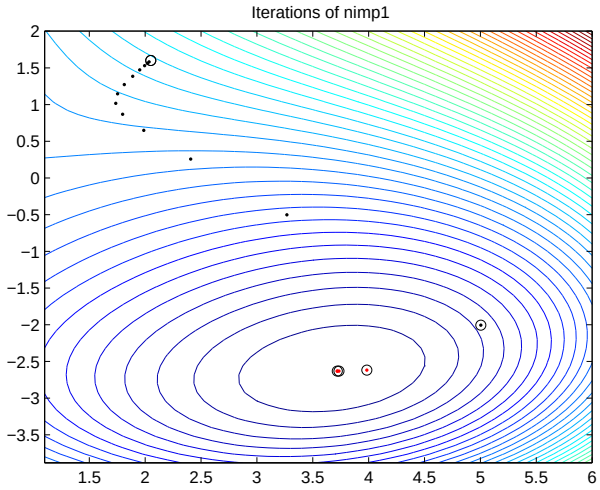

Figure 5: The solution path for NIMP1 on problem $T1$ using $D_3^{max} = 0.5$.

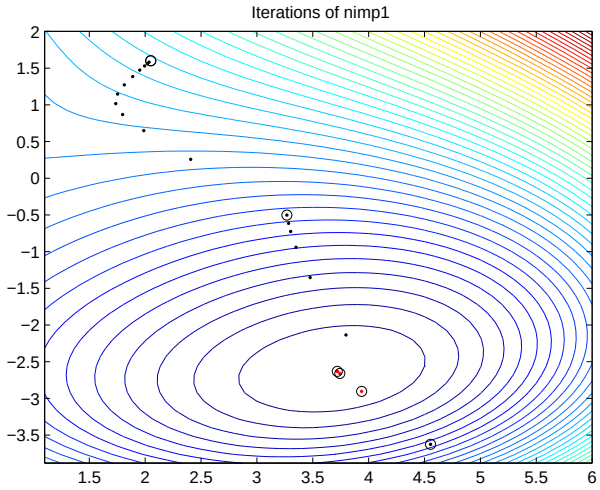

Figure 6: The solution path for NIMP1 on problem $T1$ using $D_3^{max} = 0.05$

4.2 Choosing an initial μ for each iteration

We look now at α which is involved in determining an initial value of μ on each iteration. As in the 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). Because of the orthogonality of

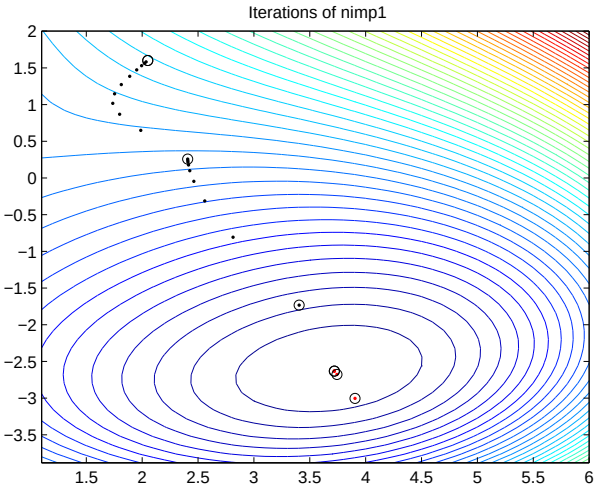

Figure 7: The solution path for NIMP1 on problem $T1$ using $D_3^{max} = 0.01$

the matrix R used in the calculation scheme (2.13) for NIMP1, we deduce that

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

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

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

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

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

for some $\alpha > 1$. Some results with this safeguarded formulae are given in Table 2. The first rows of the table show what happens when the CSDP methods are applied to problem T1 with fixed values of α while the last row uses the formula (4.1) with $\alpha = 2$. The other parameter values are

$$\delta_0 = 1, \quad \beta = 0.5, \quad \gamma = 0.25, \quad D_1^{min} = 0.1, \quad D_1^{max} = 0.6$$

$$D_2^{max} = 0.1 \quad \text{and} \quad D_3^{max} = 0.5$$

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 clear that we

$\alpha =$	NIMP1	NIMP2	UMINH
	Its/Fcs	Its/Fcs	Its/Fcs
1.5	7/10	5/13	15/53
2	7/12	7/19	13/49
3	7/14	7/21	16/55
4	7/16	5/18	15/50
5	7/16	7/23	16/55
10	7/18	6/23	18/54
15	5/18	6/24	17/67
20	6/20	6/25	18/58
30	6/21	6/26	17/72
50	7/24	5/26	18/62
100	7/25	5/28	17/64
200	7/27	5/30	17/68
1000	6/31	7/39	18/78
(4.1) $\alpha = 2$	6/13	6/18	18/47

Table 2: Results using different values of α in CSDP applied to problem T1

cannot use $\alpha = 1$ since this would make (2.12) a singular system. However 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 curved path at each iteration, giving a corresponding increase in function calls. The automatic choice (4.1) seems to yield a good compromise.

5 Further numerical results

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 method and hence they involve functions with large non-convex regions – and sometimes saddle-points – which are quite close to local minima. The problems are:

T1: $F = x_1x_2 + 0.01(x_1^2 + 2x_2^2 - 10)^2$. Starting point $x_0 = (2.05, 1.6)^T$

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

T1r2: $F = -(1+\phi(x_1,x_2))^{-2}$ where $\phi = x_1x_2 + (x_1^2+2x_2^2-10)^2/100$.
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: $F(x_1,x_2) = -(1+\phi(x_1,x_2))^{-1}$
where $\phi(x_1,x_2) = x_1x_2 + 0.01 \max\{0,(x_1^2+2x_2^2-10)\}^2$.
Starting point $x_0 = (0.26, 0.16)^T$

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

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.
Starting point $x_0 = (3, 3, ..., 3)^T$

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

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 $x_0 = (-1, 0.1)^T$

The functions of the form $F(x) = -1/1+\phi(x))^{-1}$ are suggested by the shape of the famous Runge function used to demonstrate 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 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 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 the minimum.

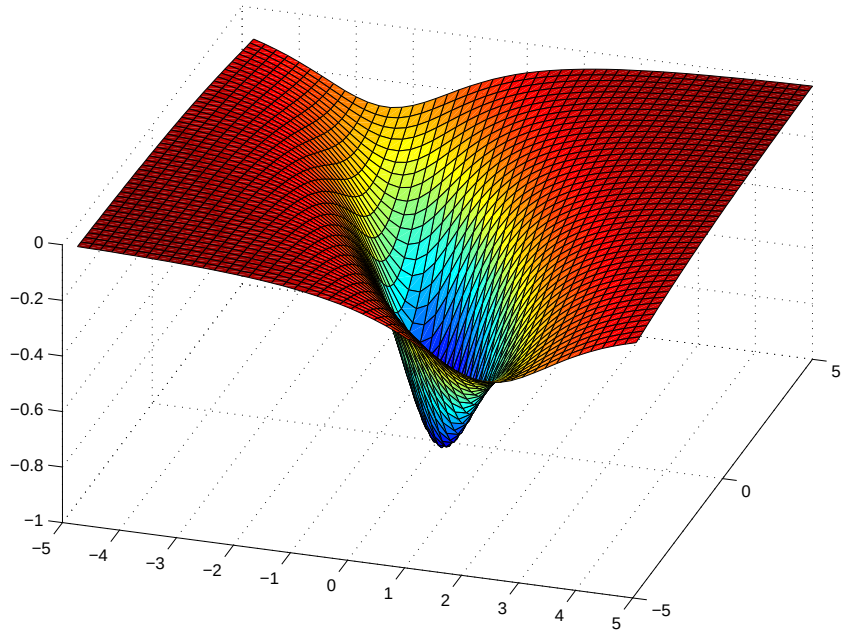

Figure 8: Surface plot for Problem T4

In Tables 3 and 4 we summarise some results for the 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 4.1, that the results were very little changed when D_3^{max} was set to 0.25.) The second column of Tables 3 and 4 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 best performance measured primarily in terms of 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 overhead algorithmic costs in computing search directions etc.)

The results in Tables 3 show that NIMP1 consistently does better than NIMP2 and UMINH. In particular, it seems that UMINH is rarely competitive. NIMP1 also

usually outperforms TR in terms of iteration count – appreciably so on problems T1b and T3. The choice of α does not greatly affect the numbers of iterations needed by the CSDP methods. All these remarks are fairly consistent with what was observed on problem T1.

The results in Table 4 show some features different from those in Table 3. For instance, on the various instances of problem T4, the CSDP methods are all much more sensitive to the choice of α. Interestingly, UMINH appears to do better on these problems than on the others in the test set, sometimes needing fewer iterations than either NIMP1 or NIMP2. Even then, however, the number of function evaluations is usually higher. We may also note that NIMP2 and UMINH behave in a rather similar way on the **T4** problems while on all the other examples the performance of NIMP2 is more like that of NIMP1. NIMP1 is also less competitive with TR on the T4 and T4r problems.

It is significant to note that the pilot version of NIMP1 often appears quite competitive with the trust-region routine TR. In the next section we consider some refinements to the CSDP algorithms which can be expected to improve their performance.

6 Discussion and Conclusions

We have been considering two methods (NIMP1 and NIMP2) derived from the implicit Euler method to estimate the CSDP through a non-convex region. Many CSDP algorithms have already been proposed (see [3] – [10]) but we believe our work differs in the way we use μ as a curvilinear search parameter and in the use of a 2nd order estimate of the CSDP step in NIMP2.

In calculating correction steps in both NIMP1 and NIMP2 we have to solve

$$(\mu I + G)p = -g \tag{6.1}$$

for a range of values for μ. We have chosen to do these repeated solutions via a once-and-for-all calculation of the eigenvalues of G by an RDR^T decomposition. This approach is similar to that employed by another CSDP method called UMINH which is due to Behrman [3] and is based on the exact solution of (2.4), also making use of the eigenvalues and eigenvectors of G. To justify the cost of such an expensive matrix decomposition on each iteration we want to make good progress along the resulting curved path $p(\mu)$. Hence the main purpose of this paper has been to explore ways of choosing steps along $p(\mu)$ which keep sufficiently close to CSDP while giving an acceptable decrease in the objective function.

Fctns	Methods	NIMP1	NIMP2	UMINH	TR
	$\alpha =$	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
T1	2	7/12	7/19	13/49	
	100	7/25	5/28	17/64	*8/9*
	a	**6/13**	6/18	18/47	
T1r	2	*8/14*	8/35	15/42	
	100	8/33	7/44	9/10	11/28
	a	**7/15**	8/22	16/45	
T1r2	2	9/13	**7/19**	16/43	
	100	7/37	8/48	14/72	*9/10*
	a	9/18	8/23	17/46	
T1a	2	*5/10*	5/15	15/38	
	100	5/18	5/23	12/51	9/10
	a	5/11	**4/14**	17/45	
T1b	2	**5/12**	5/17	19/58	
	100	**5/12**	5/25	17/82	*9/10*
	a	**5/12**	5/17	19/58	
T1ar	2	**6/14**	7/25	17/68	
	100	7/33	8/47	17/87	*9/10*
	a	**6/14**	7/24	16/50	
T2	2	9/11	8/18	18/46	
	100	8/23	9/40	19/63	*9/10*
	a	**8/14**	9/21	19/41	
T2r	2	8/13	8/29	15/45	
	100	6/23	7/33	14/57	*9/10*
	a	**6/10**	8/24	15/38	
T3	2	7/20	7/36	23/78	
	100	7/33	**6/39**	21/106	*14/15*
	a	7/20	7/28	21/66	

Table 3: Results from NIMP1,NIMP2,UMINH and TR.

There is scope further work on the details of algorithms which approximate CSDP; but preliminary results with our prototype implementations are rather encouraging. NIMP1 – and to a lesser extent NIMP2 – appear to outperform UMINH comfortably. Moreover they also seem to do better (on some problems) than a trust region approach. This applies particularly to the numbers of iterations used rather than the numbers of functions evaluations. This last remark underlines the need for further work on the curvilinear search.

Fctns	Methods	NIMP1	NIMP2	UMINH	TR
	$\alpha =$	Its/Fcs	Its/Fcs	Its/Fcs	Its/Fcs
T4_2	2	**5/6**	7/29	12/47	
	100	41/70	16/82	19/83	8/9
	a	7/8	9/21	15/39	
T4_4	2	15/38	**9/60**	12/90	
	100	53/90	18/99	19/107	*22/23*
	a	23/25	12/35	14/46	
T4_10	2	33/51	**10/56**	13/60	
	100	44/94	22/105	24/121	*12/13*
	a	33/34	15/30	18/36	
T4_20	2	34/54	**13/81**	13/83	
	100	54/116	25/125	28/143	*12/13*
	a	14/16	20/52	22/70	
T4_50	2	34/65	**13/87**	16/101	
	100	39/114	30/145	32/147	*15/16*
	a	21/23	26/65	28/79	
T4_100	2	45/83	15/101	**9/42**	
	100	47/132	32/155	36/176	*17/18*
	a	16/19	29/83	32/105	
T4r_20	2	34/40	**11/66**	11/88	
	100	55/120	26/136	26/135	*12/13*
	a	14/16	15/31	19/40	
T5	2	**8/12**	9/23	19/45	
	100	8/32	9/48	20/73	*9/10*
	a	8/14	9/28	19/46	
T5a	2	12/16	10/33	22/56	
	100	16/83	11/58	22/85	18/19
	a	**9/16**	9/24	22/61	

Table 4: Further results from NIMP1,NIMP2,UMINH and TR.

Perhaps the most important issue for the development of NIMP1 and NIMP2 is the choice and adjustment of the parameter μ in (6.1) which controls progress along the approximate CSDP. The automatic method (4.1) for choosing the initial μ for each iteration is quite closely related to the step calculation in NIMP1. This may partly explain why NIMP1 has proved to be the best of the CSDP methods in the numerical tests we have reported; and it may be possible to devise alternatives to (4.1) which are more appropriate for NIMP2 and UMINH and which can bring

about improvements in their performance.

As regards the adjustment of μ, the versions of NIMP1, NIMP2 and UMINH described in this paper have all used rather simple expansion/contraction rules (2.15), 2.14). It is, however, easy to imagine a more flexible strategy which would, for instance, allow μ to decrease more when $D_1 \approx 1$ than when $D_1 \approx D_1^{max}$.

Of possibly lesser importance, but still worth further investigation, are choices of thresholds $D_1^{min}, D_1^{max}, D_2^{max}, D_3^{max}$. We have given some consideration to the choice of D_3^{max} and have noted that the performance shown in Tables 3 and 4 does not seem to be much affected when D_3^{max} is decreased from 0.5 to 0.75. We have also shown in section 4.1, however, that setting the more demanding requirements with $D_3^{max} < 0.1$ may result in premature termination of the the curvilinear search. In other words the choice of D_3^{max} is of some significance but, within a reasonable range, it does not appear to be critical We would expect similar remarks to be true for the other parameters.

One further research question for the implementation of NIMP1 and NIMP2 relates to the repeated solution of the system (6.1) for different values for μ. Instead of using the RDR^T factors of G in the calculation scheme (2.13) we could simply perform a fresh LL^T factorization for each value of μ. The eigenvalue decomposition is expensive and may well require more computing effort than several Cholesky solutions. Such a change in the method of calculating $p(\mu)$ will, of course, not change the counts of iterations and function evaluations shown in the comparison tables: but it may well have an appreciable effect on the run-times for solving larger problems.

As a final remark, it is worth pointing out that it one could explore quasi-Newton variants of NIMP1 and NIMP2 in which the exact Hessian is replaced by an updated approximation which is not forced to be positive definite.

References

[1] Fletcher, R., Practical Methods of Optimization, Second edition, (A Wiley-Interscience Publication), John Wiley and Sons, 1987.

[2] Conn,A.,R., Gould, N.I.M., Toint, P.T., Trust Region Methods MPS-SIAM Series on Optimization, Philadelphia, 2000.

[3] Behrman, W., An Efficient Gradient Flow Method for Unconstrained Optimization, PhD Thesis, Stanford University, 1998.

[4] Brown, A.A., Optimisation Methods Involving the Solution of Ordinary Differential Equations, PhD Thesis, Hatfield Polytechnic, 1986.

[5] Brown, A.A., Bartholomew-Biggs, M.C. 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.

[6] Botsaris, C.A., Jacobson, D.H. A Newton-type Curvilinear Search Method for Optimization, J. Maths Anal.Appl., 54(1):217-229,1976.

[7] Vial, J.P., Zang, Israel, Unconstrained Optimization by Approximation of the Gradient Path. Maths. Oper. Res., 2(3):253-265, 1977.

[8] Botsaris, C.A., Differential Gradient Methods. J. Maths. Anal. Appl., 63(1):177-198, 1978

[9] Botsaris, C.A., 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] Andrei, N., Gradient Flow Algorithm for Unconstrained Optimization ICI Technical Report, April 2004.

[11] The Mathworks Co, MATLAB documentation, www.mathworks.com

[12] Branch, M.A., T.F. Coleman, Y. Li, A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems, SIAM Journal on Scientific Computing, Vol. 21, Number 1, pp. 1-23, 1999.

[13] Byrd, R.H., R.B. Schnabel, and G.A. Shultz, Approximate Solution of the Trust Region Problem by Minimization over Two-Dimensional Subspaces, Mathematical Programming, Vol. 40, pp. 247-263, 1988.

[14] Broyden C.G. and M.T. Vespucci, Krylov Solvers for Linear Algebraic Systems, Studies in Computational Mathematics, 11, Elsevier, 2004.