Further developments of methods for traversing regions of non-convexity in optimization problems

Michael Bartholomew-Biggs, Salah Beddiaf and Bruce Christianson
University of Hertfordshire, Hatfield, UK

March 2021

Abstract

This paper continues to address one of its author's obsession with the well-known problem of dealing with non-convexity during the minimization of a nonlinear function $f(x)$ by Newton-like methods. It builds on some proposals made in [11] which involve the use of a search along a curve which approximates the continuous steepest descent path. In this note we consider ways of carrying out this search more economically than in the algorithm proposed in [11] and presents some encouraging numerical evidence.

1 Introduction

This paper develops proposals made in [11] which investigates strategies for overcoming the difficulties arising in Newton-like minimization when the Hessian of the objective function is not positive definite.

If we are minimizing a continuous function of n variables, $f(x)$, whose gradient and Hessian are denoted by $g(x)$ and $G(x)$ respectively, then the local quadratic model

$$f_q(x+p) = f + p^t g + \frac{1}{2} p^t G p \qquad (1)$$

suggests that $p = -G^{-1}g$ will be a good estimate of the step from any point x to the minimum of f **provided G is positive definite**. If however G is not positive definite then the quadratic model yields no such useful correction since $p = -G^{-1}g$ will lead to a local maximum or a saddle point of f_q. To make progress therefore we have to find another way of finding a downhill direction towards a region where $f(x)$ is convex and where a minimum will be found (if our problem is a well-posed one).

The most rapid decrease in $f(x)$ can be obtained by following the *continuous steepest descent path (CSDP)* which is defined as the solution of the ODE system

$$\frac{dx}{dt} = -g(x). \tag{2}$$

It is well known (see for example [2],[4],[12]) that the CSDP can be approximated by the parametric curve $x + p(\mu)$ where $p(\mu)$ is the solution to

$$(\mu I + G)p = -g \tag{3}$$

This way of calculating $p(\mu)$ follows from an application of the implicit Euler method to (2). It can however also be derived as the basic correction-step calculation in trust-region methods of optimization which are based on minimizing quadratic model function (1) subject to a step-size constraint $|p^t p = \Delta^2$ where Δ is called the trust-region radius. Such methods are extensively described in [8]. We can interpret μ as being the Lagrange multiplier associated with the step-size constraint on but this relationship does not lead to any straightforward way of determining a value of μ which corresponds to some desired trust-region radius. Qualitatively, however, we can say that an increase in μ corresponds to a decrease in Δ and vice-versa.

In [11] a curvilinear search algorithm is proposed that computes corrections by adjusting the value of μ in (3) until a new point $x + p(\mu)$ is found which yields a sufficient decrease in the value of the objective function f. The definition of a sufficient decrease is often based on comparing the actual change in $f(x)$ with the change predicted by the quadratic model f_q. Thus, if f^+ denotes $f(x+p(\mu))$ then we can consider

$$D_q = \frac{f^+ - f}{f_q(x+p) - f} \tag{4}$$

Using (1) and (3) we can also write D_q as

$$D_q = 2\frac{f^+ - f}{p^t g - \mu p^t p} \tag{5}$$

If a step $p(\mu)$ yields a new function value such that $D_q \approx 1$ this indicates good agreement with the local quadratic model and suggests that further progress could be made by taking a larger step. This can be achieved by reducing the value of μ and solving (3) again.

Furthermore, if $D_q > 1$ we deduce that the objective function has decreased even more than was predicted by the quadratic model; and this too seems good grounds for reducing μ and taking a larger step. This last remark may however be only true up to a point because if $D_q >> 1$ it also suggests that the step p is taking the search well beyond the validity of the quadratic model and we probably need to start another iteration with a new Hessian matrix G.

Finally,if the step $p(\mu)$ leads to $D_q \leq 0$ it means that $f^+ \geq f$ and again we have gone well beyond the range where the local quadratic model is applicable. A shorter correction step is therefore required and this can be obtained by increasing μ and using (3) again. Successive adjustments of μ would continue in accordance with these principles until a correction step is found for which D_q lies in some acceptable range – for example $0.9 \geq D_q \geq 0.1$.

Another commonly used way of assessing the decrease in f involves the ratior

$$D_l = \frac{f^+ - f}{p^t g} \tag{6}$$

which compares the actual and the linearly predicted change produced in $f(x)$ by the correction step p. A value $D_l < 0$ indicates that the step p is "too long" and has produced an increase in function value. If however $D_l \approx 1$ then f behaves like its first order approximation which *may* indicate a step that is "too short". Thus a successful step can be characterised by limits such as $0.9 \geq D_l \geq 0.1$.

In choosing and adjusting values of μ for use in (3) it is helpful to have some knowledge of the eigenvalues of G. Let us assume that these are numbered in descending order so that λ_1 is the largest ("most positive") eigenvalue and λ_n is the smallest ("most negative") one. Then we certainly require $\mu > -\lambda_n$ in order to ensure that the coefficient matrix in (3) is positive definite and hence that $p(\mu)$ is a descent direction. Furthermore, in the algorithms described more fully in the next section, a knowledge of λ_1 is also useful in the process of adjusting μ.

Following the ideas in [11] we therefore propose a Newton-like algorithm in which a typical iteration from a starting point x can be outlined as follows

Outline Algorithm 1
Newton-like iteration using curvilinear searches

```
Determine the extreme eigenvalues of $G$, $\lambda_1$ and $\lambda_n$
If $G$ is not positive definite
    choose $\mu > -\lambda_n$ to use in (3)
    if necessary adjust $\mu$ iteratively until (3) gives a point $x + p(\mu)$ so that

        $1 > \eta_1 \geq D_q \geq \eta_2 > 0$

else
    choose $\mu = 0$ to use in (3)
    if necessary adjust $\mu$ iteratively until (3) gives a point $x + p(\mu)$ so that

        $D_q \geq \eta_2 > 0$

Start a new iteration with $x$ replaced by $x + p(\mu)$
```

It will be noted that when G is not positive definite the curvilinear search can perform both interpolation and extrapolation to ensure $1 > \eta_1 \geq D_q \geq \eta_2 > 0$. However only interpolation is permitted when G is positive definite because extrapolation in this case would entail the use of negative values of μ in (3) and there is no theoretical justification for such a choice within the quadratic model.

The above outline algorithm includes the approach referred to as NIMP1 in [11]. This gets its eigenvalue information by decomposing G into the product RDR^t where R is the orthonormal matrix of eigenvectors and the eigenvalues are the elements of the diagonal matrix D. This is an expensive factorization but it also enables the efficient solution of (3) for any value of μ via the calculation

$$p = -R(\mu I + D)^{-1} R^t g. \tag{7}$$

It is worth mentioning here a possible alternative version of the above algorithm in which the curvilinear search is replaced by a more conventional Armijo-type line search [1] when G is positive definite. This is the kind of search normally used with a Newton method and it has the significant advantage that each trial point only involves a recalculation of the objective function rather than the solution of the linear system (3).

Outline Algorithm 2
Newton-like iteration using curvilinear and conventional searches

```
If $G$ is not positive definite
    determine the extreme eigenvalues of $G$, $\lambda_1$ and $\lambda_n$
    choose $\mu > -\lambda_n$ to use in (3)
    if necessary adjust $\mu$ iteratively until (3) gives a point $x + p(\mu)$ so that
                    $1 > \eta_1 \geq D_q \geq \eta_2 > 0$                    (8)
else
    set $s = 1$ and calculate the Newton step $p(0)$ by solving $Gp = -sg$
    if necessary adjust $s$ iteratively until $x + sp(0)$ gives
                    either $D_q \geq \eta_2 > 0$ or $D_l \geq \eta_2 > 0$
Start a new iteration with $x$ replaced by $x + sp(0)$
```

In the next section we shall consider some modifications to the method proposed in [11] which still conform to the above outline but seek to achieve similar results more effectively and economically.

2 Implementing the outline algorithms

We note first of all that it is not necessary to use the expensive RDR^t factorization to determine whether or not G is positive definite. We can instead attempt Choleski factorization which expresses G as the product LDL^t where L is lower triangular and D is a positive diagonal matrix. This factorization process fails if and only if G is not positive definite and hence is a useful diagnostic tool. It does not however yield any information about the eigenvalues of G and hence does not give any guidance about values of μ which will cause $(\mu I + G)$ to be positive definite. We therefore need a means of calculating (at least) the most negative eigenvalue λ_n.

The Power Method is a well-known iterative technique for finding the largest magnitude eigenvalue of a matrix A. The form of the Power Method used in the present work starts from an arbitrary unit vector v_0 and performs the following sequence iteratively

$$\lambda_{k+1} = v_k^t A v_k \tag{9}$$

$$v_{k+1} = \frac{Av_k}{||Av_k||_2} \tag{10}$$

until $|\lambda_{k+1} - \lambda_k| < \epsilon|\lambda_k|$.

(Some comments on possible drawbacks with this iterative scheme will appear at the end of this paper.) Suppose we apply (9), (10) to G and that the resulting eigenvalue is $\tilde{\lambda}$. If we then apply the power method a second time to the shifted matrix $\hat{G} = (G - \tilde{\lambda} I)$ and obtain $\hat{\lambda}$ then we can assume that *either*

$$\tilde{\lambda} = \lambda_1 \text{ and } \hat{\lambda} = \lambda_n - \tilde{\lambda}$$

or

$$\tilde{\lambda} = \lambda_n \text{ and } \hat{\lambda} = \lambda_1 - \tilde{\lambda}.$$

In either case we can conclude

$$\lambda_1 = max(\tilde{\lambda}, \hat{\lambda} + \tilde{\lambda}) \;\; and \;\; \lambda_n = min(\tilde{\lambda}, \hat{\lambda} + \tilde{\lambda}).$$

It is reasonable to expect this procedure to yield the extreme eigenvalues more cheaply than than the RDR^t factorization does. It should be noted that we are relatively immune to one of the standard drawbacks of the power method. For instance if there are several similarly sized large eigenvalues the iterations can be slow to identify the truly dominant one and then to converge to the relevant *eigenvector*. But we do not need the eigenvectors at all for our purposes and so it is sufficient for us to terminate the iterations once an eigenvalue estimate has settled down.

One potential drawback with this method of estimating eigenvalues is that it may be susceptible to classic cancellation error if for instance λ_1 is very large and λ_n is relatively small. If $\tilde{\lambda}$ is obtained as a good estimate of λ_1 then $\hat{\lambda}$ will give an estimate of $\lambda_n - \lambda_1$ within a tolerance of about $\epsilon|\lambda_n - \lambda_1|$. This does not necessarily imply high accuracy in the computed value of λ_n however. Thus if the true values are $\lambda_1 = 1000, \lambda_n = -1$ and if $\epsilon = 10^{-4}$ then the computed value of $\hat{\lambda}$ could imply an error of 0.1 in the estimate of λ_n. Fortunately it turns out that we do not need very accurate estimates of the extreme eigenvalues. The most important consideration is for λ_n to be reliably bounded below; and to a lesser extent it is useful for λ_1 to be reliably bounded above. We therefore choose to safeguard the eigenvalue estimation method as follows. Once we have computed values λ_1 and λ_n by the process just described we make small perturbations by defining

$$\Delta = \epsilon(|\tilde{\lambda}| + |\hat{\lambda}|)$$

then *adding* Δ to λ_1 and *subtracting* Δ from λ_n.

We now consider how a knowledge of λ_1 and λ_n can assist in the initial choice and subsequent adjustment of μ during the curvilinear search. We know that μ needs to be greater than $-\lambda_n$ but *how much* greater would constitute a reasonable choice? Some guidance can be obtained using the condition number of $(\mu I + G)$

$$\kappa = \frac{\lambda_1 + \mu}{\lambda_n + \mu}.$$

We note that $\kappa \to 1$ as $\mu \to \infty$ so a small condition number is associated with a large μ which in turn implies a small step length $||p(\mu)||$. Thus a cautious initial choice of μ could be based on a modest condition number, say $1 < \kappa < 10$, and using

$$\mu = \frac{\lambda_1 - \kappa\lambda_n}{\kappa - 1} \tag{11}$$

Increasing μ in order to perform an interpolation step is equivalent to using (11) with a reduced condition number – i.e.

Replace κ by $\beta\kappa + (1 - \beta)$ for some $\beta \in (0, 1)$

Similarly, to decrease μ for extrapolation we replace κ by $\gamma\kappa$ for some $\gamma > 1$

We must observe, of course, that the calculation (11) will fail in the exceptional case that all the eigenvalues of G are equal so that, in particular, $\lambda_1 = \lambda_n$ because the condition number of $\mu I + G$ will be 1 for all values of μ.

We can embody the above ideas in implementations of outline algorithms 1 and 2. These all require an initial arbitrary choice of κ_0 to determine a starting value for μ in the first curvilinear search. For subsequent iterations a starting value of κ can be deduced from the value which yielded the previous successful step.

Any implementation of Outline Algorithms 1 and 2 will also require choices for the scaling parameters β and γ used in the adjustment of μ. In addition, values must be specified for η_1 and η_2 in (8) for checking that a sufficient decrease in function has been obtained. And for algorithms which involve estimating eigenvalues via the iteration (9), (10) we need to fix the accuracy parameter ϵ.

In the next section we compare the performance of four implementations which are named as follows:

NIMP61 and NIMP62 both use the RDR^t factorization of G to obtain the eigenvalues of G and to solve the system (3) for any value of μ. NIMP61 proceeds as in Outline Algorithm 1 and NIMP62 follows Outline Algorithm 2 – i.e. NIMP61 always uses a curvilinear search but NIMP62 reverts to a standard Armijo line search when the Hessian is positive definite.

NIMP81 and NIMP82 both employ the power method approach to obtain the extreme eigenvalues of G and use Choleski factorization to solve the system (3) for any value of μ. NIMP81 proceeds as in Outline Algorithm 1 and NIMP82 follows Outline Algorithm 2.

It must be mentioned that NIMP81 and NIMP82 include an emergency fallback procedure in rare cases when the estimate for λ_n is so badly wrong that the value of μ given by (11) results in the coefficient matrix in (3) being non-positive definite. If this happens the current μ is repeatedly increased until $(\mu I + G)$ can be split into Choleski factors.

3 Computational experiments

The purpose of these numerical tests is primarily to discover if calculating eigenvalues by the power method is (a) reliable and (b) significantly more efficient than using the RDR^t factorization. A secondary objective is to see if the curvilinear search has any advantages over the more conventional Armijo line search once the iterations have reached a neighbourhood round a local minimum in which the Hessian remains positive definite.

The test problems we use are listed below. Several of them make use of a matrix Q which is defined as the Hilbert matrix with modified diagonal terms $i/(2*i-1)$

P1

$$f(x) = x^t x + 10(x^t Q x - 1)^2 \tag{12}$$

Starting point is $x_1 = 0.6, x_2 = -0.8, x_i = 0$ for $i = 3, ..., n$

P2

$$f(x) = -x^t x + 100(x^t Q x - 1)^2 \tag{13}$$

Starting point is $x_1 = -0.5, x_2 = -0.68, x_i = 0$ for $i = 3, ..., n$

P3

$$f(x) = x^t Q x + 4(x^t x - 1)^2 \tag{14}$$

Starting point is $x_1 = 0.87, x_2 = 0.57, x_i = 0$ for $i = 3, ..., n$

P4

$$f(x) = -x^tQx + 10(x^tx - 1)^2 \tag{15}$$

Starting point is $x_1 = -0.3, x_2 = 0.75, x_i = 0$ for $i = 3, ..., n$

P5

$$f(x) = 0.1x^tQx + exp(-x^tx + 1) \tag{16}$$

Starting point is $x_i = 0.1$ for $i = 1, ..., n$

P6

$$f(x) = \frac{10^4}{1 + x^tQx} \tag{17}$$

Starting point is $x_i = 10$ for $i = 1, ..., n$

P7

$$f(x) = \sum_{k=1}^{n} \frac{5x_k^2 - \frac{1}{3}x_k^3}{k} \tag{18}$$

Starting point is $x_i = 9$ for $i = 1, ..., n$

These test problems have been constructed so as to feature a maximum or saddle point quite close to a minimum. The given starting points are in – or very near to – a region where the objective function is non-convex and hence all the methods will have to escape from this region before they can revert to basic Newton iterations with a positive-definite Hessian.

The tables below compare the performance of the NIMP variants on the test problems listed above. The tables give comparisons in terms of iterations, functions calls and execution time. Iterations appear in the form $I(I_{npd})$ where I is the total number of iterations and I_{npd} is the number of iterations on which the Hessian was found to be non-positive definite.

The first table demonstrates the effectiveness of the eigenvalue estimation procedure. We set a very tight tolerance 10^{-8} for the convergence of the power method iterations. The other parameter choices are the fairly cautious values

$$\kappa_0 = 2, \eta_1 = 0.9, \eta_2 = 0.1, \gamma = 2, \beta = 0.5$$

We observe that, in terms of iterations and function calls, NIMP81 and NIMP82 return virtually identical results to the methods which use exact eigenvalues. In most cases the execution times of NIMP81 and NIMP82 are significantly less than those for NIMP61 and NIMP62 – although there is one example (problem P7) where the power method have proved relatively expensive. The reason is simply that the Hessian matrix for problem P7 is of diagonal form and hence the RDR^t factorization is trivial and therefore much quicker than the iterative approach.

Having established the reliability of our eigenvalue estimates we next consider the possibility to further reduce execution times by relaxing the convergence threshold on the power method. In the second table we use $\epsilon = 10^{-5}$ along with the same parameter choices as in table one.

$$\kappa_0 = 2, \eta_1 = 0.9, \eta_2 = 0.1, \gamma = 2, \beta = 0.5,$$

It is no great surprise that the run-times for the methods using approximate eigenvalues are much smaller than those for codes that use a full eigenvalue decomposition. What could not be taken for granted, however, is that the performance of the NIMP6 and NIMP8 codes should remain similar in terms of iterations and function calls even when the eigenvalue estimates are not to precise. For problems P1 and P2 the results are still virtually identical. On problems P3 and P4 however the lower accuracy values for λ_1 and λ_n obtained by NIMP81 and NIMP82 do somewhat change the speed of convergence compared with NIMP61 and NIMP62. (It is presumably a matter of chance that the change is for the worse on problem P3 and for the better on P4.)

We have not yet observed any significant differences between the methods with use curvilinear searches throughout and those which revert to an Armijo search when G is positive definite. Further testing may shed more light on this. Further testing will also shed useful light on the sensitivity of the method to the parameters $\kappa_0, \eta_1, \eta_2, \gamma$ and β.

		NIMP61	NIMP62	NIMP81	NIMP82
	n	Its/Fcs / t(sec)	Its/Fcs / t(sec)	Its/Fcs / t(sec)	Its/Fcs / t(sec)
P1	1000	9(2)/17 / 21.3	9(2)/17 / 21.7	9(2)/17 / 1.9	9(2)/17 / 1.8
P2	1000	15(2)/24 / 37.8	15(2)/24 / 37.4	15(2)/25 / 12.1	15(2)/24 / 2.3
P3	1000	21(2)/38 / 12.3	21(2)/35 / 12.2	21(2)/37 / 8.5	21(2)/35 / 5.2
P4	1000	11(4)/18 / 24.8	11(4)/18 / 24.3	11(4)/18 / 3.1	11(4)/18 / 3.1
P5	1000	13(6)/31 / 33.5	12(5)/26 / 29.0	13(6)/31 / 9.6	12(5)/26 / 9.1
P6	1000	15(12)/20 / 36.5	15(12)/10 / 36.6	15(12)/20 / 4.8	15(12)/20 / 4.3
P7	1000	15(12)/34 / 0.8	15(12)/34 / 0.7	15(12)/34 / 54.0	15(12)/34 / 73.0

Table 1: NIMP test results
using parameters $\kappa = 2, \eta_1 = 0.9, \eta_2 = 0.1, \gamma = 2, \beta = 0.5$ and $\epsilon = 10^{-8}$

4 Comparison with other methods

We have now shown that our improved algorithms can escape from the region of non-optimal stationary points just as effectively as – and in the case of NIMP81 and NIMP82 much more efficiently than – the original algorithm presented in

		NIMP61	NIMP62	NIMP81	NIMP82
	n	Its/Fcs / t(sec)	Its/Fcs / t(sec)	Its/Fcs / t(sec)	Its/Fcs / t(sec)
P1	1000	9(2)/17 / 21.3	9(2)/17 / 21.7	9(2)/17 / 1.8	9(2)/17 / 1.7
P2	1000	15(2)/24 / 37.8	15(2)/24 / 37.4	15(2)/25 / 2.5	15(2)/24 / 2.3
P3	1000	21(2)/38 / 12.3	21(2)/35 / 12.2	23(5)/33 / 3.6	22(4)/31 / 3.2
P4	1000	11(4)/18 / 24.8	11(4)/18 / 24.3	10(2)/17 / 1.9	10(2)/16 / 2.4
P5	1000	13(6)/31 / 33.5	12(5)/26 / 29.0	13(5)/26 / 3.5	11(5)/28 / 2.8
P6	1000	15(12)/20 / 36.5	15(12)/10 / 36.6	12(8)/21 / 2.5	12(8)/21 / 2.6
P7	1000	15(12)/34 / 0.8	15(12)/34 / 0.7	17(11)/35 / 5.3	16(11)/36 / 5.1

Table 2: NIMP test results
using parameters $\kappa = 2, \eta_1 = 0.9, \eta_2 = 0.1, \gamma = 2, \beta = 0.5$ and $\epsilon = 10^{-5}$

[11]. The obvious next step is to compare these algorithms on other Newton-like techniques.

One comparison that we would do well to dispose of quickly relates to a version of Newton's method that emerged in the folklore in the early days of optimization in the 1970s. This version simply reverts to using an Armijo-type search along the local steepest descent direction whenever the Hessian is found to be non positive definite. This approach has the advantage of great simplicity (and a relatively straightforward proof of global convergence); and it is plausible to suppose that it will quite often provide an effective escape from regions where $f(x)$ is non-convex. It could in fact be regarded as a bare minimum attempt to approximate a CSDP search.

The results in the table below show how a fairly simple implementation of this hybrid Newton/Steepest descent approach behaves on our test problems. While it is quite competitive on about half of them it clearly does very badly in terms of iterations and function calls on problems P3, P6 and P7 – so badly in fact on P7 that it cannot really claim to be a serious contender (although it should be noted that the simplicity of the method does mean that it remains competitive on run-time in spite of the relatively large numbers of iterations).
In order to provide more credible opposition for NIMP8 we turn to the trust region method implemented in *fminunc* in the MATLAB optimization toolbox [10] which uses a technique described in [3], [6]. We will denote this method by TR and we note that it is comparable with NIMP in that it uses the exact Hessian of the objective function and works with a form of trust region subproblem on every iteration. It does not however obtain an exact solution to the trust region subproblem – e.g. by solving (3) – but rather it restricts itself to a *two dimensional* subspace. This subspace is defined by the negative gradient $-g$ together with either an approximate Newton direction, $n \approx -G^{-1}g$ or a direction of negative curvature s, such that $s^TGs < 0$. One of the available versions of TR finds n or s, by applying a preconditioned conjugate gradient (PCG) method see [5] to the system $Gn = -g$. When the search is far from the optimum the PCG method may be terminated with quite a low accuracy

		Newton/steepest descent
	n	Its/Fcs / t(sec)
P1	1000	13(3)/17 / 1.6
P2	1000	11(1)/18 / 1.5
P3	1000	40(20)/44 / 4.4
P4	1000	14(5)/19 / 1.8
P5	1000	27(19)/40 / 2.9
P6	1000	15(11)/113 / 3.4
P7	1000	112(106)/124 / 2.6

Table 3: Newton/steepest descent test results

approximation to the Newton direction; and, in particular, if G is found to be non positive definite the PCG method returns a direction of negative curvature, rather than an approximate Newton direction. An alternative version replaces the economical but inexact PCG approach with a standard factorization method for solving $Gn = -g$.

In the table below we apply both versions of TR to the same problems as in the previous tables. In all cases we use the standard default settings for algorithm parameters.

		TR_{pcg}	TR_{fact}
	n	Its/Fcs / t(sec)	Its/Fcs t(sec)
P1	1000	16/17 / 1.2	13/14 / 2.2
P2	1000	14/15 / 1.7	16/17 / 2.6
P3	1000	24/25 / 1.7	22/23 / 4.2
P4	1000	11/12 / 0.8	11/12 / 1.4
P5	1000	24/25 / 1.4	18/19 / 2.9
P6	1000	25/26 / 1.9	24/25 / 6.7
P7	1000	20/21 / 0.6	26/27 / 1.9

Table 4: TR test results

Broadly speaking we can see that the NIMP methods are comparable with the results with TR as regards overall iterations. (The MATLAB toolbox code does not return any information about how many iterations detected negative curvature.) The NIMP codes typically use more function calls because TR does not perform any line searches – but this is sometimes compensated for by the fact that NIMP uses fewer iterations.

As regards execution times, NIMP81 and NIMP82 are clearly competitive with the factorization version of TR but they cannot outperform the PCG version.

What is now required is a more extensive testing exercise which using for example the CUTEr test set [9] that was the basis for the experiments in [11].

5 Some convergence issues

The algorithms we are discussing in this paper all revert to the Newton method in the neighbourhood of a local minimum. Therefore ultimate convergence properties are well-known. Global convergence however depends on us being able to show that the curvilinear line search offers similar properties to the Armijo type search [1].

Proofs of global convergence of optimization algorithms typically assume a line-search is carried out along a single descent direction on every iteration. For example, the Goldstein conditions require us at each iteration to choose a step s that satisfies

$$f(x) + (1-m)s \cdot g(x) < f(x+s) < f(x) + ms \cdot g(x)$$

where m is a global positive real value strictly between 0 and 0.5, and s is in the descent cone from x, meaning that if θ is the angle between s and $g(x)$ then $\theta > \theta_0$, where $\theta_0 > \pi/2$ is a global constant.

We have previously shown [7] that it is not necessary to find a single point that satisfies both Goldstein (or Wolfe) conditions simultaneously: it's enough to have two points that satisfy one condition each, and take the first one, provided their distances from the start point are bounded by a global ratio R.

These considerations apply on many of the NIMP iterations where the curvilinear search yields a μ and hence a $p(\mu)$ which satisfies both the left and right hand Goldstein condition. However the following situation can arise which requires closer examination.

If for some μ the new point does not satisfy the left hand condition then our strategy is to extrapolate by reducing μ. But suppose that the extrapolated step yields a point whose function value violates the right hand Goldstein condition. It would then possible to try an intermediate μ in the hope that it will satisfy both conditions. However what we actually do in our NIMP implementations is simply to accept the last point from which extrapolation was attempted.

Conventional convergence theory does not cover this strategy, partly because we are looking at function decrease for moves along two different search directions. However we can make use of a new insight presented here – namely that (a) these two points don't need to be on a straight line through the start point, and (b) the new point doesn't need to be either of these two points, and doesn't even need to be in the descent cone from the start point: instead the new point can be any point that decreases the function by a greater amount.

Consider the following **Prototype Algorithm:**

At each iteration starting at a point x construct steps s_a, s_b, and s_c with the following properties:

$$f(x + s_a) < f(x) + ms_a \cdot g(x)$$

$$f(x + s_b) > f(x) + (1 - m)s_b \cdot g(x)$$

$$f(x + s_c) \leq f(x + s_a)$$

In these expressions m is, as usual, a global positive real strictly between 0 and 0.5. We also assume s_a and s_b are both in the descent cone from x, so that $\theta_a > \theta_0, \theta_b > \theta_0$ where θ_a is the angle between s_a and $g(x)$, θ_b is the angle between s_b and $g(x)$, and $\theta_0 > \pi/2$ is a global constant. We further require that $\|s_b\| \leq R\|s_a\|$ where $R > 1$ is a global constant.
At the end of each iteration we take the new point to be $x + s_c$.

Note that we do not assume that s_c lies in the descent cone. Note also that in practice any two, or all three, of the steps s_a, s_b, s_c may be the same.

At any point x, there will be bounds A and B such that any s_a in the descent cone with

$$\|s_a\| < A$$

satisfies the condition on s_a, and any s_b in the descent cone with

$$\|s_b\| > B$$

satisfies the condition on s_b.

We can now prove the following **Theorem:**

The Prototype Algorithm is globally convergent under the usual assumptions on f, namely that f is bounded below, and is Lipschitz-continuously differentiable everywhere below the start point, ie in the set $\{x : f(x) \leq f(x_0)\}$.

Proof:

We have $f(x + s_b) > f(x) + (1 - m)s_b \cdot g(x)$, whence it follows that

$$f(x + s_b) - f(x) - g(x) \cdot s_b > -ms_b \cdot g(x).$$

For some $p : 0 < p < 1$ we have $g(x + ps_b) \cdot s_b = f(x + s_b) - f(x)$, andso

$$[g(x + ps_b) - g(x)] \cdot s_b = f(x + s_b) - f(x) - g(x) \cdot s_b > -ms_b \cdot g(x)$$

But the LHS is bounded by $K\|s_b\|^2$ where K is the Lipschitz constant for g. Hence $K\|s_b\| > -m\|g(x)\| \cos\theta_b$. (Recall that $\cos\theta_b < \cos\theta_0 < 0$).

We also have

$$\Delta f \equiv f(x) - f(x+s_c) \geq f(x) - f(x+s_a) > -ms_a \cdot g(x) = -m\|s_a\|\|g(x)\| \cos\theta_a$$

and $\|s_b\| \leq R\|s_a\|$ whence (recalling that $\cos\theta_a < \cos\theta_0 < 0$)

$$\Delta f > C\|g(x)\|^2 \text{ where } C = \frac{m^2 \cos^2\theta_0}{RK}.$$

Since f is bounded below, the sum of the LHS over all steps of the algorithm must converge. Since C is a global constant, it follows that $\|g(x)\|$ tends to zero as the algorithm proceeds. Hence as usual (by compactness) a subsequence of the x must converge to a point x^* with $g(x^*) = 0$.
qed

The application of this theorem to the search in NIMP is as follows. If an extrapolation step beyond a point which fails the left hand Goldstein condition gives rise to a new point that fails the right hand Goldstein condition then we can regard the extrapolation step as s_b in the theorem while the previous attempt can be treated as s_a. We can then use the fact that s_c may be the same as s_a to justify our tactic of accepting s_a.

6 Conclusions and further work

This working note has outlined encouraging progress towards a much more efficient implementation of the ideas introduced in [11]. We can now be said to have a working prototype but more extensive computational experiments are still required using the CUTEr test set [9] that was the basis for the experiments in [11]. In parallel with these experiments some of the looser aspects of algorithms NIMP81 and NIMP82 must be more precisely defined before we can do a thorough convergence analysis.

In the previous section we have shown how existing convergence results based on the Armijo-Goldstein line search conditions might be extended to cover the kind of curvilinear search used in NIMP81 and NIMP82. But there is at least one aspect of these searches which has not been made watertight. This concerns the case where we have performed several extrapolation steps and it is still the case that $D_q > 1$ but it is not possible to decrease μ (i.e. increase κ) any further without causing $(\mu I + G)$ to become unacceptably ill-conditioned and thereby risking the search direction p lying outside an acceptable descent cone. A possible way of proceeding in this case would be to revert to doing a conventional line search extrapolation along the current p; but we have not incorporated this possibility into any of the implementations reported in this paper. Another way of viewing this issue is to note that a full and formal definition of the NIMP8 algorithms would require an upper limit to be imposed on allowable values of the condition number κ.

Another computational matter concerns the robustness of the Power Method iteration (9), (10). Consider the situation where A is a 2-by-2 diagonal matrix with elements 1 and -1. Let v be the randomly chosen unit vector $v = (cos\theta, sin\theta)$. Then the first iteration of (9), (10) gives

$$\lambda = v^t A v = cos^2\theta - sin^2\theta = cos(2\theta)$$

and then the next eigenvector estimate is $Av = (cos\theta, -sin\theta)$ (which is unchanged by normalization).

On the next iteration, $Av = (cos\theta, sin\theta)$ and

$$\lambda = v^T A v = cos(2\theta)$$

as before and so the iteration has converged to an arbitrary value in the range [-1,1]. This appears to be a variation on the difficulty of chooseing μ on the basis of condition number of $(\mu I + G)$ in the special case when G has all its eigenvalues equal. A comprehensive implementation of NIMP8 needs to take account of this.

One underlying aspect of this research also needs to be kept in mind: namely, that it is our hope to extract some ideas from these algorithms which can still be used in techniques auch as quasi-Newton methods which do not depend on the availability of exact second derivatives.

The conclusions of these remaining investigations will appear in a further paper to be laid before the public as soon as possible.

References

[1] L. Armijo. “Minimization of functions having Lipschitz continuous first partial derivatives”. In: *Pacific J Math.* 16.1 (1966), pp. 1–3.

[2] C. A. Botsaris and D. H. Jacobson. “A Newton-type Curvilinear Search Method for Optimization”. In: *J. Maths Anal.Appl.* 54.1 (1976), pp. 217–229.

[3] M. A. Branch, T. F. Coleman, and Y. Li. “A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems”. In: *SIAM Journal on Scientific Computing* 21.1 (1999), pp. 1–23.

[4] A. A. Brown and M. C. Bartholomew-Biggs. “Some Effective Methods for Unconstrained Optimization Based on the Solution of Systems of Ordinary Differential Equations”. In: *J. Optim. Theory Appl.* 62.2 (1989), pp. 211–224.

[5] C. G. Broyden and M. T. Vespucci. “Krylov Solvers for Linear Algebraic Systems”. In: *Studies in Computational Mathematics* 11 (2004).

[6] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. “Approximate Solution of the Trust Region Problem by Minimization over Two Dimensional Subspaces”. In: *Mathematical Programming* 40 (1988), pp. 247–263.

[7] B. Christianson. “Global Convergence using De-linked Goldstein or Wolfe Linesearch Conditions”. English. In: *Advanced Modeling Optimization* 11.1 (2009), pp. 25–31.

[8] A. R. Conn, N. I. M. Gould, and P. T. Toint. “Trust Region Methods”. In: *MPS-SIAM Series on Optimization, Philadelphia* (2000).

[9] N. I. M. Gould, D. Orban, and Ph. L. Toint. “CUTEr and SifDec: A Constrained and Unconstrained Testing Environment, revisited”. In: *ACM Transactions on Mathematical Software* 29.4 (December 2003), pp. 373–394.

[10] MATLAB. *Optimization Toolbox, Version 7.10.0 (R2010a),* The MathWorks Inc. Natick, Massachusetts, 2010.

[11] Salah Beddiaf Michael Bartholomew-Biggs and Bruce Christianson. “A Comparison of methods for traversing regions of non-cnvexity in optimization problems”. In: *Numerical Algorithms* (2019).

[12] J. p. Vial and I. Zang. “Unconstrained Optimization by Approximation of the Gradient Path”. In: *Maths. Oper. Res.* 2.3 (1977), pp. 253–265.