Nonlinear Optimization and Related Topics, pp. 123-142
G. Di Pillo and F. Giannessi, Editors

Iterative methods for ill-conditioned linear systems from optimization

Nicholas I. M. Gould (n.gould@rl.ac.uk)
Department for Computation and Information, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England, EU

Abstract

Preconditioned conjugate-gradient methods are proposed for solving the ill-conditioned linear systems which arise in penalty and barrier methods for non-linear minimization. The preconditioners are chosen so as to isolate the dominant cause of ill conditioning. The methods are stablized using a restricted form of iterative refinement. Numerical results illustrate the approaches considered.

Keywords: penalty functions, ill-conditioning, preconditioning, conjugate gradients.

1 Introduction

Let A and H be, respectively, full-rank m by n ($m \leq n$) and symmetric n by n real matrices. Suppose furthermore that any nonzero coefficients in this data are *modest*, that is the data is $O(1)$.[1] We consider the iterative solution of the linear system

$$(H + A^T D^{-1} A)x_* = b \tag{1.1}$$

where b is modest and D is a positive definite diagonal matrix with small entries. Systems of the form (1.1) occur throughout optimization.

[1]We abuse notation here, since by $O(a)$, we mean a quantity which is at most a small constant > 1 times a.

Example 1.1. Given the equality constrained optimization problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f(x) \;\text{ subject to }\; c_i(x) = 0 \;\; (i = 1, \ldots, m) \tag{1.2}$$

and a positive penalty parameter μ, the quadratic penalty function is the composite function

$$\phi(x, \mu) = f(x) + \frac{1}{2\mu} \sum_{i=1}^{m} c_i(x)^2. \tag{1.3}$$

The quadratic penalty method traces a local minimizer $x(\mu)$ of ϕ for a sequence of μ converging to zero. Under mild conditions, the limit of the sequence of $x(\mu)$ gives a local solution of (1.2) [11]. Each sequential minimization is ideally performed using Newton's method. Writing $A^T(x) \overset{\text{def}}{=} \nabla_x c(x)$, the Newton equations are

$$\left(H(x, y^q(x, \mu)) + \frac{1}{\mu} A^T(x) A(x) \right) \Delta x = -g(x, y^q(x, \mu)), \tag{1.4}$$

where $g(x, y)$ and $H(x, y)$ are respectively the gradient $\nabla_x \ell(x, y)$ and Hessian matrix $\nabla_{xx} \ell(x, y)$ of the Lagrangian function $\ell(x, y) = f(x) - y^T c(x)$, and the Lagrange multiplier estimates $y^q(x, \mu) = -c(x)/\mu$. As it can be shown that $y_q(x(\mu), \mu)$ converge to Lagrange multipliers at the solution of (1.2), (1.4) is precisely of the form (1.1) with $D = \mu I$. □

Example 1.2. Given the inequality constrained optimization problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f(x) \;\text{ subject to }\; c_i(x) \geq 0 \;\; (i = 1, \ldots, m) \tag{1.5}$$

and a positive barrier parameter μ, the (logarithmic) barrier function is the composite function

$$\psi(x, \mu) = f(x) + \frac{1}{2\mu} \sum_{i=1}^{m} \log c_i(x). \tag{1.6}$$

The barrier method traces a local minimizer $x(\mu)$ of ψ for a sequence of μ converging to zero. As for example 1.1., under mild conditions, the limit of the sequence of $x(\mu)$ gives a local solution of (1.5) [11], and each sequential minimization may be performed by Newton's method. The Newton equations are now

$$\left(H(x, y^b(x, \mu)) + A^T(x) D^{-1}(x, \mu) A(x) \right) \Delta x = -g(x, y^b(x, \mu)), \tag{1.7}$$

where the Lagrange multiplier estimates $y_i^b(x, \mu) = \mu / c_i(x)$ converge to Lagrange multipliers at the solution of (1.5) as x approaches $x(\mu)$, and the diagonal matrix $D(x, \mu)$ has diagonal entries $c_i(x)/y_i^b(x, \mu)$. If the solution to (1.5) is strictly complementary, some entries of $D(x, \mu)$ converge to zero while the remainder approach infinity. Combining the terms for which $d_{ii}(x, \mu)$ approaches infinity (and thus for which $d_{ii}^{-1}(x, \mu)$ converges to zero) and the Hessian $H(x, y^b(x, \mu))$ in a composite H, (1.7) is again of the form (1.1) with D now being made up from those terms for which $d_{ii}(x, \mu)$ converges to zero. □

At a first glance, the presence of the rank m term $A^T D^{-1} A$ would suggest that accurate solutions to (1.1) are hard to obtain when D is small. Indeed, the eigenvalue spectrum splits into two segments, m large eigenvalues corresponding to the $A^T D^{-1} A$ term and the remaining $n - m$ modest eigenvalues from the portion of H lying in the null-space of A [22]. Fortunately, a number of authors ([4], [16]) have noted that accurate solutions are possible despite this ill-conditioning. A simple way of seeing this is to introduce "auxiliary" variables

$$y_* = D^{-1} A x_* \tag{1.8}$$

into (1.1) to obtain the augmented system

$$\begin{pmatrix} H & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} x_* \\ y_* \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{1.9}$$

The conditioning of this problem then depends on the matrix

$$K(D) = \begin{pmatrix} H & A^T \\ A & -D \end{pmatrix}, \tag{1.10}$$

and, for small D is related to that of the perturbation

$$K \equiv K(0) = \begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix}. \tag{1.11}$$

It is the matrix (1.11) which often reflects the "real" conditioning of the underlying problem (see [24] for a discussion of this for optimization applications). Significantly, M. Wright [25] and S. Wright [27] have recently shown that it is actually possible to obtain accurate solutions *directly* from (1.1) in many cases so long as a backward stable factorization of $H + A^T D^{-1} A$ is used—such an approach is commonplace, at least in linear programming circles, since then H vanishes and the computationally convenient Cholesky factors of the positive-definite matrix $A^T D^{-1} A$ obtained (see, for example, [1], or [26] for details).

The system (1.9) then suggests an attractive way of accurately determining x_*. Notice that (1.10) is symmetric and, for sufficiently small D, indefinite. Thus any of the stable symmetric, indefinite factorizations (see, [6], [5], or [12], in the dense case, and [10], or [9], in the sparse case) may be applied.

However, there are two difficulties with such an approach. The first is simply that, just because (1.10) is well conditioned, and just because there are stable methods for solving (1.9), it does not follow that we can compute all the components of the solution with high relative accuracy. Indeed, we would normally only expect to compute the large components of the solution to high accuracy, and to do so might require that we perform iterative refinement (see, for example, [20, Chapter 11]). However, (1.9)

suggests that y_* is modest, while, if we consider (1.8), it seems likely[2] that x_* is actually $O(\|D\|)$ rather than simply $O(1)$—we shall say that data of $O(\|D\|)$ is *small*. Thus, while we can compute y_* to high relative accuracy using the method we have suggested, the same is not likely to be true for x_*. In order to avoid this defect, suppose that $y_{\rm est}$ is a highly accurate approximation of y_*, and let $y_{\rm cor} = y_* - y_{\rm est}$. Then (1.9) may simply be rearranged to give

$$\begin{pmatrix} H & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} x_* \\ y_{\rm cor} \end{pmatrix} = \begin{pmatrix} b - A^T y_{\rm est} \\ D y_{\rm est} \end{pmatrix}. \tag{1.12}$$

But now, so long as $y_{\rm est}$ is a good approximation of y_*, $y_{\rm cor}$ must be small, and x_* will not be swamped by $y_{\rm cor}$ if a stable method is used to solve (1.12). This suggests we should compute y_* from (1.9), and then recover x_* from (1.12). This is reminiscent of iterative refinement, but since only part of the residual is used, we prefer to call it *iterative semi-refinement*. Note, that even this is not without difficulty since there may be significant cancellation when forming $b - A^T y_{\rm est}$. A similar method was proposed in the case $H = I$ and $D = 0$ for least squares problems by Businger and Golub [7] and analysed by Golub and Wilkinson [15]. Their conclusions were that such a method only performed well when the required x_* is significantly smaller than y_*, which is fortunately the case of interest here.

The second disadvantage only becomes apparent when m and n are large. For then, it sometimes happens that the factors of $K(D)$ are considerably denser than $K(D)$ itself. Indeed, this fill-in may ultimately cause the factors to exhaust the available computer memory and thus for the factorization to fail. For example, consider the variable-dimension problem `CVXQP3.SIF` from the CUTE test set [3]. Discarding the simple bounds, and factorizing the resulting matrix (1.10) with $D = 10^{-8}I$ using the Harwell Subroutine Library [19] sparse symmetric-indefinite factorization code `MA27` for increasing values of n, we find the following:

n	m	nnz K(D)	nonzeros in factors
100	75	783	7365
1000	750	7981	6200099
10000	7500	79981	$>$ 15721736 (failed)

Table 1.1: Applying `MA27` to the matrix $K(D)$ obtained from `CVXQP3.SIF`. Here "nnz $K(D)$" is the number of nonzeros in the upper triangular part of $K(D)$, while "nonzeros in factors" is the number of real words of storage required to hold the factors.

Thus, we see that the factorization of even moderate-sized problems may be out of the question, and we are forced to consider alternatives.

[2]This is, of course, not rigorous since Ax_* may actually be small because x_* lies close to the null-space of A with x_* being modest.

In this paper, we are concerned with alternative, iterative methods for solving (1.1). Two possibilities immediately present themselves. One could apply an iterative method directly to (1.1), or alternatively, one could use an iterative method to solve (1.9). Both alternatives have disadvantages. It is likely that the bad conditioning of (1.1) will adversely affect an iterative method unless a sophisticated preconditioner is applied. On the other hand, the indefinite nature of (1.10) restricts the choice of iterative methods that might be applied to (1.9); MINRES [23] is probably the method of choice in this case, but the conflict between the form of permissible preconditioners (symmetric and positive definite) and the form of (1.10) (symmetric and indefinite) is unfortunate (see, however, [13], for some possibilities). We will not explore this possibility further in this paper and shall concentrate on iterative methods for (1.1).

In Section 2, we shall consider conjugate-gradient methods for the problem. We start by reviewing the traditional preconditioned conjugate-gradient method, and discuss the form of preconditioners we shall allow. We then indicate that quantities generated by such preconditioners are modest despite the potential for large values. We also explain why some form of iterative refinement is needed when applying the preconditioner to ensure that the iterates behave properly. Finally, we propose variants on the basic method which have different matrix-vector product requirements, and indicate how economies may be made in the iterative refinement without jeopardising the iteration. In Section 3, we perform numerical comparisons between these various methods, using a variety of simple preconditioners. We draw our conclusions and suggest further developments in Section 4.

2 Preconditioned conjugate gradients

2.1 The traditional conjugate-gradient method

We shall suppose in what follows that $H + A^T D^{-1} A$ is positive definite. In this case, the method of (preconditioned) conjugate gradients may be described as follows. We suppose that the *preconditioner* W is a "simple"[3] positive definite matrix which approximates $H + A^T D^{-1} A$. and define the gradient

$$g(x) = (H + A^T D^{-1} A)x - b. \tag{2.13}$$

Then we may solve (1.1) by applying Algorithm 2.1.

This iteration is the traditional conjugate-gradient method applied to the system

$$W^{-1}(H + A^T D^{-1} A)x_* = W^{-1} b, \tag{2.21}$$

[3]Simple here means that $Wx = b$ is easy to solve.

Algorithm 2.1: The preconditioned conjugate-gradient method for (1.1)

Given $x = 0$, $g\ (\equiv g(0)) = -b$, $r = W^{-1}g$, $p = -r$ and $\sigma = r^T g$, perform the following iteration until convergence:

$$
\begin{aligned}
&\text{Form } (H + A^T D^{-1} A)p \\
\alpha &= \sigma / p^T (H + A^T D^{-1} A)p && (2.14)\\
x_{\text{new}} &= x + \alpha p && (2.15)\\
g_{\text{new}} &= g + \alpha (H + A^T D^{-1} A)p \quad (\equiv g(x_{\text{new}})) && (2.16)\\
r_{\text{new}} &= W^{-1} g_{\text{new}} && (2.17)\\
\sigma_{\text{new}} &= g_{\text{new}}^T r_{\text{new}} && (2.18)\\
\beta &= \sigma_{\text{new}} / \sigma && (2.19)\\
p_{\text{new}} &= -r_{\text{new}} + \beta p && (2.20)
\end{aligned}
$$

see, for instance, [14, Section 4.8.5.1]. The iteration is normally stopped when the "preconditioned" residual $\sigma = g^T r \equiv g^T W^{-1} g$ is smaller than some predefined tolerance (possibly relative to the initial "preconditioned" residual). In order to mimic the coefficient matrix $H + A^T D^{-1} A$, we choose preconditioners of the form $M + A^T D^{-1} A$. Luenberger [21, Chapter 12], suggested that, at the very least, the preconditioner should reflect the terms which lead to the ill-conditioning, and proposed a preconditioner of the form $I + A^T D^{-1} A$. The vector r_{new} required in (2.17) must be found as the solution to the system

$$(M + A^T D^{-1} A) r_{\text{new}} = g_{\text{new}}. \tag{2.22}$$

It may appear at first sight that (2.22) will be hard to solve accurately since the matrix $M + A^T D^{-1} A$ has both modest and large eigenvalues. However, as we saw in the introduction, this effect is illusory since, by introducing

$$s_{\text{new}} = D^{-1} A r_{\text{new}}, \tag{2.23}$$

equation (2.22) may be rewritten as

$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r_{\text{new}} \\ s_{\text{new}} \end{pmatrix} = \begin{pmatrix} g_{\text{new}} \\ 0 \end{pmatrix}. \tag{2.24}$$

Provided that M is a good approximation to H and D is small, the coefficient matrix for (2.24) approximates (1.11), which we have suggested reflects the natural conditioning of the problem. Hence both r_{new} and s_{new} are at most modest (they may be smaller) so long as g_{new} is.

2.2 Applying a special conjugate-gradient method to (1.1)

We now show that all computed quantities remain (at most) modest. To see this, suppose that x, g, p, s and

$$q = D^{-1}Ap \tag{2.25}$$

are modest. This is certainly the case at the start of the iteration, since $x = 0$, and $g = -b$, while the latter is modest by the assumptions we made on the problem data at the start of Section 1. Furthermore, $p = -r$, where r and s may be found from

$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r \\ s \end{pmatrix} = - \begin{pmatrix} g \\ 0 \end{pmatrix}, \tag{2.26}$$

which has a modest right-hand side, and thus both p and s are modest. Finally, $q = -s$ which is modest because s is. Turning to the iteration itself, (2.15) shows that x_{new} is modest since x and p are,[4] since (2.16) may be replaced by the equivalent

$$g_{\text{new}} = g + \alpha(Hp + A^T q). \tag{2.27}$$

It also follows, as we indicated at the end of the previous paragraph, that r_{new} and s_{new} are modest as g_{new} is. Finally (2.20) and (2.23) show that

$$q_{\text{new}} = -D^{-1}Ar_{\text{new}} + \beta q = -s_{\text{new}} + \beta q, \tag{2.28}$$

and hence q_{new} is also modest. It is worth noting that (2.25) suggests that p is actually small rather than modest, while (2.23) suggests the same is true for s_{new}.

Any worry that the D^{-1} term in the matrix-vector product $(H+A^T D^{-1}A)p$, which occurs in (2.14) and (2.16), might swamp the remaining contribution is unfounded since the product can be rewritten as the modest $Hp + A^T q$. The recurrence (2.28) is also significant algorithmically since (2.16) may be replaced by the less expensive (2.27).

In summary, we propose the following special algorithm, Algorithm 2.2, for solving (1.1).

Notice that there is no longer any need for matrix-vector products involving A, but just with A^T. However, also note that Algorithm 2.2 requires an extra vector of storage above that of its naive predecessor Algorithm 2.1.

[4] This is not strictly rigorous since we are making an implicit assumption here that α and, later, β are modest. In numerical tests, this always seems to be the case.

Algorithm 2.2: A special conjugate-gradient method for problem (1.1)

Given $x = 0$, set $g = -b$, solve

$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix} \tag{2.29}$$

and set $p = -r$, $q = -s$ and $\sigma = r^T g$. Perform the following iteration until convergence:

$$\begin{aligned} &\text{Form } Hp + A^T q \\ &\alpha = \sigma / p^T (Hp + A^T q) && (2.30) \\ &x_{\text{new}} = x + \alpha p && (2.31) \\ &g_{\text{new}} = g + \alpha(Hp + A^T q) && (2.32) \\ \text{Solve } &\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r_{\text{new}} \\ s_{\text{new}} \end{pmatrix} = \begin{pmatrix} g_{\text{new}} \\ 0 \end{pmatrix} && (2.33) \\ &\sigma_{\text{new}} = g_{\text{new}}^T r_{\text{new}} && (2.34) \\ &\beta = \sigma_{\text{new}} / \sigma && (2.35) \\ &\begin{pmatrix} p_{\text{new}} \\ q_{\text{new}} \end{pmatrix} = -\begin{pmatrix} r_{\text{new}} \\ s_{\text{new}} \end{pmatrix} + \beta \begin{pmatrix} p \\ q \end{pmatrix} && (2.36) \end{aligned}$$

2.3 An alternative conjugate-gradient method for (1.1)

There is one remaining concern over Algorithm 2.2, namely that the potentially small solution components r_{new} in (2.33) may not be computed very accurately, at least relative to the remaining modest components s_{new}. We have seen this problem in Section 1, and the cure in this case is essentially the same. The easiest solution is simply to apply iterative refinement to (2.29) and, at every iteration, to (2.33). A slightly less expensive idea is to note that, as s_{new} is modest, (2.23) indicates that r_{new} is likely of $O(\|D\|)$, and thus it may be preferable to apply iterative semi-refinement rather than iterative refinement to each system. However, it turns out that there is a significantly less expensive alternative.

Suppose that we transform (2.33) to

$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r_{\text{new}} \\ u_{\text{new}} \end{pmatrix} = \begin{pmatrix} g_{\text{new}} - A^T z_{\text{new}} \\ D z_{\text{new}} \end{pmatrix}. \tag{2.37}$$

where

$$s_{\text{new}} = z_{\text{new}} + u_{\text{new}}$$

and $z_{\rm new}$ is a good approximation to $s_{\rm new}$. Suppose further that the previous right-hand side

$$\begin{pmatrix} v \\ w \end{pmatrix} \stackrel{\rm def}{=} \begin{pmatrix} g - A^T z \\ Dz \end{pmatrix} \tag{2.38}$$

was small. Our aim is to choose $z_{\rm new}$ so that the right-hand side of (2.37) is also small. Consider the first component of this vector. We have that

$$\begin{aligned} v_{\rm new} \equiv & \; g_{\rm new} - A^T z_{\rm new} \\ = & \; g + \alpha(Hp + A^T q) - A^T z_{\rm new} \\ = & \; g - A^T z + \alpha Hp - A^T(z_{\rm new} - z - \alpha q) \\ = & \; v + \alpha Hp - A^T(z_{\rm new} - z - \alpha q) \end{aligned} \tag{2.39}$$

using (2.32) and (2.38). Now suppose we pick

$$z_{\rm new} = z + \alpha q. \tag{2.40}$$

Then, combining (2.39) and (2.40), we deduce that

$$v_{\rm new} = v + \alpha Hp. \tag{2.41}$$

But, as we have already noted, (2.25) suggests that p is small, and thus $v_{\rm new}$ is small if v is. Turning to the second component of the right-hand side of (2.37), (2.38) and (2.40) show that

$$w_{\rm new} \equiv Dz_{\rm new} = D(z + \alpha q) = w + \alpha Dq, \tag{2.42}$$

which is small, if w is, since q is modest. Furthermore, (2.23) and (2.38) give

$$r^T g = r^T w + r^T A^T z = r^T w + s^T Dz = r^T w + s^T v. \tag{2.43}$$

Since (2.14) may be written as the less expensive

$$\alpha = g^T r/(p^T Hp + q^T Dq), \tag{2.44}$$

and as the matrix-vector product Dq is needed to form the denominator of (2.44), the update (2.42) may be performed efficiently. Combining the above recurrences, we arrive at the following alternative to Algorithm 2.2, Algorithm 2.3.

Notice that there is now no longer any need for matrix-vector products involving A or A^T, nor do we maintain $g(x)$. However, also note that Algorithm 2.3 requires two more vectors of storage than Algorithm 2.2 and three more than Algorithm 2.1.

Algorithm 2.3 needs a suitable starting vector z. Probably the simplest choice is to proceed as we did in Section 1 by first solving

$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r \\ z \end{pmatrix} = \begin{pmatrix} -b \\ 0 \end{pmatrix}$$

to obtain z (discarding the inaccurate r). In this case, the initial v and w are both small.

Algorithm 2.3: An alternative conjugate-gradient method for (1.1)

Given $x = 0$ and an appropriate z for which $v = -b - A^T z$ and $w = Dz$ are small, solve

$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r \\ u \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix} \tag{2.45}$$

and set $s = z + u$, $p = -r$, $q = -s$ and $\sigma = r^T v + s^T w$. Perform the following iteration until convergence:

$$\text{Form } Hp \text{ and } Dq$$

$$\alpha = \sigma/(p^T Hp + q^T Dq) \tag{2.46}$$

$$\begin{pmatrix} x_{\text{new}} \\ z_{\text{new}} \\ v_{\text{new}} \\ w_{\text{new}} \end{pmatrix} = \begin{pmatrix} x \\ z \\ v \\ w \end{pmatrix} + \alpha \begin{pmatrix} p \\ q \\ Hp \\ Dq \end{pmatrix} \tag{2.47}$$

$$\text{Solve } \begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r_{\text{new}} \\ u_{\text{new}} \end{pmatrix} = \begin{pmatrix} v_{\text{new}} \\ w_{\text{new}} \end{pmatrix} \tag{2.48}$$

$$s_{\text{new}} = z_{\text{new}} + u_{\text{new}} \tag{2.49}$$

$$\sigma_{\text{new}} = r_{\text{new}}^T v_{\text{new}} + s_{\text{new}}^T w_{\text{new}} \tag{2.50}$$

$$\beta = \sigma_{\text{new}}/\sigma \tag{2.51}$$

$$\begin{pmatrix} p_{\text{new}} \\ q_{\text{new}} \end{pmatrix} = -\begin{pmatrix} r_{\text{new}} \\ s_{\text{new}} \end{pmatrix} + \beta \begin{pmatrix} p \\ q \end{pmatrix} \tag{2.52}$$

2.4 Cost of the algorithm

In Table 2.2, we compare the cost per iteration of the three algorithms given in this section, neglecting the (possibly not inconsiderable) cost of applying the preconditioner. Which variant proves to be the most cost effective depends on the density of the matrices involved.

Algorithm	vectors required	matrix-vector products	inner-products	axpys
2.1	$4(n)$, $1(m)$	H, A, A^T, D	$2(n)$	$4(n)$
2.2	$4(n)$, $2(m)$	H, A^T	$2(n)$	$4n$, $1(m)$
2.3	$4(n)$, $4(m)$	H, D	$2(n)$, $2(m)$	$5(n)$, 2 (m)

Table 2.2: Costs per iteration of Algorithms 2.1–2.3. The parentheses indicate the lengths of vectors involved.

2.5 Stablizing the conjugate-gradient method for (1.1)

In the development of the previous algorithm, there was a tacit assumption that if we balance the right-hand side of (2.48), then the solution to (2.48) will itself be balanced. Of course, this may not necessarily be the case, and we may still be forced to apply some form of iterative refinement if we wish to generate accurate solutions. Since we prefer iterative semi-refinement, we propose the following algorithm.

Algorithm 2.4: iterative semi-refinement: in (v, w, z)**, out** (r, u, v, w, z)

1. Solve
$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r \\ u \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix}$$
2. If necessary, update
$$\begin{pmatrix} v \\ w \\ z \end{pmatrix} \leftarrow \begin{pmatrix} v - A^T u \\ w + Du \\ z + u \end{pmatrix}$$
and resolve
$$\begin{pmatrix} M & A^T \\ A & -D \end{pmatrix} \begin{pmatrix} r \\ u \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix}$$

It is easy to see that we may replace (2.45) and (2.48) by calls to Algorithm 2.4. Indeed, the only relationship we need to check is that (2.50) is valid if Step 2 of Algorithm 2.4 is applied. If we (temporarily) denote the quantities computed in this step with a bar, we trivially have that $\bar{r} = r$ and $\bar{s} = s$, while

$$\bar{\sigma} = \bar{r}^T \bar{v} + \bar{s}^T \bar{w} = r^T (v - A^T u) + s^T (w + Du) = \sigma - u^T (Ar - Ds) = \sigma$$

follows from (2.26). In summary, our preferred algorithm may be stated as Algorithm 2.5.

The remaining issue is to decide when Step 2 of Algorithm 2.4 is required. We have experimented with a number of possibilities, but have seen little, if any, improvement over the simplest expedient of executing Step 2 whenever $\|r\|$ is significantly smaller than $\|u\|$ in Step 1. We have settled on executing Step 2 whenever

$$\|r\| \leq \|D\|^{0.5} \|u\|$$

Algorithm 2.5: A stablized conjugate-gradient method for (1.1)

Given $x = 0$, let $v = -b$, $w = 0$ and $z = 0$, apply Algorithm 2.4 to obtain (r, u, v, w, z), and set $s = z + u$, $p = -r$, $q = -s$ and $\sigma = r^T v + s^T w$. Perform the following iteration until convergence:

$$\text{Form } Hp \text{ and } Dq$$

$$\alpha = \sigma/(p^T Hp + q^T Dq) \tag{2.53}$$

$$\begin{pmatrix} x_{\text{new}} \\ z_{\text{new}} \\ v_{\text{new}} \\ w_{\text{new}} \end{pmatrix} = \begin{pmatrix} x \\ z \\ v \\ w \end{pmatrix} + \alpha \begin{pmatrix} p \\ q \\ Hp \\ Dq \end{pmatrix} \tag{2.54}$$

$$\text{Apply Algorithm 2.4 to obtain } (r_{\text{new}}, u_{\text{new}}, v_{\text{new}}, w_{\text{new}}, z_{\text{new}})$$

$$s_{\text{new}} = z_{\text{new}} + u_{\text{new}} \tag{2.55}$$

$$\sigma_{\text{new}} = r_{\text{new}}^T v_{\text{new}} + s_{\text{new}}^T w_{\text{new}} \tag{2.56}$$

$$\beta = \sigma_{\text{new}}/\sigma \tag{2.57}$$

$$\begin{pmatrix} p_{\text{new}} \\ q_{\text{new}} \end{pmatrix} = -\begin{pmatrix} r_{\text{new}} \\ s_{\text{new}} \end{pmatrix} + \beta \begin{pmatrix} p \\ q \end{pmatrix} \tag{2.58}$$

as an acceptable heuristic.

We remark that Algorithm 2.4 could equally well be applied to Algorithm 2.2. However, Algorithm 2.2, unlike Algorithm 2.3, makes no effort to balance the right-hand sides of the preconditioning systems (2.29) and (2.33), and thus Step 2 of Algorithm 2.4 will likely be necessary at every iteration.

3 Numerical Experiments

In this section we compare Algorithms 2.1–2.3 using a variety of preconditioners on a number of standard optimization test problems. The problems we use arise as the larger convex quadratic programs—that is problems of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ \ \tfrac{1}{2} x^T H x + c^T x \ \text{ subject to } \ Ax = b \ \text{ and } \ l \leq x \leq u$$

for given vectors b, c, l and u—within the CUTE test set [3]. For those examples which lie in a class of similar problems, a representative is chosen. From these examples, we discard the data b and c and add a specified amount (in our examples, 0.1) to the i-th diagonal of H if either of l_i or u_i is finite—the intention here is to try to mimic

the type of Hessian matrix which might have arisen from a mixed penalty-barrier function for the quadratic program, and the entries on the diagonal correspond to barrier terms for the simple bounds $l \leq x \leq u$. We now construct the vector b in (1.1) by specifying x_* and setting $y_* = D^{-1}Ax_*$ and forming $b = Hx_* + A^T y_*$. Since we are presuming that b should be modest, this requires that the same is true of y_*, and so suggests that we should choose $x_* = O(\|D\|)$. In our experiments, we pick $D = \mu I$ for some small μ and $x_* = \mu e$, where e is the vector of ones. We report on those experiments for which $\mu = 10^{-8}$. Similar results were obtained with random small D.

All experiments were performed in double precision on an IBM RISC System/6000 3BT workstation with 64 Megabytes of RAM, using the xlf90 compiler and optimization level -O3. All matrix factorizations are performed using the forthcoming Harwell Subroutine Library sparse symmetric-indefinite factorization module `HSL_MA27`. The iteration is stopped as soon as the test

$$\sigma \leq \max(\epsilon_r \cdot \sigma_0, \epsilon_a)$$

is satisfied. Here σ_0 is the initial value of σ, while ϵ_r and ϵ_a are small constants—in our experiments, $\epsilon_r = 10^{-12}$, while ϵ_a is set to the machine precision ($\approx 10^{-16}$). A run is considered unsuccessful whenever more than $2(n-m+1)$ iterations have occurred, or more than 1800 CPU seconds are needed, or more than one million words of real storage is required for the factors of the preconditioner.

We first illustrate the need for preconditioning. In Table 3.3, we compare the solution obtained using the basic Algorithm 2.1 without preconditioning, with those from the same algorithm using Luenberger's preconditioner $M = I$, as well as Algorithm 2.2 with the same preconditioner. A single step of iterative refinement is applied, since there are frequent failures when the preconditioning systems without such a precaution.

We make the following observations. Firstly, as one might expect, even a naive preconditioner $M = I$ provides a dramatic reduction in the number of iterations performed. Of course, a reduction in the iteration count does not necessarily result in a reduction in the total CPU time required (see, for example, `AUG3DCQP`), since the cost of forming and applying the preconditioner is often non-negligible. When it is possible to form the factors of the preconditioner in Algorithm 2.1, this algorithm is often, but not always, faster than Algorithm 2.2. However, the number of failures due to the density of the factors of $W = I + A^T D^{-1} A$ limits the use of such a method. In this sense, Algorithm 2.2 is to be preferred, since (at least for these examples) the algorithm always succeeded. Perhaps the best strategy would be to use Algorithm 2.1 so long as a factorization of $M + A^T D^{-1} A$ is possible, keeping Algorithm 2.2 in reserve for the remaining problems. However, using Algorithm 2.1 with more sophisticated preconditioners is even more likely to fail through lack of storage space.

Next, we consider using slightly more general, but nonetheless rather unsophisticated, preconditioners. The first is simply to use $M = H$, which is equivalent to

			Algorithm 2.1						Algorithm 2.2		
			$W = I$			$M = I$			$M = I$		
Problem	n	m	err	iter	cpu	err	iter	cpu	err	iter	cpu
AUG2DCQP	20200	10000	-16	484	40	-16	3	5	-16	3	3
AUG2DQP	20200	10000	-12	1422	118	-15	13	7	-15	13	6
AUG3DCQP	27543	8000	-15	144	16		mem	16	-15	3	18
AUG3DQP	27543	8000	-11	353	38		mem	16	-15	11	22
BLOCKQP1	20006	10001	-8	2	0		mem	0	-8	1	2
BLOWEYA	20002	10002	-8	8112	771		mem	0	-8	10	3
CVXQP1	15000	7500		maxit	1202	-16	2167	260	-16	2173	523
GOULDQP2	19999	9999	-10	295	23	-14	39	5	-14	39	10
KSIP	10021	10001	-8	14	2	-10	32	8	-9	31	11
MOSARQP1	30000	10000	-10	479	55	-14	64	23	-14	64	36
SOSQP1	20000	10001	-21	1	0		mem	0	-12	2	1
STCQP2	8193	4095	-11	779	46	-15	157	11	-15	157	18
UBH1	18009	12000	-8	10082	740	-8	3589	402	-8	3178	770
YAO	20002	10000	-10	93	7	-14	14	2	-14	14	4

Table 3.3: Primitive preconditioners. Key: "n, m" = number of columns and rows of A, "err" = $log_{10}\|x - x_*\|$, "iter" = number of iterations, "cpu" = CPU time required in seconds, "maxit" indicates that more than $2(n - m + 1)$ iterations were required, "mem" indicates that the factors required more than one million words of storage.

solving the problem using a direct method. At the other extreme, the second preconditioner used is simply to pick M as the diagonal of H. A third choice is barely more sophisticated, in which M is a band matrix whose entries within the band correspond to those of H, except that the i-th and j-th diagonal entries for each term h_{ij} outside the band are augmented by $\sqrt{|h_{ij}|}$ [2]: we consider both diagonal and tridiagonal bands enhanced in this way. The effect of these preconditioners on Algorithm 2.2 is illustrated in Table 3.4; the final columns from the previous table are repeated for reference purposes.

Now we see the effect, as well as the cost, of more sophisticated preconditioning. In general, the more M is allowed to reflect the true structure of H, the better (in general) is the convergence behaviour, and conversely, the more chance that there will be insufficient room to hold the factors. In particular, the choice $M = H$ is most effective when there is sufficient space, while the diagonal and, especially, the enhanced diagonal preconditioners curb the wilder behaviour of the identity preconditioners with little loss in reliability. Ideally, a form of incomplete factorization, in which entries/fill-ins from H, but not A or D, might be removed, would seem to be a useful possibility, and we are currently investigating such ideas.

Finally, in Table 3.5, we examine whether the stablized Algorithm 2.5 is effective.

Problem	$M = I$			$M = H$			$M =$ diagonal of H			$M =$ enhanced diagonal of H			$M =$ enhanced tridiagonal of H		
	err	iter	cpu	err	iter	cpu	err	iter	cpu	err	iter	cpu	err	iter	cpu
AUG2DCQP	-16	3	3	-16	1	2	-16	1	2	-16	1	2	-16	1	2
AUG2DQP	-15	13	6	-15	1	2	-15	1	2	-15	1	2	-15	1	2
AUG3DCQP	-15	3	18	-15	1	17	-15	1	17	-15	1	17	-15	1	17
AUG3DQP	-15	11	22	-15	1	17	-15	1	17	-15	1	17	-15	1	16
BLOCKQP1	-8	1	2		cond	2	-8	3	2	-8	1	2		cond	2
BLOWEYA	-8	10	3		mem	110		mem	90	-9	35	10		mem	115
CVXQP1	-16	2173	523		mem	15		mem	9		mem	9		mem	9
GOULDQP2	-14	39	10	-15	2	2	-15	19	6	-14	19	6	-15	2	2
KSIP	-9	31	11	-11	2	4	-11	2	4	-11	2	4	-11	2	4
MOSARQP1	-14	64	36	-15	2	6	-14	9	9	-15	12	11	-15	12	11
SOSQP1	-12	2	1	-13	2	2	-13	2	2	-12	1	1	-13	2	1
STCQP2	-15	157	18		mem	6	-15	35	5	-14	34	6	-14	33	6
UBH1	-8	3178	770	-13	2	2	-13	2	2	-13	2	2	-13	2	2
YAO	-14	14	4	-16	1	1	-16	1	1	-16	1	2	-16	1	1

Table 3.4: Using Algorithm 2.2 with more sophisticated preconditioners. Key: "err" $= log_{10}\|x-x_*\|$, "iter" = number of iterations, "cpu" = CPU time required in seconds, "cond" indicates that the matrix was too ill-conditioned for iterative refinement to converge, "mem" indicates that the factors required more than one million words of storage.

We observe that the method is no less effective or reliable than its predecessors, but offers a significant CPU time saving in some cases. For example, compare CVXQP1,

Problem	$M = I$			$M = H$			$M =$ diagonal of H			$M =$ enhanced diagonal of H			$M =$ enhanced tridiagonal of H		
	err	iter	cpu	err	iter	cpu	err	iter	cpu	err	iter	cpu	err	iter	cpu
AUG2DCQP	-17	3 (3)	3	-17	1 (2)	2	-17	1 (2)	2	-17	1 (2)	2	-17	1 (2)	2
AUG2DQP	-15	13 (2)	4	-16	1 (2)	2	-16	1 (2)	2	-16	1 (2)	2	-16	1 (2)	2
AUG3DCQP	-15	3 (3)	18	-15	1 (2)	17	-15	1 (2)	17	-15	1 (2)	17	-15	1 (2)	17
AUG3DQP	-15	11 (2)	20	-16	1 (2)	17	-16	1 (2)	17	-16	1 (2)	17	-16	1 (2)	17
BLOCKQP1	-8	2 (1)	2		c (0)	2	-7	2 (2)	2	-8	1 (2)	2		c (0)	2
BLOWEYA	-8	10 (11)	4		m (0)	109		m (0)	90	-10	17 (11)	5		m (0)	115
CVXQP1	-13	2456 (16)	364		m (0)	15		m (0)	9		m (0)	9		m (0)	9
GOULDQP2	-14	39 (2)	7	-15	1 (2)	2	-14	19 (2)	4	-14	19 (2)	4	-15	1 (2)	2
KSIP	-8	28 (2)	7	-8	1 (1)	4	-8	1 (1)	4	-8	1 (1)	4	-8	1 (1)	4
MOSARQP1	-14	64 (2)	23	-15	1 (1)	6	-14	9 (1)	8	-15	11 (1)	8	-15	11 (1)	8
SOSQP1	-20	1 (2)	1	-20	1 (2)	1	-20	1 (2)	1	-20	1 (2)	1	-20	1 (2)	1
STCQP2	-16	160 (3)	12		m (0)	6	-15	35 (4)	3	-14	34 (5)	3	-14	33 (4)	3
UBH1	-8	4536 (4)	674	-11	1 (2)	2	-11	1 (2)	2	-11	1 (2)	2	-11	1 (2)	2
YAO	-14	14 (2)	3	-16	1 (1)	1	-16	1 (1)	1	-16	1 (1)	1	-16	1 (1)	1

Table 3.5: Using Algorithm 2.5 with more sophisticated preconditioners. Key: "err" $= log_{10}\|x - x_*\|$, "iter" = number of iterations, with the number of iterative semi-refinements in brackets, "cpu" = CPU time required in seconds, "c" indicates that the matrix was too ill-conditioned for iterative refinement to converge, "m" indicates that the factors required more than one million words of storage.

MOSARQP1, STCQP2 and UBH1 when $M = I$, and BLOWEYA, GOULDQP2, MOSARQP1 and STCQP2 when M is the enhanced diagonal of H, in Tables 3.4 and 3.5. Notice, in particular, how few iterations require iterative semi-refinement when $M = I$. For the other cases, the total number of iterations is rarely high enough for these savings to become apparent in the CPU times, but for those examples where a significant reduction in CPU time occurs, there is a corresponding drop in the number of iterative semi-refinements required (see, for instance, STCQP2 when M is the enhanced diagonal of H).

4 Comments and perspectives

In this paper, we have shown that the iterative solution of highly ill-conditioned structured linear systems from optimization is possible even when the application of a direct method to the original system, or to its related augmented system, fails through lack of memory. We believe that the preconditioner must reflect the dominant form of ill-conditioning, while efforts to reflect the remaining structure of the coefficient matrix are also worthwhile. When applying such preconditioners, care must be taken to ensure that the solution is computed accurately. This may be achieved by some form of iterative refinement, and our preferred Algorithm 2.5 aims to lessen the need for refinement at every iteration.

We have not yet examined the best choice for the matrix M in the preconditioner, although it is clear that some compromise between the goal of controlling the non-dominant conditioning, and the cost of forming and manipulating the resulting preconditioner must be reached. Our current belief is that some form of incomplete factorization in which entries or fill-ins from H, but not from A or D, may be dropped is worth investigating in the future.

Of course, we have made a tacit assumption that $H + A^T D^{-1} A$ is positive definite throughout this paper. In optimization applications, systems of the form (1.1) most readily appear as necessary optimality conditions for the problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \tfrac{1}{2}\, x^T (H + A^T D^{-1} A) x - b^T x.$$

We may remove the restriction that $H + A^T D^{-1} A$ be positive definite in these cases so long as we introduce some other means of "convexifying" the problem; the most popular technique is to impose a "trust-region" constraint of the form $\|x\| \leq \Delta$, for some $\Delta > 0$. Such ill-conditioned trust-region problems have been considered by Coleman and Hempel [8] when the dimension is small, but the challenge is to construct effective iterative methods for solving larger problems. Our belief is that this is possible by combining the methods proposed here with the Generalized Lanczos Trust-Region method recently proposed by Gould, Lucidi, Roma and Toint [18].

Similar methods for quadratic programming (where $D = 0$) are currently being considered [17].

Acknowledgement

The author is grateful to Iain Duff for his comments and suggestions on an earlier draft of this paper. Useful conversations with Jorge Nocedal and Steve Wright at the 1998 Erice workshop on Nonlinear Optimization and Applications are also appreciated.

References

[1] E. D. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior Point Methods in Mathematical Programming*, pages 189–252, Dordrecht, The Netherlands, 1996. Kluwer Academic Publishers.

[2] M. A. Aziz and A. Jennings. A robust incomplete Choleski conjugate gradient algorithm. *International Journal on Numerical Methods in Engineering*, 20:949–966, 1984.

[3] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[4] C. G. Broyden and N. F. Attia. A smooth sequential penalty function method for nonlinear programming. In A. V. Balakrishnan and M. Thomas, editors, *11th IFIP Conference on System Modelling and Optimization*, number 59 in Lecture Notes in Control and Information Sciences, pages 237–245, Heidelberg, Berlin, New York, 1984. Springer Verlag.

[5] J. R. Bunch and L. C. Kaufman. Some stable methods for calculating inertia and solving symmetric linear equations. *Mathematics of Computation*, 31:163–179, 1977.

[6] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, 1971.

[7] P. Businger and G. H. Golub. Linear least squares solutions by Housholder transformations. *Numerische Mathematik*, 7:269–276, 1965.

[8] T. F. Coleman and C. Hempel. Computing a trust region step for a penalty function. *SIAM Journal on Scientific Computing*, 11(1):180–201, 1990.

[9] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *Transactions of the ACM on Mathematical Software*, 9(3):302–325, 1983.

[10] I. S. Duff, J. K. Reid, N. Munksgaard, and H. B. Neilsen. Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. *Journal of the Institute of Mathematics and its Applications*, 23:235–250, 1979.

[11] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. J. Wiley and Sons, Chichester, England, 1968. Reprinted as *Classics in Applied Mathematics 4*, SIAM, Philadelphia, USA, 1990.

[12] R. Fletcher. Factorizing symmetric indefinite matrices. *Linear Algebra and its Applications*, 14:257–272, 1976.

[13] P. E. Gill, W. Murray, D. B. Ponceléon, and M. A. Saunders. Preconditioners for indefinite systems arising in optimization. *SIAM Journal on Matrix Analysis and Applications*, 13(1):292–311, 1992.

[14] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.

[15] G. H. Golub and J. H. Wilkinson. Note on iterative refinement of least squares solutions. *Numerische Mathematik*, 9:139–148, 1966.

[16] N. I. M. Gould. On the accurate determination of search directions for simple differentiable penalty functions. *IMA Journal of Numerical Analysis*, 6:357–372, 1986.

[17] N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. Technical Report RAL-TR-98-069, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 1998.

[18] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

[19] Harwell Subroutine Library. *A catalogue of subroutines (release 12)*. AEA Technology, Harwell, Oxfordshire, England, 1995.

[20] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, USA, 1996.

[21] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, second edition, 1984.

[22] W. Murray. Analytical expressions for eigenvalues and eigenvectors of the Hessian matrices of barrier and penalty functions. *Journal of Optimization Theory and Applications*, 7:189–196, 1971.

[23] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[24] S. M. Robinson. Generalized equations and their solutions. 2. applications to non-linear programming. *Mathematical Programming Studies*, 19:200–221, 1982.

[25] M. H. Wright. Ill-conditioning and computational error in interior methods for nonlinear programming. *SIAM Journal on Optimization*, 9(1):84–111, 1999.

[26] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, USA, 1997.

[27] S. J. Wright. Effects of finite-precision arithmetic on interior-point methods for nonlinear programming. Technical Report MCS-P705-0198, Argonne National Laboratory, Argonne, Illinois, USA, 1998.