The University of Reading
Department of Mathematics

A note on permutations for quadratic programming problems

H. S. Dollar

Numerical Analysis Report 5/06

May 17, 2006

A note on permutations for quadratic programming problems

H. S. Dollar[1]

Abstract

Saddle-point problems frequently arise in many applications in science and engineering. Dollar, Gould, Schilders and Wathen recently proposed the use of implicit factorization constraint preconditioners for use within iterative methods when (approximately) solving saddle-point problems. Such a preconditioner may we require a preprocessing step which carries out a (symmetric) permutation of the saddle-point problem. We investigate several permutations which are motivated by the properties of saddle-point matrices arising in a class of constrained optimization problems.

1 Introduction

Suppose we wish to solve saddle-point problems of the form

$$\underbrace{\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix}}_{\mathcal{H}} \begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} c \\ d \end{bmatrix}}_{b}, \tag{1}$$

where $H \in \mathbb{R}^{n\times n}$ is symmetric, and $A \in \mathbb{R}^{m\times n}$ has full rank m. There are many applications in which the solution of such systems is required, [1]. We shall concentrate on the application of quadratic programming problems:

$$\min_x \frac{1}{2} x^T Q x + g^T x \quad \text{subject to} \quad Ax - d = 0,\ x \leq 0. \tag{2}$$

Such problems can be solved by interior point methods and every iteration of such a method will require the solution of a system of the form (1) where $H = Q + \Theta^{-1}$ and Θ is a diagonal matrix with strictly positive elements. As the interior point method approaches the solution of (2), some of the entries of Θ^{-1} grow as $\mathcal{O}(\mu)$ and others as $\mathcal{O}(\frac{1}{\mu})$, where $\mu > 0$ is called the barrier parameter and tends towards zero as the optimal solution is approached, [2].

When solving such saddle-point systems, we'd like to use a projected preconditioned conjugate gradient (PPCG) method with a constraint preconditioner of the form

$$P = \begin{bmatrix} 0 & 0 & A_1^T \\ 0 & H_{22} & A_2^T \\ A_1 & A_2 & 0 \end{bmatrix}, \tag{3}$$

[1]Department of Mathematics, University of Reading, Whiteknights, P.O. Box 220, Reading, Berkshire, RG6 6AX, U.K. (H.S.Dollar@reading.ac.uk).

where $A_1 \in \mathbb{R}^{m\times m}$ is nonsingular, $A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$, and $H_{22} = H_{m+1:n,m+1:n} \in \mathbb{R}^{(n-m)\times(n-m)}$. Clearly, when using this preconditioner we will only need to solve much smaller systems involving A_1 (and its transpose) and H_{22}. Such a preconditioner forms one of the implicit factorization preconditioners proposed by Dollar, Gould, Schilders and Wathen [4]; all of these preconditioners require A_1 to be nonsingular. In practice, we know that A is of full rank but we certainly cannot assume that the first m columns of A are linearly independent. However, by the assumption that A is of full rank and $m \leq n$, we can always find an n by n permutation matrix Π such that $A\Pi = \widehat{A}$ and the first m columns of $\widehat{A}$ are linearly independent. Letting

$$\widehat{H} = \Pi^T H \Pi,$$

we then solve

$$\underbrace{\begin{bmatrix} \widehat{H} & \widehat{A}^T \\ \widehat{A} & 0 \end{bmatrix}} \begin{bmatrix} \widehat{x} \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} \Pi^T c \\ d \end{bmatrix}} \tag{4}$$

and set

$$x = \Pi \widehat{x}.$$

We will then use a preconditioner of the form

$$\widehat{P} = \begin{bmatrix} 0 & 0 & \widehat{A}_1^T \\ 0 & \widehat{H}_{22} & \widehat{A}_2^T \\ \widehat{A}_1 & \widehat{A}_2 & 0 \end{bmatrix}. \tag{5}$$

In the following sections we will consider the desirable properties of such a permutation and propose several methods for obtaining them. We will also compare these methods numerically by solving a set of quadratic programming problems.

2 Desirable properties of the permutation

In the D.Phil thesis of Dollar [3], she investigated some of the desirable properties of the permutation Π. In particular, she found that the permutation should aim to move the large diagonal entries of H into $\widehat{H}_{22}$ since the preconditioned system $\widehat{P}^{-1}\widehat{\mathcal{H}}$ has

- an eigenvalue at 1 with multiplicity $2m$,
- $n - m$ eigenvalues defined by the generalized eigenvalue problem

$$\left(\widehat{H}_{22} + \widehat{A}_2^T \widehat{A}_1^{-T} \widehat{H}_{11} \widehat{A}_1^{-1} \widehat{A}_2 - \widehat{A}_2^T \widehat{A}_1^{-T} \widehat{H}_{21}^T - \widehat{H}_{21} \widehat{A}_1^{-1} \widehat{A}_2\right) x_z = \lambda \widehat{H}_{22} x_z. \tag{6}$$

The above statements follow from [6, Theorem 2.1] in which we use the fundamental nullspace basis of $\widehat{A}$:

$$\widehat{Z} = \begin{bmatrix} -\widehat{A}_1^{-1}\widehat{A}_2 \\ I \end{bmatrix}.$$

Remark 2.1. *Finding such a permutation closely relates to the problem of finding a nullspace basis of A, since if A_1 is nonsingular, then we can form the fundamental nullspace.*

If the permutation Π moves all the large diagonal entries of H into $\widehat{H}_{22}$, then the eigenvalues of (6) will start to cluster around 1 as we move towards the optimal solution of (2) and the preconditioned system will be well conditioned. Conversely, if all large diagonal entries of H were moved into $\widehat{H}_{11}$, the eigenvalues of (6) would have magnitude $\mathcal{O}(\frac{1}{\mu})$ and the preconditioned system will be very ill-conditioned.

3 Permutation methods

Dollar [3] firstly found a permutation $\overline{\Pi}_1$ such that the diagonal entries of $\overline{\Pi}_1^T H \overline{\Pi}_1$ are sorted in non-decreasing order. A permutation $\overline{\Pi}_2$ was then obtained by carrying out an LU factorization of $\overline{\Pi}^T A^T$ with threshold row pivoting. The hope was that by using a threshold we could reduce the number of large diagonal entries of H that were moved back into $\widehat{H}_{11}$, where $\Pi = \overline{\Pi_1 \Pi_2}$. Unfortunately $\overline{\Pi}_2$ was frequently far from being the identity so many of the large entries ended up in $\widehat{H}_{11}$.

In an attempt to improve on this method she also looked at finding a permutation $\widetilde{\Pi}_1$ such that the diagonal entries of $\widetilde{\Pi}_1^T H \widetilde{\Pi}$ are sorted in non-increasing order. As before, a permutation $\widetilde{\Pi}_2$ is obtained by carrying out an LU factorization of $\widetilde{\Pi}^T A^T$ with threshold row pivoting. The resulting permutations $\overline{\Pi} = \overline{\Pi}_1 \overline{\Pi}_2$ and $\widetilde{\Pi} = \widetilde{\Pi}_1 \widetilde{\Pi}_2$ are then compared and Π chosen by carrying out the following steps:

$d_1 = \texttt{diag}(\widetilde{\Pi}^T H \widetilde{\Pi})$
$d_2 = \texttt{diag}(\overline{\Pi}^T H \overline{\Pi})$
$r_1 = \frac{\texttt{mean}(d_1(m+1:m+n))}{\texttt{mean}(d_1(1:m))}$
$r_2 = \frac{\texttt{mean}(d_2(m+1:m+n))}{\texttt{mean}(d_2(1:m))}$
if $r_1 > r_2$ **then**
  $\Pi = \widetilde{\Pi}$
else
  $\Pi = \overline{\Pi}$
end if

Although carrying out such a method to find a permutation was generally a lot more effective than choosing a permutation which either ignores the diagonal entries of H, or just automatically sets $\Pi = \overline{\Pi}$ or $\Pi = \widetilde{\Pi}$ without comparing

the two, the numerical tests carried out in the thesis [3] imply that there might be another method to find a permutation Π which, for convenience, can be coded using standard MATLAB® functions and is more effective than the above method.

Let us consider how the LU factorization command `[L,U,P] = lu(A,thresh)` works in MATLAB®. The variable `thresh` is a pivot threshold in [0,1]. Pivoting occurs when the diagonal entry in a column has magnitude less than `thresh` times the magnitude of any sub-diagonal entry in that entry. In our code, we carry out the command `[L,U,Pi] = lu(A',thresh)`. We would like the rows of A^T that correspond to large entries of H to be avoid being chosen by the LU threshold method, so we would like to scale these rows so that their entries are small relative to those corresponding to small diagonal entries in H. There are two obvious ways to do this scaling:

- set $\overline{A} = AD^{-1}$, find Π by carrying out an LU factorization with threshold pivoting on $\overline{A}^T$;
- set $\overline{A} = AD^{-\frac{1}{2}}$, find Π by carrying out an LU factorization with threshold pivoting on $\overline{A}^T$;

where $D = \mathtt{diag}(H)$.

The first idea comes from simply scaling the columns of A by a value inversely proportional to the corresponding diagonal entry in H. The second idea was obtained by symmetrically scaling the original saddle-point system so that resulting (1,1)-block has unit diagonal entries:

$$\underbrace{\begin{bmatrix} D^{-\frac{1}{2}}\widehat{H}D^{-\frac{1}{2}} & D^{-\frac{1}{2}}\widehat{A}^T \\ \widehat{A}D^{-\frac{1}{2}} & 0 \end{bmatrix}}_{\mathcal{H}_D} \begin{bmatrix} \widehat{x} \\ y \end{bmatrix} = \begin{bmatrix} D^{-\frac{1}{2}}\Pi^T c \\ d \end{bmatrix},$$

and $x = D^{-\frac{1}{2}}\Pi\widehat{x}$. Let

$$P_D = \begin{bmatrix} D^{-\frac{1}{2}}\widehat{G}D^{-\frac{1}{2}} & D^{-\frac{1}{2}}\widehat{A}^T \\ \widehat{A}D^{-\frac{1}{2}} & 0 \end{bmatrix}.$$

The eigenvalues of the preconditioned system $P_D^{-1}\mathcal{H}_D$ will be clustered around 1 if the entries in the last $n - m$ columns of $\widehat{A}D^{-\frac{1}{2}}$ are all small.

Let us compare the different methods over a subset of QP problems from the CUTEr test set [5]. We shall record:

- k, the number of interior point iterations carried out;
- Σ_1, the total number of PPCG iterations carried out to find the predictor steps;
- Σ_2, the total number of PPCG iterations carried out to find the corrector steps.

Table 1: Comparison of interior point and PPCG iterations for the `LUH`, `LUD` and `LUDsq` permutations.

			`LUH`			`LUD`			`LUDsq`		
Name	m	n	k	Σ_1	Σ_2	k	Σ_1	Σ_2	k	Σ_1	Σ_2
AUG2DQP	1600	3280	436	2292	553	20	1397	1524	21	1427	1536
AUG2DCQP	1600	3280	87	3591	5001	22	1508	1651	20	1407	1535
AUG3DQP	1000	3873	12	847	882	11	419	425	28	1311	1137
AUG3DCQP	1000	3873	13	896	946	11	701	704	20	1336	1206
CONT-050	2401	2597	6	20	19	6	20	19	6	20	19
CVXQP1_M	500	1000	9	792	811	9	353	357	9	294	298
CVXQP2_M	250	1000	11	222	231	11	378	382	11	253	258
CVXQP3_M	750	1000	10	766	774	10	247	252	10	214	214
DUALC1	215	223	15	53	62	11	39	41	11	37	37
DUALC2	229	235	31	174	181	7	26	25	7	26	25
DUALC5	278	285	6	15	15	6	19	19	6	20	20
DUALC8	503	510	17	144	165	8	35	35	8	35	35
KSIP	1001	1021	10	32	33	9	29	30	9	29	30
MOSARQP1	700	3200	10	738	743	12	232	235	12	454	458
PRIMAL1	85	410	10	369	367	10	359	358	10	337	331
PRIMAL2	96	745	12	544	549	12	625	620	12	549	549
PRIMAL3	111	856	9	534	532	9	600	602	9	563	562
PRIMAL4	75	1564	7	565	561	7	424	426	7	402	403
PRIMALC1	9	239	31	126	134	27	102	113	27	97	106
PRIMALC2	7	238	40	64	87	21	56	59	21	57	57
STCQP2	2052	4097	14	14	14	14	14	14	14	14	14

We note that each iteration of the PPCG method will be comparable in CPU time and memory usage for each of the different permutation methods. The method used in my thesis to find the permutation will be denoted by `LUH`, and the ideas presented in this note will be denoted by `LUD` and `LUDsq` respectively.

The results are given in Table 1. We observe that the methods `LUD` and `LUDsq` are generally using a significantly reduced number of PPCG iterations to solve the QP problems compared with the `LUH` method. To help us further analyze the results the total number of PPCG iterations used are compared using a performance profile, Figure 1. We observe that, as expected, the methods `LUD` and `LUDsq` are generally performing significantly better than the `LUH` method. The `LUD` and `LUDsq` methods are performing similarly for around 75% of the problems, but `LUD` is generally performing better for the remaining problems. This is because `LUD` method is more likely to "detect" columns in A corresponding to large diagonal entries in H early on in the interior point method.

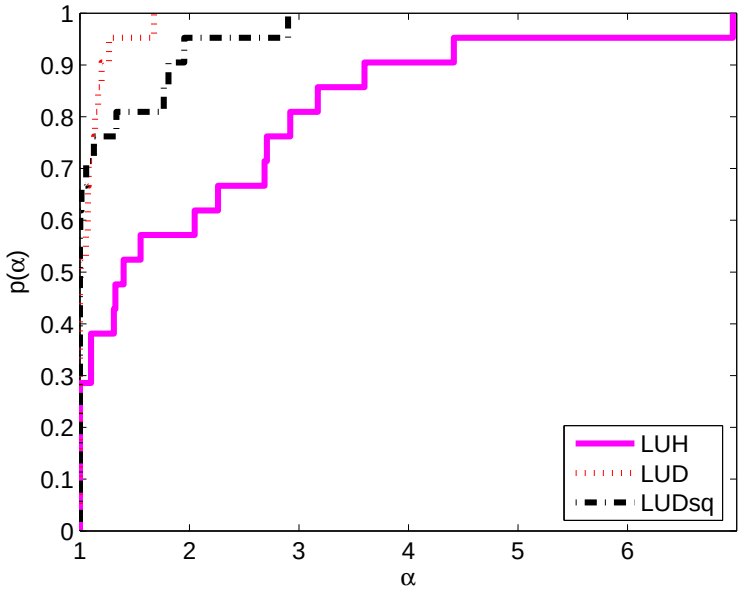

Figure 1: Performance profile comparing the total number of PPCG iterations.

4 Conclusions

We have shown how the choice of permutation used to obtain a non-singular A_1 can have a dramatic effect on the number of PPCG iterations required during a run of an interior point method for solving quadratic programming problems and that, for certain choices of preconditioner, taking the entries of the H into account when forming A_1 can be advantageous.

The currently proposed methods will not be suitable for very large problems so the next stage of this work will be to develop a similar method which is also suitable for large values of n and m.

References

[1] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta Numerica, 14 (2005), pp. 1–137.

[2] L. Bergamaschi, J. Gondzio, and G. Zilli, *Preconditioning indefinite systems in interior point methods for optimization*, Comput. Optim. Appl., 28 (2004), pp. 149–171.

[3] H. S. Dollar, *Iterative Linear Algebra for Constrained Optimization*, Doctor of Philosophy, Oxford University, 2005.

[4] H. S. Dollar, N. I. M. Gould, W. H. A. Schilders, and A. J. Wathen, *Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 170–189.

[5] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited*, ACM Transactions on Mathematical Software, 29 (2003).

[6] C. Keller, N. I. M. Gould, and A. J. Wathen, *Constraint preconditioning for indefinite linear systems*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1300–1317.