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

where $A_1 \in \mathrm{R}$

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

$$Z = \begin{pmatrix} -A_1^{-1}A_2 \\ I \end{pmatrix}.$$

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 P moves all the large diagonal entries of H into 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 H_{11}; the eigenvalues of (6) would have magnitude $\mathcal{O}(\frac{1}{t})$ and the preconditioned system will be very ill-conditioned.

3 Permutation methods

Dollar [3] firstly found a permutation P_1 such that the diagonal entries of $P_1^T H P_1$ are sorted in non-decreasing order. A permutation P_2 was then obtained by carrying out an LU factorization of $P^T A^T$ 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 H_{11}; where $P = P_1 P_2$: Unfortunately P_2 was frequently far from being the identity so many of the large entries ended up in H_{11}:

In an attempt to improve on this method she also looked at finding a permutation $\tilde{P}_1$ such that the diagonal entries of $\tilde{P}_1^T H \tilde{P}$ are sorted in non-increasing order. As before, a permutation $\tilde{P}_2$ is obtained by carrying out an LU factorization of $\tilde{P}^T A^T$ with threshold row pivoting. The resulting permutations $P = P_1 P_2$ and $\tilde{P}$

the two, the numerical tests carried out in the thesis [3] imply that there might be another method to ¯nd a permutation ¦ which, for convenience, can be coded using standard Matlab® functions and is more e®ective 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 $\overline{A} = AD^{-1}$; ¯nd ¦ by carrying out an LU factorization with threshold pivoting on $\overline{A}^T$;
- set $\overline{A} = AD^{-\frac{1}{2}}$; ¯nd ¦ by carrying out an LU factorization with threshold pivoting on $\overline{A}^T$;

where D = `diag`(H):

The ¯rst 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

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

			LUH			LUD			LUDsq		
Name	m	n	k	$\S_1$	$\S_2$	k	$\S_1$	$\S_2$	k	$\S_1$	$\S_2$
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.

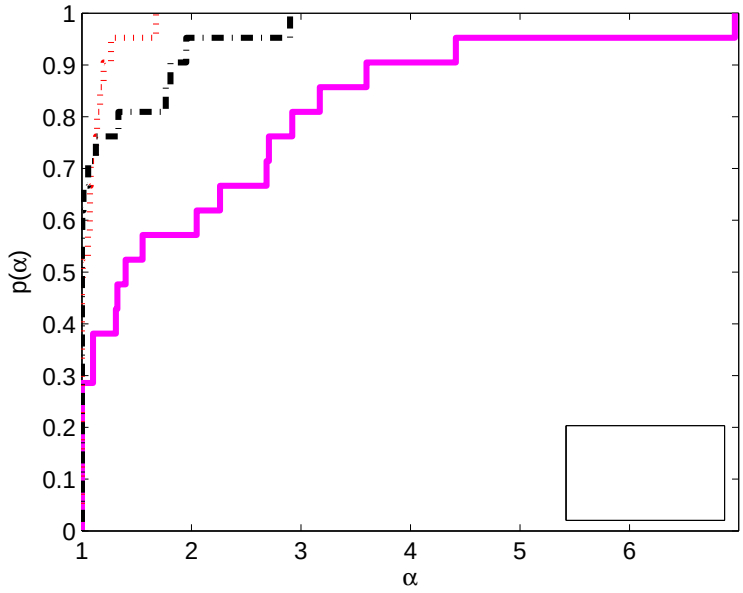

Figure 1: Performance pro¯le comparing the total number of PPCG iterations.

4 Conclusions

We have shown how the choice of permutation used to obtain a non-singular A_1 can have a dramatic e®ect 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 inde¯nite 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.