A Sparse QS-Decomposition for Large Sparse Linear System of Equations

Wujian Peng[1] and Biswa N. Datta[2]

[1] Department of Math, Zhaoqing University, Zhaoqing, China,
douglas_peng@yahoo.com
[2] Department of Math, Northern Illinois University, Dekalb, IL, USA,
dattab@math.niu.edu

Summary. A direct solver for large scale sparse linear system of equations is presented in this paper. As a direct solver, this method is among the most efficient direct solvers available so far with flop count as $O(n\,logn)$ in one-dimensional situations and $O(n^{3/2})$ in second dimensional situation. This method has advantages over the existing fast solvers in which it can be used to handle more general situations, both well-conditioned or ill-conditioned systems; more importantly, it is a very stable solver and a naturally parallel procedure! Numerical experiments are presented to demonstrate the efficiency and stability of this algorithm.

1 Introduction

One of the core tasks in mathematical computations is to solve algebraic linear system of equations, which often arise from solving physical and engineering problems modeled in PDE and discretized by FEM or FDM. Frequently one may encounter systems with up to hundreds of thousands of unknowns and coefficient matrix is usually very sparse, i.e., most of the entries in the coefficient matrix are zeros.

Currently direct methods for solving large scale linear systems are variations of Gaussian elimination method (GE), such as TDMA, the multifrontal methods [1, 4], Pardiso [2] and superLU [5],etc. It is well-known that GE method is not stable due to the accumulated rounding errors, even if partial pivoting is applied; the excessive flops needed in GE (which is a part reason for the instability) is also an big issue. Though parallel can be partly implemented in reordering of unknowns [3, 6], a forward and a backward substituting procedure are always needed.

In this paper we are to discuss a complete parallel efficient algorithm for solving linear system of equations. This algorithm is based on the recognizing of certain special sequence of vectors, which is then orthogonalized by a special algorithm called Layered Group Orthogonalization (LGO). The new direct solver is based on LGO and is applied to solve linear systems with banded matrices arising from some one-dimensional problems. Numerical examples show some astonishing stability behavior of this new method. More applications of the idea presented here are to be followed by later papers.

2 A Quasi-Orthogonal Vector Sequence

We start from recognizing a special sequence of vectors, which are closely related to banded matrices.

Definition 1. *A sequence of vectors $v_1, v_2, \cdots, v_n$ is called quasi-orthogonal if there exists an integer m such that after grouping the above sequence into groups of m vectors*

$$\{v_1, \cdots v_m\}, \{v_{m+1} \cdots, v_{2m}\}, \cdots, \{v_{lm+1}, \cdots, v_n\}$$

there holds

$$v_i^T v_j = 0, \forall v_i \in G_{i'}, v_j \in G_{j'} \qquad (|i' - j'| > 1) \tag{1}$$

where G_i is the ith group containing vectors $\{v_{(i-1)m+1} \cdots, v_{im}\}$. Note the last group does not necessarily have m vectors. Furthermore, since any integer greater than m can be used to group the vector sequence while (1) is satisfied, we call this sequence as k-orthogonal sequence with k as the smallest integer m available.

Obviously a regular orthogonal sequence can be regarded as a quasi-orthogonal sequence, but a k-orthogonal sequence ($k > 0$) is not orthogonal. We are to present an efficient algorithm to orthogonalize a k-orthogonal sequence a moment later, but first let's give some examples of quasi-orthogonal sequences.

Example 1. Let $A \in \mathcal{R}$ be a bidiagonal matrix, $A = (a_1, a_2, \cdots a_n)$ be a partition with $a_i (i = 1, \cdots n)$ as its column vectors, then $a_1, a_2, \cdots a_n$ form a 1-orthogonal sequence. If one partitions A into a sequence of row vectors, one also gets a 1-orthogonal sequence.

Similarly one can see that the row/column vectors of a tridiagonal matrix form a 2-orthogonal sequence. As a matter of fact, given any banded matrix A with at most $k+1$ nonzero entries at each row/column, the associated row/column vectors always form a k-orthogonal sequence.

Note here we define k-sequence from vectors with only a few nonzero entries. Does there exist indeed vector sequence with all entries as nonzero numbers to form a k-orthogonal sequence? The following remark provides a positive answer to this question.

Remark 1. Let $Q \in \mathcal{R}^{n \times n}$ be an orthogonal matrix, $v_1, v_2, \cdots, v_n$ be a k-orthogonal sequence with each $v_i \in \mathcal{R}^n$, then sequence $Qv_1, Qv_2, \cdots, Qv_n$ also form a k-orthogonal sequence.

Outline of the proof Note that we have

$$(Qv_i)^T Qv_j = v_i^T Q^T Q v_j = v_i^T v_j = 0, \text{ for } |i' - j'| > 1,$$

where $v_i \in G_{i'}, v_j \in G_{j'}$, the conclusion thus comes directly from the above observation.

It is easy to see that any vector combinations inside each group will not change the k-orthogonal property of the sequence. As a direct consequence, the following statement is true and will be used in later section.

Remark 2. Let $G_1, G_2, \cdots G_l$ be a sequence of groups associated with a k-orthogonal sequence. If we orthogonalize vectors in each group, the newly formed groups of vector, still denoted by $G_1, G_2, \cdots G_l$, keep the k-orthogonal property.

3 Layered Group Orthogonalization

As we mentioned previously, the k-orthogonal sequence is not a regular orthogonal sequence, thus a process is needed to orthogonalize this sequence. However classical Gram-Schmidt process or Hessenberg is not suitable for this purpose since the sparsity of the vector (matrix) structure will be destroyed. Although Hessenberg QR factorization can be used to orthogonalize the sequence, its parallel implementations is not an easy task [7] since in essence QR factorization is a sequential algorithm

In this section we are to provide a new type of orthogonalization process especially designed for this type of quasi-orthogonal sequence, which is called Layered Group Orthogonalization process (LGO). The detail comes as follows.

3.1 Algorithm (LGO)

Let $v_1, v_2, \cdots, v_n$ be a k-orthogonal vector sequence and is grouped into groups $G_1, G_2, \cdots G_l$. Assuming each group has $m(\geq k)$ vectors.

Step 1: Orthogonalize all odd groups (these can be done independently), i.e., $G_1, G_3, \cdots G_l$ are orthogonalized at this step (assuming l is odd).

Step 2: If there are no even groups left, stop. Otherwise update each even group by making it orthogonal to its neighboring odd groups.

$$\bar{G}_{2i} = \bar{G}_{2i} - \bar{G}_{2i+1}\bar{G}_{2i+1}^T\bar{G}_{2i} - \bar{G}_{2i-1}\bar{G}_{2i-1}^T\bar{G}_{2i}$$

where $\bar{G}_j$ represents the matrix formed by vectors in vector group G_j.

Step 3: Take out all odd groups and renumber the groups left in the same order: $G_2, G_4, \cdots G_{2s} \longrightarrow G_1, G_2, \cdots, G_s$

Step 4: Go back to Step 1.

To justify the above algorithm, we need first the following conclusions.

Proposition 1 *The vector sequence formed by vectors in all odd groups after step 1 are orthogonal.*

Proof By the definition of k-orthogonality, vectors from different odd groups are orthogonal already. By Remark 2 and the fact that vectors in each odd group are orthogonalized in step 1, vectors from the same group are orthogonal to each other. This completes the proof.

Proposition 2 *The vectors in every even group are orthogonal to vectors in any odd groups after step 2. i.e., given any vector $v_i \in G_{2i'+1}$, $v_j \in G_{2j'}$, there holds $v_i^T v_j = 0$.*

The proof of this proposition is omitted .

Proposition 1 and 2 actually indicate this fact: "half" vectors in the original k-orthogonal sequence are orthogonalized in the first run of the loop in the above algorithm. To make the second run keeps similar orthogonalization function, we need the conclusions that follows immediately.

Proposition 3 *Vector sequence formed by vectors from even groups after second step in the above algorithm still form a k-orthogonal sequence.*

Proof It is omitted here.

3.2 Matrix representation of LGO

For the sake of simplicity, we assume k-orthogonal sequence $v_1, v_2, \cdots, v_n$ is grouped into $l+1$ groups $G_0, G_1, G_2, \cdots, G_l$ with each containing k vectors, we denote the first step operation in the algorithm by matrix $S_1^{(i)}$ and $S_2^{(i)}$ for the second step in each run of the loop ($i = 1, 2, \ldots$). Then one can see that

$$S_1^{(1)} = \begin{pmatrix} R_1^{-1} & & & & \\ & I_k & & & \\ & & R_3^{-1} & & \\ & & & I_k & \\ & & & & \ddots \end{pmatrix} \quad \text{and } S_2^{(1)} = \begin{pmatrix} I_k & -\hat{G}_1^T G_2 & & & \\ 0 & I_k & 0 & & \\ & -\hat{G}_3^T G_2 & I_k & -\hat{G}_3^T G_4 & \\ & & 0 & I_k & 0 \\ & & & & \ddots \end{pmatrix}$$

assuming $l+1$ is odd. Thus one can see that the first step in each run of the loop always corresponding to a block diagonal matrix with only a small portion of blocks are non-identity blocks, while the second step matrix has a small portion of columns having at most three non-zero blocks.

The whole process, denoted by matrix S^{-1}, has the following form: $S^{-1} = S_1^{(1)} S_2^{(1)} \cdots S_1^{(r)} S_2^{(r)}$, where $r = [\log(n/k)]$. Let Q be the resulted orthogonal matrix by LGO process, then one has $A = QS$, where A is the matrix formed by the k-orthogonal sequence as its column vectors. Note that both Q and S are sparse matrices with $O(k^2 n \log(n/k))$ nonzero entries. Actually the following graphs show their sparsity pattern.

4 LGO Solver and Numerical Experiments

An immediate application of the LGO factorization is of course to solve linear system $Ax = b$ with A as a banded matrix. Formerly one would have by multiplying both sides the inverse of matrix S,

$$S^{-1}Ax = S^{-1}b.$$

Note that $Q = S^{-1}A$ is an orthogonal matrix, thus the solution x can be written as

$$x = (S^{-1}A)^T S^{-1}b = Q^T S^{-1}b.$$

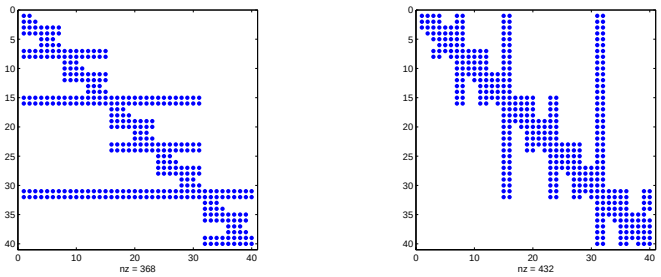

Fig. 1. Distribution of nonzeros of sparse matrix Q and S.

In actual calculations, the inverse of matrix S is not explicitly formed, and as we noted previously, the operation $S^{-1}A$ and $S^{-1}b$ only cost $O(k^2 n \log(n/k))$ flops.

Next we present results of some numerical experiments. We take the simplest banded matrix–tridiagonal matrix. As we know currently the most frequently used method is the so-called TDMA. Our comparison of LGO method and TDMA in case A is diagonally dominant shows that these two methods reach almost the same precision and the result are thus not listed here.

In case of A not diagonally dominant, we tried to use TDMA, UMFPACK and LGO to solve a system with A slightly away from diagonally dominant as follows:

$$A = tridiag\{-1.05, 2, -1\}$$

and the condition number of this matrix A grows from hundreds when n is less than 100 to $O(10^{13})$ when n is about 1,000.

In Fig. 2 one can see that both TDMA and UMFPACK work well in case of system with 1,000 unknowns, however when the size of system is greater than 1,500, the relative error becomes unacceptable. In order to test the stability of LGO, we

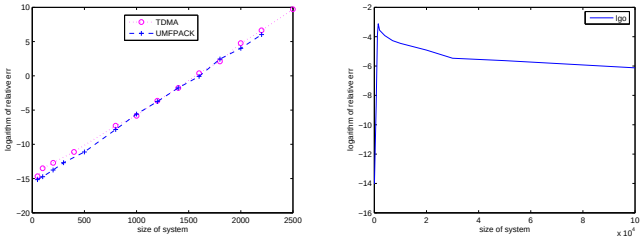

Fig. 2. Comparison of relative error by using TDMA, UMFPACK and LGO.

construct an extremely ill-conditioned linear system with A as a tridiagonal matrix with diagonal entries as 2, upper diagonal entries as -1 and lower diagonal entries run from 1 to $n-1$ with n as the size of the matrix. The following table is the calculated condition numbers of A when n varies.

n	10	15	20	25
cond (A)	9.3066e+004	5.0624e+008	4.7161e+013	3.9742e+017

Both TDMA and UMFPACK fail when $n = 30$. But by using LGO solver we successfully obtained pretty good solutions for n up to $100,000$ with the exact solution as $f(x) = x(1-x)e^{x+6}$ (see Fig. 3).

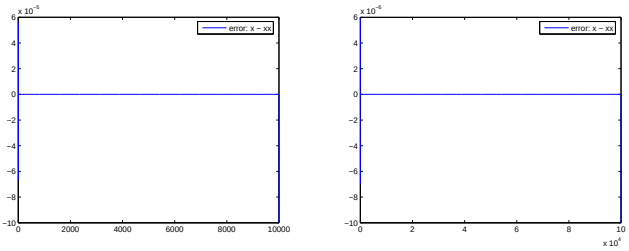

Fig. 3. Error between the exact solution and calculated solution using LGO.

5 A Nested Direct Domain Decomposition Idea

In this section we are going to briefly talk about another important application of the aforementioned direct solver, which is actually a generalization of LGO.

Based on our discovery of the quasi-orthogonal sequence introduced in previous section, we find that in FEM or FDM, each mesh node corresponding to a row in resulted coefficient matrix can be naturally grouped into different groups by its geometric locations; and if two groups are separated by at least a few grid lines (Fig. 4), then row vectors corresponding to these two groups can be orthogonalized independently. A simple implementation is described in the following.

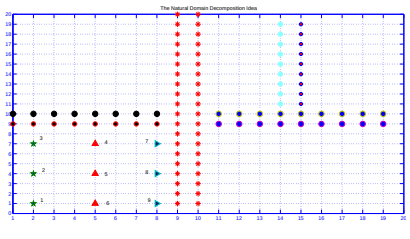

Fig. 4. Partitioning of direct domain decomposition.

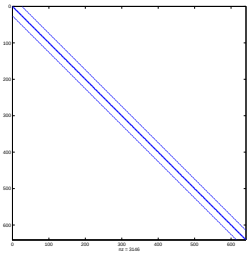

Fig. 5. Zero pattern of stiffness matrix.

A generalized LGO process is used to handle a second dimensional problem on $[0,1]^2$. Here we assume the domain is discretized with a rectangular mesh. In each layer of the LGO process we use small rectangular areas as the subdomains and the

subdomains in the upper layer are formed by grouping the neighboring rectangular subdomains in previous layer which share a common vertex. By using lexicography order of nodes the resulted coefficient matrix A is a banded matrix with zeros inside the band, as shown in Fig. 6.

The generalized LGO factors of A in this case have however some different zero patterns, as we can see from the following figures (Figs. 6, 7) that the counts for nonzeros are of order $O(n^{1.5})$, which can also be verified by estimating each step of the generalized LGO process.

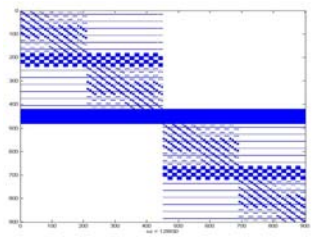

Fig. 6. Zero pattern of factor Q.

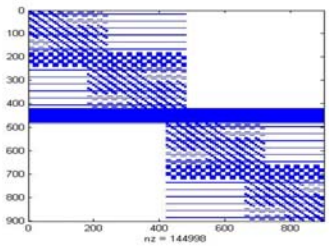

Fig. 7. Zero pattern of factor S.

Comparison between UMFPACK and LGO have been made on both well-conditioned and ill-conditioned systems from some simple two dimensional problems. Numerical experiments show that for standard Poisson problem defined on rectangular domain $[0,1]^2$, both methods work well (Fig. 8). However, for an artificially constructed discrete operator which leads to the following Toeplitz-like matrix $A = (a_{ij})_{n\times n}$ (very ill-conditioned) where

$$a_{ij} = \begin{cases} 4, & (i = j) \\ -6, & (i = j-1 \text{ and } (i \bmod l) \neq 1) \\ -2, & (i = j+1 \text{ and } (i \bmod l) \neq 0) \\ -1, & (|i-j| = l) \\ 0, & (\text{ otherwise}) \end{cases}$$

and l is the positive square root of system size n, UMFPACK fails when the size of system n is greater than a few thousands while LGO solver still shows satisfying relative error when n is more than $20,000$ (Fig. 9). More meaningful tests and applications of LGO are currently under the way and the results will be presented in later papers.

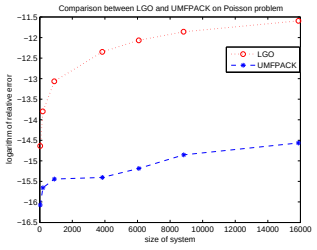

Fig. 8. Comparison on Poisson problem.

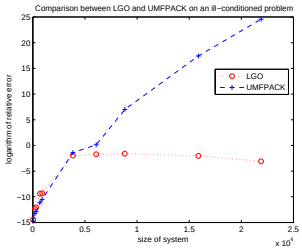

Fig. 9. Comparison on ill-conditioned problem.

Acknowledgement We would like to thank Professor Qun Lin for his great support while the first author was visiting LSEC. We also benefit from our discussion with Dr. Qiya Hu , Dr. Linbo Zhang and Dr. Zhongzhi Bai in LSEC as well as Dr. Jun Zou and Dr. Shuhua Zhang. We are encouraged to finish this part of research and gave a report at the conference.

References

1. I.S. Duff. A survey of sparse matrix research. In *Proc. IEEE*, 65(4): 500–535, 1977.
2. K. Gartner, W. Fichtner, and A. Stricker. Pardiso: A high-performance serial and parallel sparse linear solver in semiconductor device simulation. *J. Future Generation Comp. Syst.*, 18:69–78, 2001.
3. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10: 345–363, 1973.
4. A.M. Erisman, I.S. Duff and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, NY, 1986.
5. J. Gilbert, J. Demmel and X. Li. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 20(4):915–952, 1999.
6. X.S. Li and J.W. Demmel. Making sparse Gaussian elimination scalable by static pivoting. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, Orlando, FL, 1998.
7. J. Touriño, R. Doallo and E.L. Zapata. Sparse Householder qr factorization on a mesh. In *Proceedings of the Fourth Euromicro Workshop on Parallel and Distributed Processing*, Braga, 1998.