LAB 3-FACTORIZATION OF SPARSE MATRICES

BY

SACHIN SRINIVASA SHETTY

Contents

List of Figures

List of Tables

1 Introduction

Sparse matrix is a matrix in which most of the elements are zero.The sparse matrix factorisation can be performed by utilising the sparisity of the matrix.This method is compared with the dense matrix factorisation.The matrices we are using are symmetric and positive definite matrix.The choleksy factorisation is implemented as it more efficient than the LU decomposition for this case.The sparse matrix are ordered using different algorithms and the results are studied.

In the Part 1 of this study,we studied the 2 reordering schemes namely,Cuthill McKee and Reverse Cuthill McKee.The results are compared with the implementation on the original scheme.In Part 2,We used an efficient library: SuperLU to factorise the matrix.The various option of the SuperLU was also studied.The results were compared with the dense matrix factorisation.

2 Part 1:Sparse Cholesky and the importance of numbering

In this part, the Cholesky factorization is implemented on the sparse matrices.The matrices can be factorized as $A = LL^T$.The Cholesky factorisation is implemented using three different schemes.They are:

- Original System (without reordering)
- Cuthill McKee Algorithm
- Reverse Cuthill McKee Algorithm

The Cuthill McKee algorithm is used to permute the sparse matrix in a way to reduce the bandwidth. It is a variant of the Breadth-first search (BFS) algorithm.BFS is a traversing algorithm where you start traversing from a selected node (starting node) and traverse the graph layer wise thus exploring the neighbor nodes (nodes that are directly connected to the source node). The Reverse Cuthill McKee algorithm is the same algorithm as Cuthill McKee but the resulting index numbers are reversed. The Cuthill McKee algorithm is implemented in the file *graph.cc*.

The effect of the Cuthill McKee and Reverse Cuthill McKee is studied on the *square*4 matrix.The shape of the input matrix is shown below:

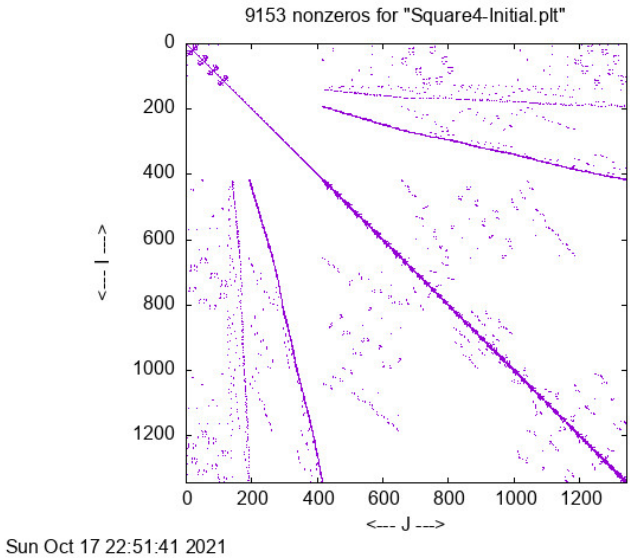


Figure 1: Shape of the matrix

The below figure shows the shape of the matrix after the Cholesky factorisation.It can be seen that the number of non zero terms have increased when compared to the original matrix.

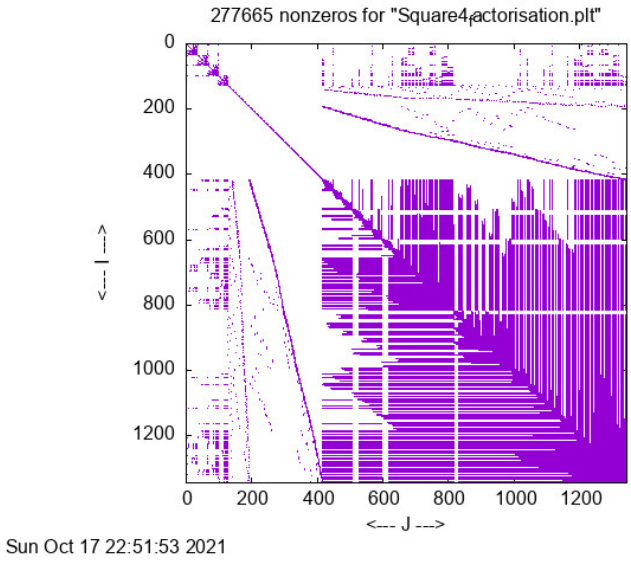
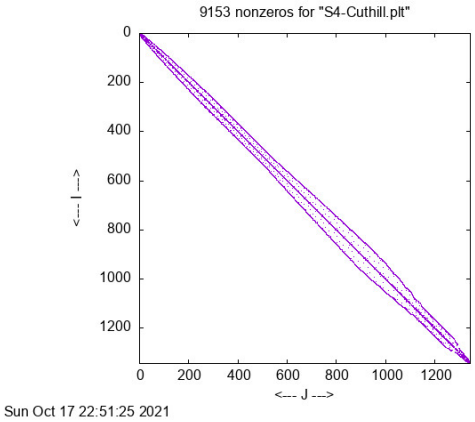
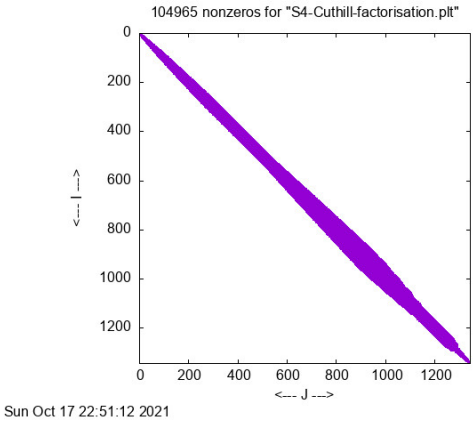


Figure 2: Shape of the factorised matrix without reordering

The Cuthill McKee algorithm is then implemented to the matrix. The shape of the matrix and the factorized matrix using the Cuthill Mckee is shown below. It can be observed that the bandwidth of the original matrix is reduced. The terms of the matrix are confined to the bandwidth. And, after the factorization, it can be seen that non-zero elements are also confined to the bandwidth. The non-zero elements after factorization are also reduced and the storage size of the matrix is also reduced by the implementation of Cuthill McKee.



(a) Shape using Cuthill McKee

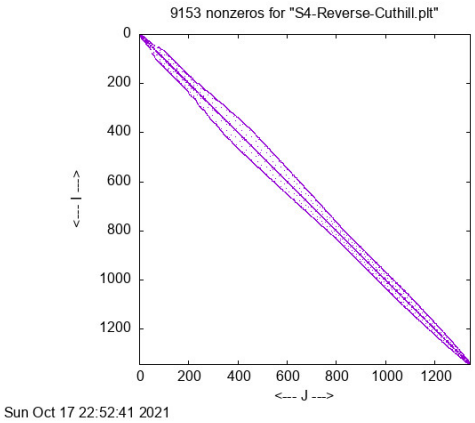


(b) Shape after factorisation using Cuthill McKee

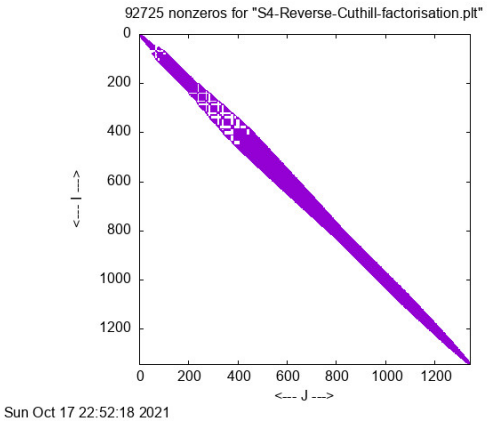
Figure 3: Cuthill McKee Algorithm

The Reverse Cuthill McKee algorithm is then implemented in the matrix. The shape of the matrix and the factorized matrix using the Reverse Cuthill Mckee is shown below. The Reverse Cuthill McKee is the same as Cuthill McKee but the order is reversed. By the implementation of the Reverse Cuthill McKee, the number of the non-zero elements is reduced when compared with the Cuthill McKee.

These schemes were then implemented to the matrix of different sizes. The number of non-zero



(a) Shape using Reverse Cuthill McKee



(b) Shape after factorisation using Reverse Cuthill McKee

Figure 4: Reverse Cuthill McKee Algorithm

elements after factorization for all the schemes is shown in the table below. It can be observed that the number of non-zeros are reduced by the implementation of Cuthill McKee and Reverse Cuthill McKee. The time taken by the different schemes for the factorization is shown in the table and the plot below. The time taken by the Cuthill McKee and Reverse Cuthill McKee is much less than the time taken to factorize the matrix in the original order. The Reverse Cuthill McKee is faster than the Cuthill McKee scheme.

Input Matrix		Number of Non-zero after Factorisation (NNZ)		
Matrix	**NNZ**	**Original Order**	**Cuthill McKee order**	**Reverse Cuthill McKee oder**
Square	165	217	203	165
Square2	609	1718	1107	940
Square3	2337	15573	7421	6477
Square4	9153	139505	53155	47035

Table 1: Comparison of Number of Non-zeros for different schemes

Input Matrix		Time taken for Factorisation		
Matrix	**NNZ**	**Original Order**	**Cuthill McKee order**	**Reverse Cuthill McKee oder**
Square	165	0.000478441	0.000327122	0.000305871
Square2	609	0.00786559	0.00207625	0.00171174
Square3	2337	0.49517	0.0744588	0.018773
Square4	9153	23.4122	0.812674	0.663658

Table 2: Comparison of Time taken for different schemes

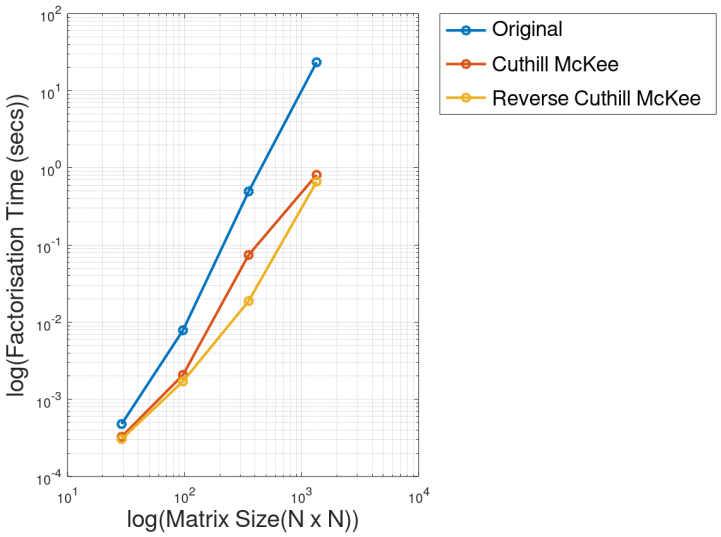
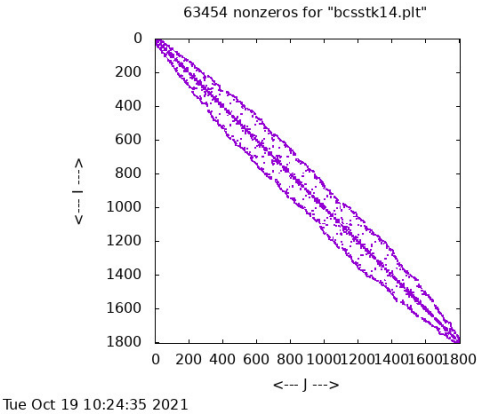
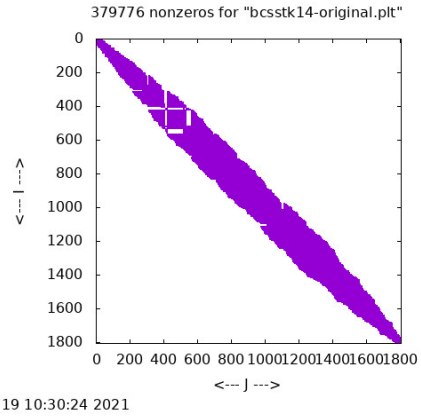


Figure 5: Loglog plot of time(secs) vs matrix size(NxN) for various schemes

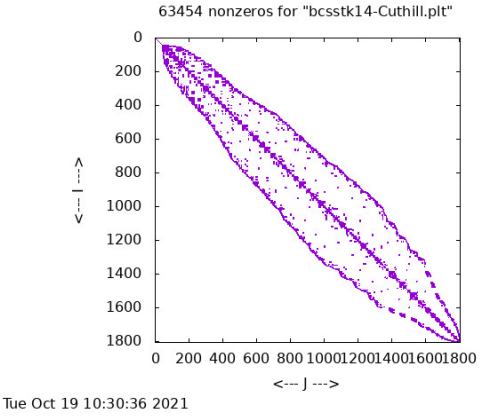
These various schemes were implemented on the large symmetric band matrix *bccstk*14.*mtx*.It was observed that the computational time increases on reordering this matrix.Also,the bandwidth of the matrix is increased.The following images shows the shape of the *bcsstk*14 matrix after the reordering and the factorisation.



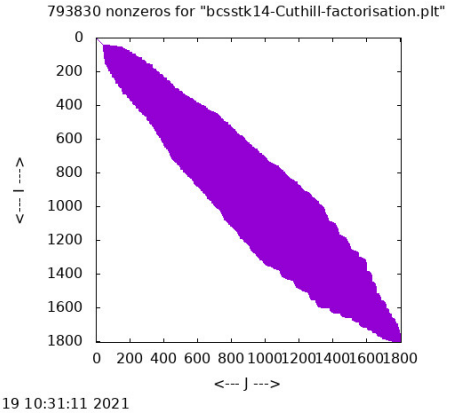
(a) Shape of Input Matrix



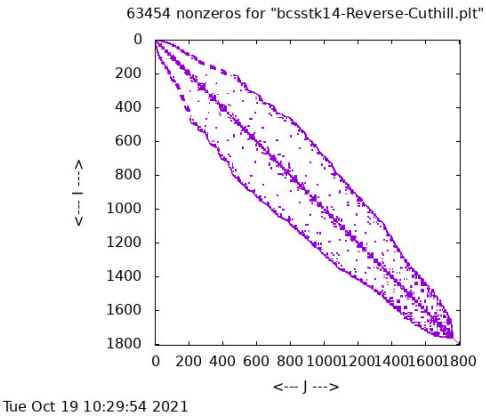
(b) Shape after factorisation without reordering schemes



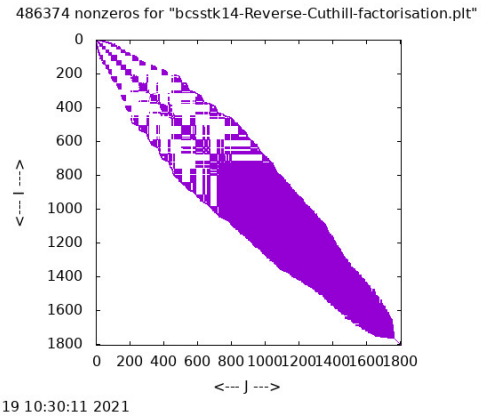
(a) Shape using Cuthill McKee



(b) Shape after factorisation using Cuthill McKee



(a) Shape using Reverse Cuthill McKee



(b) Shape after factorisation using Reverse Cuthill McKee

Figure 6: Reodering Schemes on the Large Band Symmetric matrix *bcsstk*14

3 Part 2:Using an Efficient Library : SuperLU

SuperLU is a general-purpose library for the direct solution of large, sparse, non-symmetric systems of linear equations. We are implementing this library to solve the large, sparse, and symmetric system. The SuperLU is implemented on different matrices to study its performance. The below table shows the recorded value. It can be observed from the table that the factorization time is dependent on the matrix which in turn depends on the number of non-zero terms in the matrix. The solving time was observed to be very close to zero..

Input Matrix		**SuperLU**				
Size (nxn)	**NNZ**	**NNZ (L+U)**	**Fill ratio**	**Fact. time**	**Factor Flops**	**Solve time**
1224	56126	138370	2.5	0.02	7.33E+06	0
1806	63454	525338	8.3	0.1	8.08E+07	0
3948	117816	1820709	15.5	0.49	4.99E+08	0.01
4884	290378	1962004	6.8	0.43	4.54E+08	0

Table 3: Statistics of SuperLU implemented on different matrices

Many options in the superLU library can be used to modulate our solution. The matrix *bcsstk*15.*mtx* is used to study the option ColPerm and option DiagPivotThresh.The options are varied and the respective results are recorded. The recorded values are shown in the tables below. The option ColPerm is used to implement different renumbering strategies used to fill in. The 4 strategies were implemented. The COLUMAND was observed to have the least fill ratio and factorization time. The option DiagPivotThresh is used to control the pivoting, where 0.0 refers to almost no pivoting and 1.0 referred to a lot of pivoting. It can be observed that as the pivot control is increased the fill ratio also increases and the time taken by the factorization is also increased.

ColPerm	**L+U**	**Fill ratio**	**Fact. time**	**Solve time**
NATURAL	2385733	20.2	0.91	0.01
MMD-ATA	2121264	18	0.73	0
MMD-AT-PLUS	9365613	79.5	17.02	0.03
COLUMAND	1820709	15.5	0.53	0

Table 4: SuperLU statistics for the option ColPerm on matrix *bcsstk*15.*mtx*

DiagPivotThresh	**L+U**	**Fill ratio**	**Fact. time**	**Solve time**
0.0	1492404	12.7	0.37	0
0.2	1791038	15.2	0.48	0.01
0.4	1803790	15.3	0.56	0.01
0.6	1806414	15.3	0.51	0.01
0.8	1812870	15.4	0.44	0.01
1.0	1820709	15.5	0.54	0

Table 5: SuperLU statistics for the option DiagPivotThresh on matrix *bcsstk*15.*mtx*

The factorization time obtained by using SuperLU is then compared with Dense matrix factorization. The values recorded are shown in the table below. It can be seen that the factorization time taken by the dense matrix is much more than the SuperLU which uses the sparse property of the matrix.

Size (nxn))	**Dense matrix factorisation**	**Using SuperLU (Sparse)**
1224	3.41531	0.02
1806	27.3827	0.1
3948	630.724	0.49
4884	1227.74	0.43

Table 6: Comparison of Dense matrix factorisation and Using SuperLU

4 Conclusion

In conclusion, the reorder schemes help to implement the sparse factorization in an efficient way for symmetric positive definite matrices. The number of non-zero terms after factorization decreases in the implementation of Cuthill McKee and Reverse Cuthill Mckee. The computational time taken by these reordering schemes was less when compared with factorization without reordering schemes. It was also observed that when these reordering schemes were implemented on a large symmetric band matrix, the computational time is observed to increase.

The usage of an efficient library: SuperLU reduces the factorization time of the matrices by a significant amount. The options *ColPerm* and *DiagPivotThresh* were also studied. The effect of different parameters for these options was also studied. The factorization time was compared with dense matrix factorization time, a significant decrease in time can be observed. The usage of sparsity property in sparse matrix factorization helps to reduce the computational time, the number of non-zeros after factorization and inturn the storage space.