LAB 2-LU AND CHOLESKY FACTORISATION,DENSE STORAGE AND SYMMETRIC BAND

BY

SACHIN SRINIVASA SHETTY

Contents

1 Introduction **2**

2 Part1:Dense Matrix factorization **2**
2.1 Result and Observations 2
2.2 Error in Factorization 3

3 Part 2:Symmetric Positive Definite Band Matrix **4**
3.1 Result and Observation 5
3.2 Error in Factorisation 6

4 Conclusion **7**

List of Figures

1 Plot for Time taken for different implementations of LU factorization 3
2 Error in Factorization for different implementations of LU factorization 4
3 Code for Computation of BandWidthUp 4
4 Code for Computation of BandWidthDown 5
5 Code for Computation of the Operator()(int i,int j) 5
6 Plot for Computational time of LU and Cholesky Factorization 6
7 Error in Factorization 6

List of Tables

1 Computational Time for different implementation of LU factorization 2
2 Computational Time between LU factorization and Cholesky 5

1 Introduction

This lab aims to analyze the computational time of different implementations of LU factorization and also to analyze the efficient way to store a band matrix to reduce the computational time. In the first part of this lab, The three implementations of LU factorization are analyzed and are compared with the LAPACK implementation. In the second part of this lab, The best way to factorize band matrices and also the efficient way to store band matrices for factorization are studied. The LU factorization is performed in dense form and the Cholesky factorization is performed in compressed symmetric band storage. This implementation is performed for various matrix sizes and is analyzed. The implementations are carried out in C++ and results are shown in the table and are compared with graphs.

2 Part1:Dense Matrix factorization

The matrices used to study in this part are Symmetric Positive Definite matrices.The LU factorization is implemented using 3 methods:

- Direct Implementation :
  The LU factroization is performed using hand-written loops.
- Level 2 Blas implementation :
  The LU factorization is performed using level 1 blas *xSCAL* and level 2 blas *GER*.
- Level 3 Blas implementation :
  The LU factorization is performed using block decomposition.The Level 2 blas implementation is used to factorize of block size r.The level 3 blas *dTRSM* and *dGEMM* are then used.

This implementation is performed for various sizes of the matrix.Then,a comparison is made with LU factorization implemented using LAPACK.

2.1 Result and Observations

The below table shows the computational time of various implementations for different matrix sizes.

Matrix (N*N)	Basic	L2	L3,r=6	L3,r=12	LAPACK
10	2.092e-06	4.31E-06	1.3393e-05	3.932e-06	0.0155106
20	6.381e-06	6.323e-06	1.1488e-05	1.245e-05	0.0085604
50	8.3457e-05	3.7526e-05	5.3639e-05	3.3289e-05	0.00681531
125	0.00137609	0.00357772	0.000313156	0.000174157	0.020171
250	0.00948855	0.00891025	0.0591454	0.00148911	0.0438293
500	0.297472	0.190004	0.0889699	0.083051	0.151946
1000	7.56314	0.879496	0.51795	0.170283	0.251475
1500	54.4139	4.0017	1.71489	0.693009	0.396071
2000	165.77	10.5012	2.3367	1.4063	1.30099

Table 1: Computational Time for different implementation of LU factorization

The below graph shows the computational time of different implementations for different matrix sizes.

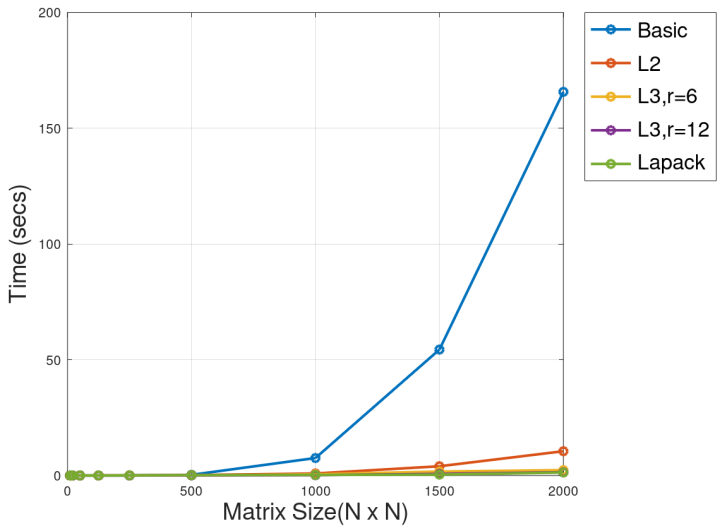

Figure 1: Plot for Time taken for different implementations of LU factorization

As seen from the plot, When the size of the matrices is of the lower order, the computational time taken by all the implementations are similar. But as the order of the matrices is increased, the computational time between the implementations can be distinguished. The computational time taken by the basic implementation is high. When compared with the Level 2 blas implementation, the computational time decreases rapidly. This is because the blas library optimizes the usage of memory allocation in the cache. The Level 3 blas implementation is more optimized than the Level 2 blas implementation. The Level 3 blas is implemented here using the block decomposition. It can be observed that as we increase the block size, the computational time is decreased. The Lapack library is a faster implementation compared to other implementations for higher-order matrices.

2.2 Error in Factorization

The Error in Factorization is calculated using the 2-norm.The below figure shows the Error in factorization for various LU factorization implementations.

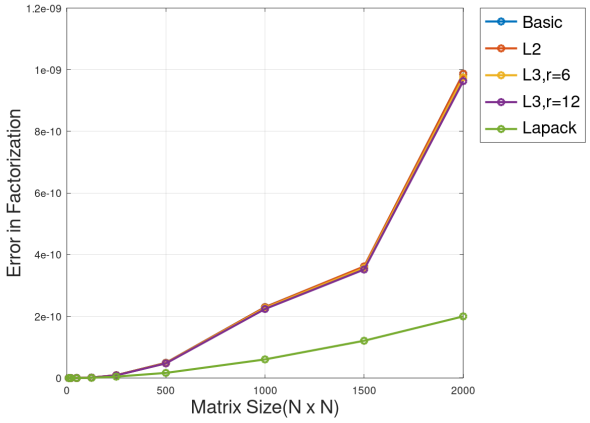

Figure 2: Error in Factorization for different implementations of LU factorization

It can be concurred from the plot that for lower order of matrices,the error is similar between all the implementations.But as the size of matrices is increased,the error using LAPACK is comparatively less when compared with the basic implementation.

3 Part 2:Symmetric Positive Definite Band Matrix

The matrices used to study in this part of the lab are Symmetric Positive Definite Band Matrix.Firstly,The matrices are stored in the dense format.Then LU factorization is performed on the matrices using LAPACK implementation .The computational time is recorded.The matrices are then stored in symmetric band storage.Then,this matrices are factorised using Choleksy factorization.The Chlolesky factorisation is implemented with LAPACK.The computational time is recorded and is compared with the previous method.

The below images shows the codes for computation of BandWidthUp and BandWidthDown

```
int computeBandwidthUp(const dmatrix_denseCM &A ){
 int n=A.getNbLines();
  int m=A.getNbColumns();
  int z=0;
 for(int i=0;i<m-1;i++){
   z=0;
    for(int j=0;j<m-1;j++){
      if((j+i+1)<m){
        if(A(j,j+i+1)==0){
        z+=1;
       }}}
      if(z==(n-1)){
        std::cout<<"z="<<z<<std::endl;
       return (m-z-1);}
      n--;
  }
```

Figure 3: Code for Computation of BandWidthUp

```
int  computeBandwidthDown(const dmatrix_denseCM &A ){
  int n=A.getNbLines();
  int m=A.getNbColumns();
  int z=0;
for(int i=0;i<m-1;i++){
   z=0;
    for(int j=0;j<m-1;j++){
      if((j+i+1)<m){
        if(A(j+i+1,j)==0){
        z+=1;
       }}}
      if(z==(n-1)){
       return (m-z-1);}
      n--;
  }
```

Figure 4: Code for Computation of BandWidthDown

The below image shows the code implementation of operator()(int i,int j) function

```
double &dsquarematrix_symband::operator()(int i, int j) {
  int k=lb+1;
  int diff=(j-i);
  int address=(k*j)-diff+lb;
  return *(a+address);
std::cout << "DO YOUR WORK HERE "<< __FILE__ << " Line " << __LINE__ << std::endl;
 throw;
}
```

Figure 5: Code for Computation of the Operator()(int i,int j)

3.1 Result and Observation

The following table represents the computation time of LU factorization using LAPACK and cholesky factorization.

Matrix (N*N)	LU	Cholesky
5	0.00111952	1.14E-16
1224	0.328017	0.0218929
1806	0.928027	9.06E-02
3948	6.15644	0.379068
4884	12.7626	0.0891945

Table 2: Computational Time between LU factorization and Cholesky

The following plot represents the computation time of LU factorization using LAPACK and cholesky factorization.

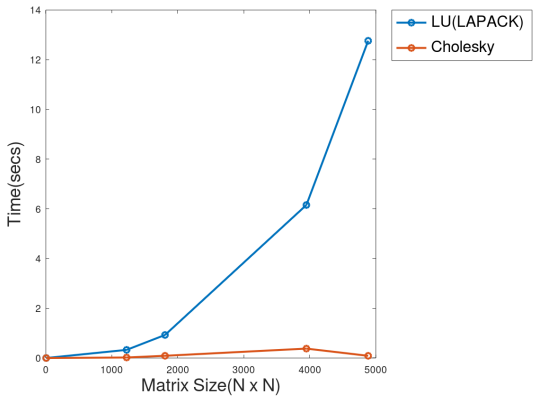

Figure 6: Plot for Computational time of LU and Cholesky Factorization

From the table and the plot above, It can be concurred that the computational time taken by Cholesky factorization is much less compared to the LU factorization. This is because the matrix is stored in the compressed Symmetric band storage. As matrices considered are sparse, storing them in the Symmetric band storage helps to access the non-zero elements in a faster way. With help of the LAPACK library, there is an optimization of the memory allocation in the cache. Therefore, the Cholesky factorization is much faster than LU factorization.

3.2 Error in Factorisation

The error in factorisation was calculated between the LU factorisation and the Cholesky factorisation using the 2-norm.The following plot was obtained:

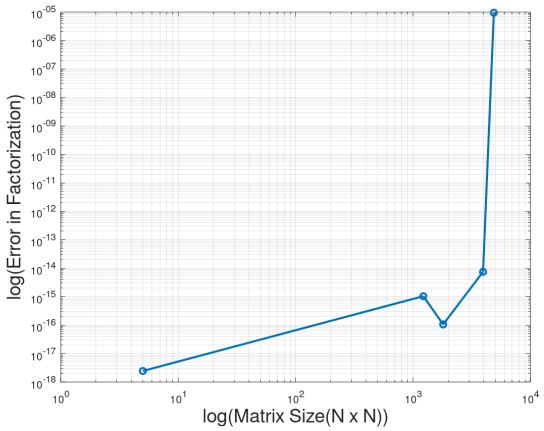

Figure 7: Error in Factorization

It can be concurred from the graph that the error is of very lower order.It is very close to zero.So,the factorization obtained in the both cases are similar but the computation time is very less when the matrices are stored in symmetric band storage.

4 Conclusion

In the first part of the study, we studied the computational time taken by the various implementations. The implementations considered were Basic, Level 2 Blas, Level 3 Blas, and LAPACK. The Level 3 blas was implemented using the block decomposition. It was observed that as the block size was increased the computational decreased. The time taken by the LAPACK library was comparatively less than the other implementations for higher-order matrices. So, we can conclude that the LAPACK implementation was the best way to compute the LU factorization of a positive definite dense matrix.

In the second part of the study, we studied the computational time between the LU factorization and Cholesky factorization. The matrices used for the study were symmetric positive definite band matrices. In the implementation of LU factorization, the matrices were stored in dense format. But in the implementation of Cholesky factorization, the matrices were stored in compressed symmetric band storage. Both the methods were implemented using the LAPACK library. It was observed that the computational time taken by Cholesky factorization is comparatively less. So, we can conclude that storing the Band matrices in Compressed Band storage form instead of the dense form helps us to reduce the computational time of the factorization.