PROCEEDINGS OF SCIENCE

Invertor - Program to Compute Exact Inversion of Large Matrices

R. Thiru Senthil[a,b,*]

[a] *The Institute of Mathematical Sciences, Taramani, Chennai 600113, India*

[b] *Homi Bhabha National Institute, Anushakti Nagar, Mumbai 400094, India*

E-mail: rtsenthil@imsc.res.in

We have written a general purpose code for exact inversion of large matrices in C language by treating matrices in block forms. We have optimized the computation speed using in-place inversion, dynamic memory handling and recursion techniques. This code is written to adapt with programs in C and Fortran language, which requires faster and exact solution for system of linear equations. We have applied it in our study of tau neutrino events at India based Neutrino Observatory (INO). With this program the time required for computing the exact inversion of matrices of order 100, 1000 are 6ms and 5.24s respectively in Intel i7 6700 CPU, 8GB RAM machine. We also present our procedure to implement parallel processing for calculating inverse by treating the matrix as large number of blocks and performing simultaneous operations among them.

41st International Conference on High Energy physics - ICHEP2022
6-13 July, 2022
Bologna, Italy

*Speaker

PoS(ICHEP2022)1129

1. Introduction

The requirement to calculate inversion of matrices is present in various science and engineering applications. Solving a system of linearly independent equations is a typical example which requires the inversion of matrix which is constructed out of the coefficients of the unknown variables in the system. If the matrix is large, calculating the exact inverse of a matrix tends to be a time consuming process. Usually numerical techniques are used in order to compute the solution. In this work, we present our program 'Invertor' written in C language, which calculates the exact inversion of large matrices by considering them in blocks form.

We start with a matrix M which is blocked as,

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{1}$$

Here the blocks A and D are square matrices while B and C are not necessarily need to be square matrices. If the inverse exists for the block A and its Schur complement $S_A = D - CA^{-1}B$, then the inverse of the matrix will be given as [1],

$$M^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BS_A^{-1}CA^{-1} & -A^{-1}BS_A^{-1} \\ -S_A^{-1}CA^{-1} & S_A^{-1} \end{bmatrix} \tag{2}$$

Similarly if the block D and its Schur complement $S_D = A - BD^{-1}C$ are invertible, then the inverse of the matrix is,

$$M^{-1} = \begin{bmatrix} S_D^{-1} & -S_D^{-1}BD^{-1} \\ -D^{-1}CS_D^{-1} & D^{-1} + D^{-1}CS_D^{-1}BD^{-1} \end{bmatrix} \tag{3}$$

During the computation of inverse of the matrix by starting with either A^{-1} or D^{-1}, requires additional memory for the calculation of their corresponding Schur complements. We avoid the usage of additional memory and perform the inversion in-place.

2. In place inversion methodology

In this procedure, we start with the matrix M, in blocks form and perform the calculations in each step at the same memory location of the matrix. The steps that we follow are:

1. In the matrix M which have blocks form, we compute A^{-1} and replace A with A^{-1}.

$$\begin{bmatrix} A^{-1} & B \\ C & D \end{bmatrix}$$

2. We now compute $-A^{-1} \times B$ at the location of B and replace B.

$$\begin{bmatrix} A^{-1} & -A^{-1}B \\ C & D \end{bmatrix}$$

3. Now, we replace D with its Schur complement $S_A = D - CA^{-1}B$ by computing $D + (C \times -A^{-1}B)$ using already computed $-A^{-1}B$ and the available C blocks.

$$\begin{bmatrix} A^{-1} & -A^{-1}B \\ C & S_A \end{bmatrix}$$

4. In next step we compute the inverse of Schur complement in place at the location of S_A block.

$$\begin{bmatrix} A^{-1} & -A^{-1}B \\ C & S_A^{-1} \end{bmatrix}$$

5. Next we calculate $-C \times A^{-1}$ and replace C.

$$\begin{bmatrix} A^{-1} & -A^{-1}B \\ -CA^{-1} & S_A^{-1} \end{bmatrix}$$

6. Now, we calculate $S_A^{-1} \times (-CA^{-1})$ at the location of $-CA^{-1}$ block.

$$\begin{bmatrix} A^{-1} & -A^{-1}B \\ -S_A^{-1}CA^{-1} & S_A^{-1} \end{bmatrix}$$

7. We can now compute $A^{-1} + (-A^{-1}B) \times (-S_A^{-1}CA^{-1})$ at the location of A^{-1} using rest of the blocks,

$$\begin{bmatrix} A^{-1} + A^{-1}BS_A^{-1}CA^{-1} & -A^{-1}B \\ -S_A^{-1}CA^{-1} & S_A^{-1} \end{bmatrix}$$

8. In final step, we multiply S_A^{-1} with $-A^{-1}B$ and obtain the complete inverted matrix M^{-1} which is,

$$\begin{bmatrix} A^{-1} + A^{-1}BS_A^{-1}CA^{-1} & -A^{-1}BS_A^{-1} \\ -S_A^{-1}CA^{-1} & S_A^{-1} \end{bmatrix}$$

We have avoided the requirement of additional memory for Schur complement inversion computation by adopting the above procedure. It can be noted that we perform at the maximum of two matrices multiplication only wherever required and these multiplications are such that the already existing block will get replaced by multiplying another block on it from the left or right. We perform these multiplication also in place without creating additional temporary memory. At the end of the procedure we see that the matrix M is replaced by the inverted matrix M^{-1}.

We compute inversion of matrix or blocks of order 2 or 3 directly (using derived analytical expressions). In the case of large matrices, we create sub blocks for A and S_A further recursively to compute the inverse. The computation time taken for inverting matrices of order 100 and 1000 are 0.006 s and 5.24 s respectively in Intel i7 6700 CPU, 8GB RAM machine. Similar procedure was also applied for the case of starting with D^{-1} (using equation 3). We found that there is no significant difference in computation time among the two procedures.

3. A discussion on parallel processing for inversion

By combining equations 2 and 3, we have,

$$M^{-1} = \begin{bmatrix} S_D^{-1} & -A^{-1}BS_A^{-1} \\ -D^{-1}CS_D^{-1} & S_A^{-1} \end{bmatrix} \tag{4}$$

for the blocked matrix M whose blocks A and D and their corresponding Schur complements are invertible [2]. Adopting equation 4 requires additional computation (twice in inversion comparing to procedure using equation 2 or 3). But if the original matrix M is blocked with large number blocks (as given in equation 5) where the diagonal blocks are square matrices, we can calculate the inversion with the guidance of equation 4. During this calculation, the inversion and multiplication among the blocks can be carried out simultaneously using parallel processing.

$$M = \begin{bmatrix} Block_{11} & | & Block_{12} & | & \dots & | & Block_{1k} \\ \vdots & & \ddots & & & & \vdots \\ Block_{k1} & | & Block_{k2} & | & \dots & | & Block_{kk} \end{bmatrix} \tag{5}$$

The development of algorithm with parallel processing for inverting matrices by treating them as large number of blocks will be the future progress in our analysis.

The results that we have discussed here do not include the floating point round off to a given precision. This is required to avoid the overflow and underflow of values in the memory during the intermediate computation. Even though the precision control slows down the computation significantly, it is necessary and will be included in the upcoming results.

4. Conclusions

We have presented our preliminary result of performing inversion of large matrices by treating them in blocks form. Even though the procedure of inverting large matrices using block inversion technique is well known, our aim of this presentation is to provide readily available interface to adopt with different users in C and Fortran who require exact fast inversion for large matrices. Our future results will include the parallel processing algorithms and precision control over blockwise inversion. The program is available at the web page https://www.imsc.res.in/~rtsenthil/invertor.html.

Acknowledgments

RTS would like to thank D. Indumathi for discussions and motivation to this work. We acknowledge the use of high performance computing facility - Nandadevi cluster at the Institute of Mathematical Sciences.

References

[1] Bernstein Dennis S, *Matrix Mathematics: Theory, Facts, and Formulas* (Second Edition). Princeton University Press, 2009.

[2] Horn RA and Johnson CR, *Matrix Analysis*, Cambridge University Press, 2013.