Invertor - Program to Compute Exact Inversion of Large Matrices

R. Thiru Senthil [1,2]

[1]The Institute of Mathematical Sciences, Taramani, Chennai 600113, India
[2]Homi Bhabha National Institute, Anushakti Nagar, Mumbai 400094, India

Contact

+91 (44) 22543371
rtsenthil@imsc.res.in
https://www.imsc.res.in/∼rtsenthil

Abstract

We have written a general purpose code for analytical 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 adopt with programs which requires the faster and exact solution of system of linear equations in C and Fortran. We have applied it in our study of tau neutrino events at India based Neutrino Observatory (INO). As example, 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 alternate technique and results of computing inversion of ultra large matrices (of order $> 10^4$) using parallel processing in clusters.

Introduction

The requirement to calculate inversion of matrices is present in various science and enginnering applications. A system of linearly independent equations is solved by computing inverse of matrix constructed out of the coeffients of unknown variables. If the system is large, the calculation of analytical solution using matrix inversion method tends to be time consuming process and we shall use various numerical techniques in order to compute the solution. The choice of technique to compute matrix inversion is compromised for accuracy over time. In this work, we present the analytical computation of inverse of large matrices by considering them in block forms. We have optimized the computation time using recursion and inplace inversion during coding. It is written in C language.

Methodology

Let us have a matrix M in block form as

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

We have blocks A and B as square matrices whereas blocks B and C are not necessarily in square matrices. If the block A is non-singular, we can calculate the inverse of this block as A^{-1}, then the inverse of full matrix is given by,

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

where as $S_A = (D - CA^{-1}B)$ is Schur complement of block A. It is necessary that S_A^{-1} exists in order to have the inverse of complete matrix.

The inverse could be calculated by following steps. We describe here computation and status of inverse matrix in each stage. In each calculation we replace the computed elements in the same memory of the resultant inverted matrix. It avoids overuse of memory in calculation.

1. First we compute A^{-1}.

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

2. We calculate $S_A = (D - CA^{-1}B)$. This step required dynamic memory allocatin for S_A.
3. We compute S_A^{-1} and we shall store this result in place of D, then we can free the memory used temporavarily to compute S_A.

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

4. In the last step we can compute the remaining blocks as required using A^{-1}, S_A^{-1} and the blocks of given matrix.

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

In this procedure, for computing inverse of large matrix we temporarily create memory during the calculation of S_A^{-1}. This can be avoided if we follow different approach with inplace inversion. This will improve the computation speed significantly.

In Place Inversion Methodology

In this procedure, we shall start with the given matrix and compute the inverse in each stage at the same memory location of the started matrix. The steps that we follow are:

1. We start with

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

2. First we compute A^{-1} in the place of A.

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

3. We now compute $A^{-1}B$ by left multiplying $-A^{-1}$ with B.

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

4. Now, we compute the Schur complement of A using A^{-1}, $A^{-1}B$ and C and replaces with D as,

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

5. In next step we compute the inverse of Schur complement $S_A^{-1} = (D - CA^{-1}B)^{-1}$.

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

6. Next we calculate CA^{-1} by right multiplying A^{-1} with C.

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

7. Now, we left multiply $-S_A^{-1}$ with CA^{-1}.

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

8. We can now compute the inverted form at the location of A^{-1} using rest of the blocks as,

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

9. In final step, we right multiply S_A^{-1} with $-A^{-1}B$.

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

Using the above procedure, we avoid the requirement of additional memory for computation. In short, the original matrix is replaced by the inverted matrix after applying this procedure.

Inverting Large Matrices

A matrix of order 2 or 3 could directly be computed. When the matrix order exceeds 3, we block them into the form given above and compute the inverse using the procedure described. In the case of very large matrices, we can subblock A further and compute inverse by calling the same function (Recursion). We further apply the same resursion technique to compute inverse of S_A also. Since arrays are treated as pointers in C, this is achievable with out storing additional memory.

Results

The results of optimized in place inversion code is listed in the table. We used Intel i7 6700 CPU, 8GB RAM machine in this analysis.

Order	Time in seconds
100	0.006
1000	5.24

Table 1: Computation time for matrix inversion

A Discussion on Parallel Processing for Inversion

If we start with D^{-1}, we can also calculate the the inverted matrix as,

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

where as Schur complement of block D is given as $S_D = A - BD^{-1}C$.

We could combine using A^{-1} and D^{-1} to start with, then the full inversion is given by,

$$\begin{bmatrix} (A - BD^{-1}C)^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}$$

This provides the possibility of performing the calculation using parallel processing. By simultaneously computing A^{-1} and D^{-1} of low order sub blocks we performed the calculations by designing multiple alogorithms. So far, the computation time required with those procedure are slightly larger than the inplace inversion technique. This is due to the requirement of additional multiplication and inversions required during the sub processes. As an example for a matrix of order 1000, the process completed in $15.62s$ in Nandadevi cluster having 40 cores. It is significantly higher comparing to the standard in place inversion code which comptes inversion in $6.94s$.

Conclusions

- We have written C based Invertor program to invert large matrices by treating the matrices in block forms.
- We have optimized the computation time using dynamic memory handling and in place inversion techniques with recursion.
- We performed the computation using parallel processing in various possible ways but the computation time is yet to be optimized.

Acknowledgements

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