Method to Calculate the Inverse of a Complex Matrix using Real Matrix Inversion

By Andreas Falkenberg

Consultant

Im Espenhagen 10 ; 51702 Bergneustadt; Germany

Abstract

This paper describes a simple method to calculate the invers of a complex matrix. The key element of the method is to use a matrix inversion, which is available and optimised for real numbers. Some actual libraries used for digital signal processing only provide highly optimised methods to calculate the inverse of a real matrix, whereas no solution for complex matrices are available, like in [1]. The presented algorithm is very easy to implement, while still much more efficient than for example the method presented in [2].

1. Introduction and Motivation

Matrix inversion is a task, which often is required in modern digital signal processing systems. For example in many modern telecommunication systems matrix inversion is used for instance to calculate the parameters for equalizers. One interesting example is provided by the TD-CDMA standard [5], where the Cholesky decomposition is proposed as key element of the receiver algorithm. The motivation for this work was that no implementation of a complex matrix inversion is provided in the C/C++ Library, which is part of the VisualDSP++ development environment for the analog devices DSPs [1]. On the other hand a very efficient implementation of the matrix-inversion algorithm for real matrices is provided already in the same system making use of the hardware capabilities of the TigerSHARC like SIMD and parallel instructions. The solution, which is provided in literature [2] to reduce the complex matrix inversion to real matrix inversion is not sufficient enough due to its high complexity. On the other hand the implementation of the entire SVD algorithm or any other algorithm using complex arithmetic is certainly a good solution, but may not fully utilize the already highly optimised matrix-inversion, which is given through the referenced library [1]. Further the time involved to implement and test the entire SVD needs to be considered. Especially producing a highly optimised version involves lots of assembler optimisation. The herein presented solution is estimated to be only about 15% more complex than the pure SVD implementation, whereas it can be implemented within three lines of code, when the real matrix implementation is available.

2. The problem of complex matrix inversion

First we assume that the following system of complex equations shall be solved, using a matrix inversion. Let A and C real matrices. In fact the A contains the real portion and C the imaginary portion of the matrix to be inverted.

$$(A + iC) \cdot (x + iy) = (b + id)$$

The result of the equations is given as *b* and *d*, which are also real vectors, representing the real and the imaginary part of the resulting complex vector. The goal is to find *x* and *y*, hence solving the equations. To solve the shown system of equations the complex matrix inversion can be used:

$$(A+iC)^{-1}\cdot(A+iC)\cdot(x+iy)=(A+iC)^{-1}\cdot(b+id)$$

Which leads to the required solution:

$$(x+iy)=(A+iC)^{-1}\cdot(b+id)$$

Since this problem occurred as part of a DSP implementation, first the availability of a complex matrix inversion algorithm in the library was considered. Since the matrix inversion for complex matrices was not available, the first solution was taken from [1], which is to solve the following system:

$$\begin{aligned} A\cdot x-C\cdot y&=b\\ C\cdot x+A\cdot y&=d \end{aligned}$$

This can be written as problem using a real matrix inversion, with the following formula:

$$\begin{pmatrix} A & -C\\ C & A \end{pmatrix}\cdot\begin{pmatrix} x\\ y \end{pmatrix}=\begin{pmatrix} b\\ d \end{pmatrix}$$

With this solution it is possible to use the highly optimized real matrix inversion, which was already part of the given library. This method works for both pseudo-inverse as well as inverse matrices. As already shown in [1] this solution is estimated to be about 100% less efficient than a well prepared and optimized SVD algorithm specifically for complex matrices. Although this may be a quick solution to proof the functionality, it may not be sufficient for production code.

3. Solution

So now again we are looking to improve the inversion of this real matrices. We start with the method provided in [3][4] which shows a recursive implementation of a matrix inversion.
First we split the matrix in four submatrices, which we then inverted. So essentially the method provides a solution for the following problem:

$$\begin{pmatrix} y_{00} & y_{01}\\ y_{10} & y_{11} \end{pmatrix}=\begin{pmatrix} x_{00} & x_{01}\\ x_{10} & x_{11} \end{pmatrix}^{-1}$$

This can be done using the following set of equations, which reduces the 2Nx2N Matrix inversion to two NxN Matrix Inversions as described in [3].

$$r_0 = x_{00}^{-1}$$
$$r_1 = x_{10} \cdot r_0$$
$$r_2 = r_0 \cdot x_{01}$$
$$r_3 = x_{10} \cdot r_2$$
$$r_4 = r_3 - x_{11}$$
$$r_5 = r_4^{-1}$$
$$y_{01} = r_2 \cdot r_5$$
$$y_{10} = r_5 \cdot r_1$$
$$r_6 = r_2 \cdot y_{10}$$
$$y_{00} = r_0 - r_6$$
$$y_{11} = -r_5$$

This result in itself does not improve our situation. Since we still are in the same order of complexity. Still these formulas can help to generate a nice solution for the complex inversion. We need to combine it with the previous result, which we know about complex matrix inversion. So first we replace the arguments x with the matrices A and C. This leads to the following set of equations:

$$r_0 = A^{-1}$$
$$r_1 = C \cdot r_0$$
$$r_2 = -r_0 \cdot C$$
$$r_3 = C \cdot r_2$$
$$r_4 = r_3 - A$$
$$r_5 = r_4^{-1}$$
$$y_{01} = r_2 \cdot r_5$$
$$y_{10} = r_5 \cdot r_1$$
$$r_6 = r_2 \cdot y_{10}$$
$$y_{00} = r_0 - r_6$$
$$y_{11} = -r_5$$

Now we write down the formulas more explicit:

$$y_{01} = -A^{-1} \cdot C \cdot (-C \cdot A^{-1} \cdot C - A)^{-1}$$
$$y_{10} = (-C \cdot A^{-1} \cdot C - A)^{-1} \cdot C \cdot A^{-1}$$
$$y_{00} = A^{-1} + A^{-1} \cdot C \cdot y_{10}$$
$$y_{11} = -(-C \cdot A^{-1} \cdot C - A)^{-1}$$

We can also easily show that in fact the following is true:

$$y_{10} = -y_{01}$$
$$y_{00} = y_{11}$$

This set of equations now can be reduced such that the smallest possible number of NxN inversions and NxN multiplications is reached. With the following set of equations:

$$y_{01} = A^{-1} \cdot C \cdot (C \cdot A^{-1} \cdot C + A)^{-1} = \left(C + A \cdot C^{-1} A\right)^{-1}$$
$$y_{10} = -(C \cdot A^{-1} \cdot C + A)^{-1} \cdot C \cdot A^{-1} = -(C + A \cdot C^{-1} A)^{-1}$$
$$y_{00} = A^{-1} + A^{-1} \cdot C \cdot y_{10}$$
$$y_{11} = (C \cdot A^{-1} \cdot C + A)^{-1}$$

We can generate the following result:

$$r_0 = A^{-1} \cdot C$$
$$y_{01} = r_0 \cdot (C \cdot r_0 + A)^{-1}$$
$$y_{10} = -(C \cdot r_0 + A)^{-1} \cdot C \cdot A^{-1}$$
$$y_{00} = A^{-1} + r_0 \cdot y_{10}$$
$$y_{11} = (C \cdot r_0 + A)^{-1}$$

So with the above shown the total is:

$$r_0 = A^{-1} \cdot C$$
$$y_{01} = r_0 \cdot y_{11}$$
$$y_{10} = -y_{01}$$
$$y_{00} = y_{11}$$
$$y_{11} = (C \cdot r_0 + A)^{-1}$$

So our original matrix was $(A + iC)$ so now our resulting inverse matrix is $(y_{00} + iy_{10})$. The shown solution is very easy to implement when only a real inversion is available. So a complex NxN inversion is calculated through two real inversions and three real matrix multiplications.

4. Conclusion

The solution is a very easy to implement method for complex matrix inversion. In fact it can be implemented in three lines of code, since we do not explicitly distinguish between y_{00} and y_{11} in the final code. The same is true for y_{10} and y_{01}. The goal was not to find the best possible algorithm but to find the best possible algorithm while still using the already highly optimised real matrix inversion. The shown method is much better than the actual “quick and dirty” solution as shown in [2] but about equal in implementation effort. Theoretically we can assume complex matrix inversion to be four times more complex than real matrix inversion, if the implementation is sufficiently performed. With this we are in the range of 5 times of a real matrix in version, which is still about 40% better than the solution in [2]. In fact depending on the pressure of the project this solution always should replace the “quick and dirty” solution provided in [2] and in many cases still may replace a new SVD implementation whatsoever.

REFERENCES

[1] Visual DSP++ 4.0 C/C++ Compiler and Library Manual for TigerSHARC Processors; Analog Devices; 2005.

[2] W. Press, S.A. Teukolsky, W.T. Vetterling, B.R. Flannery; Numerical Recipes in C++, The art of scientific computing, Second Edition; p52 : “Complex Systems of Equations”;Cambridge University Press 2002.

[3] W. Press, S.A. Teukolsky, W.T. Vetterling, B.R. Flannery; Numerical Recipes in C++, The art of scientific computing, Second Edition; p106 : “Is Matrix Inversion an N^3 process”;Cambridge University Press 2002.

[4] V. Strassen, Numerische Mathematik, vol.13, pp.354; 1969.

[5] P. W. Baier, T. Bing, and A. Klein, "TD-CDMA”, in *Third generation mobile communication systems*”, R. Prasad, W. Mohr, and W. Konhäuser (Eds.), chapter 2, pp. 25-72. Boston, London: Artech House, 2000.