An Efficient Algorithm to Compute the Inverse of a Fourth Order Positive Definite Symmetric Matrix

Prakash A. Kulkarni, K. N. Sudharshan
Accord Software & Systems Private Limited, Bangalore, India

BIOGRAPHY

Prakash. A. Kulkarni is a Senior Engineer at Accord Software and Systems. He did his M.Tech from IIT, Bombay, India in 1990 in Control System Engineering. His areas of interest include GPS based navigation, Differential Geometric Control Theory and Digital Communication.

K. N. Sudharshan is a Systems Engineer at Accord Software and Systems. He has a B.E in Electronics and Communication. His areas of interest are algorithm development and GPS based application development.

ABSTRACT

This paper presents an efficient and simple algorithm to compute the inverse of a fourth order positive definite symmetric matrix. In GPS related algorithms (whether iterative or non-iterative), the computations of user position or velocity require determination of the inverse of such 4x4 symmetric matrices.

The standard Gaussian elimination technique based on ***LU*** decomposition technique does not exploit the advantage of matrix symmetry. The approach described here involves a single step $\boldsymbol{LDL}^T$ decomposition [Golub 1996] to obtain a block diagonal matrix with two blocks. The blocks will be of the size 1x1 and 3x3. The inverse of the first block is obvious, where as the inverse of the second block may be computed using the classical formula ***Adj(G)/ det(G)***. Pivotation can be performed efficiently, again by exploiting the symmetry. This algorithm requires considerably lesser floating-point operations than the standard algorithms.

INTRODUCTION

Fundamental to GPS based navigation are the pseudo-range and delta-range equations [Kaplan, 1996]. Receiver position, receiver velocity, receiver clock parameters (clock bias and clock drift) are obtained by solving these equations.

Let ***(x, y, z)*** denote the receiver position vector when it receives the GPS signals from a satellite $\boldsymbol{S_k}$. Let $\boldsymbol{(x_k, y_k, z_k)}$ denote the position vector of $\boldsymbol{S_k}$ at the instance of signal broadcast, in the ECEF coordinate system [Kaplan] Let ***b*** be the receiver clock bias. The pseudo-range $\boldsymbol{\rho_k}$ corresponding to the satellite $\boldsymbol{S_k}$ is given by

$$\rho_k = \sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + c*b \qquad \ldots(1)$$

where, ***c*** is the velocity of light.

Similarly let ***(u, v, w)*** and $\boldsymbol{(u_k, v_k, w_k)}$ denote the receiver velocity vector and the velocity of $\boldsymbol{S_k}$ respectively represented in ECEF coordinate system. Let ***d*** denote the receiver clock drift rate. The pseudo-delta-range $\boldsymbol{\delta\rho_k}$ are given by

$$\begin{aligned}\delta\rho_k = [&(u_k - u)(x_k - x) + \\ &(v_k - v)(y_k - y) + \\ &(w_k - w)(z_k - z)] / r_k + c*d\end{aligned} \qquad \ldots(2)$$

The input quantities to any navigation algorithm are the pseudo-ranges, the pseudo-delta-ranges, the satellite positions and the satellite velocities. The receiver position, the receiver velocity, the clock bias and the clock drift are the unknowns to be estimated. At least four pseudo-range and pseudo-delta-range equations from different satellites are required to determine these unknowns.

However, most of the GPS receivers, track more than four satellites at a time, for better accuracy and reliability (e. g. Receiver Autonomous Integrity Monitoring [Young, 1986] requires at least five satellite measurements).

Redundant satellite measurements give rise to an over determined system of equations, where the number of unknowns is less than the number of equations. In such a situation, least square estimates are sought after. There exist iterative techniques based on Newton-Raphson method and also non-iterative techniques based on Bancroft's method [Strang, 1997] which are used to give least square solutions. While attempting to obtain the least square solution, by any of the above mentioned methods, one comes across an over determined linear system of equations of the form:

$$\boldsymbol{AX} = \boldsymbol{b}, \tag{3}$$

where, $\boldsymbol{A}$ is n x 4 matrix with n > 4, $\boldsymbol{X}$ is 4 x 1 vector and $\boldsymbol{b}$ is n x 1 vector.

The least square estimate of the above equation is given by

$$\boldsymbol{X} = \boldsymbol{S}^{-1}\boldsymbol{A}^{T}\boldsymbol{(b)} \tag{4}$$

where, $\boldsymbol{S} = \boldsymbol{A}^{T}\boldsymbol{A}$ is 4 x 4 symmetric matrix. Often the matrix $\boldsymbol{A}$ happens to be the Geometry Matrix $\boldsymbol{G}$ (i.e., Jacobean of the nonlinear pseudo-range equations 1).

The accuracy of the computed receiver position and velocity depends on two factors: measurement accuracy and satellite geometry. The various DOP values (Dilution Of Precision) determine the extent of inaccuracy caused by the satellite geometry in the receiver position and velocity. These DOP values are obtained by computing inverse of $\boldsymbol{S}=\boldsymbol{G}^{T}\boldsymbol{G}$.

Thus the computation of the inverse of such 4x4 symmetric positive definite matrices is necessary.

A standard practice to compute the inverse of a general square matrix is the Gaussian elimination technique based on $\boldsymbol{LU}$ [Golub, 1996] decomposition. However, when the matrix is symmetric, computational effort will be much less if $\boldsymbol{LDL}^{T}$ [Golub, 1996] decomposition technique is employed.

The approach presented in this paper, uses $\boldsymbol{LDL}^{T}$ technique and the classical formula of inverse

$$\boldsymbol{inv(M) = adj(M) / det(M)} \tag{5}$$

This approach results in an algorithm that uses considerably less floating-point operations. Also symmetry of the matrix is exploited to reduce data memory requirement. A pivotation technique presented in this paper requires only 3 swaps.

The mathematical details of the algorithm are explained in the next section. In the appendix a MATLAB function implementing the algorithm is presented.

ALGORITHM DESCRIPTION

The algorithm is explained in this section. It involves five steps. The algorithm is presented as a series of matrix transformations to emphasize the theory. Implementation details are presented in the appendix.

Consider Eq (3). Let $\boldsymbol{S}=\boldsymbol{A}^{T}\boldsymbol{A}$ be represented by

$$\boldsymbol{S}=\begin{bmatrix} S_{11} & S_{12} & S_{13} & S_{14} \\ S_{12} & S_{22} & S_{23} & S_{24} \\ S_{13} & S_{23} & S_{33} & S_{34} \\ S_{14} & S_{24} & S_{34} & S_{44} \end{bmatrix} \tag{6}$$

The columns of $\boldsymbol{A}$ are assumed to be linearly independent (this is the case when more than four distinct satellite measurements are used).

With this assumption and from the fact that $\boldsymbol{S}=\boldsymbol{A}^{T}\boldsymbol{A}$, the matrix $\boldsymbol{S}$ has the following properties

- $\boldsymbol{S}$ is symmetric and positive definite.
- The determinant of $\boldsymbol{S}$ is positive.
- All diagonal elements of $\boldsymbol{S}$ are positive.

Step1: Diagonal Pivotation

Diagonal pivotation is employed here to prevent division by small numbers.

First, the largest diagonal member of $\boldsymbol{S}$ is identified. If $\boldsymbol{S_{kk}}$ is the largest diagonal element of matrix $\boldsymbol{S}$, then swap 1^{st} row with k^{th} row and 1^{st} column with the k^{th} column of matrix $\boldsymbol{S}$. The symmetry is preserved after these operations. These operations can be represented as multiplication with the permutation matrices. Let $\boldsymbol{P_k}$ represent the permutation matrix that swaps k^{th} row and the 1^{st} row.

$$\boldsymbol{P}_2=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{P}_3=\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{P}_4=\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

The resulting matrix $\boldsymbol{R}$, is related to $\boldsymbol{S}$ as given below,

$$\boldsymbol{R} = \boldsymbol{P_k}\, \boldsymbol{S}\, \boldsymbol{P_k}^{T} \tag{8}$$

Step2: Block Diagonalization by Gaussian Elimination Technique

The matrix $\boldsymbol{R}$ defined in Eq no (8) is block diagonalized by carrying out elementary row and column operations. For this, consider the following lower triangular matrix $\boldsymbol{L}$.

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ r_2 & 1 & 0 & 0 \\ r_3 & 0 & 1 & 0 \\ r_4 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Here, $r_j = - S_{j1} / S_{11}$, for j = 2,3 and 4.

Let $\boldsymbol{B} = \boldsymbol{LRL}^{T}$ (10)

Now the matrix $\boldsymbol{B}$ will be of the form

$$\boldsymbol{B} = \begin{bmatrix} b_{11} & 0 & 0 & 0 \\ 0 & b_{22} & b_{23} & b_{24} \\ 0 & b_{23} & b_{33} & b_{34} \\ 0 & b_{24} & b_{34} & b_{44} \end{bmatrix} \tag{11}$$

Step3: 3x3 Block Inversion

The above $\boldsymbol{B}$ matrix is a symmetric block diagonal matrix of the form

$$\boldsymbol{B} = \begin{bmatrix} b_{11} & \boldsymbol{0} \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{B_{22}} \end{bmatrix} \tag{12}$$

where the block $\boldsymbol{0}$ is [0, 0, 0] and the block $\boldsymbol{B_{22}}$ is

$$\boldsymbol{B}_{22} = \begin{bmatrix} b_{22} & b_{23} & b_{24} \\ b_{23} & b_{33} & b_{34} \\ b_{24} & b_{34} & b_{44} \end{bmatrix} \tag{13}$$

Clearly the inverse of the block diagonal matrix is given by

$$\boldsymbol{B}^{-1} = \begin{bmatrix} 1/b_{11} & \boldsymbol{0} \\ \boldsymbol{0}^{T} & \boldsymbol{B_{22}}^{-1} \end{bmatrix} \tag{14}$$

Here, the inverse of the 3x3 symmetric matrix B_{22} needs to be computed. To compute $\boldsymbol{B_{22}^{-1}}$ one can use the following classical formula while taking the advantage of the symmetry of $\boldsymbol{B_{22}}$.

$$\boldsymbol{B_{22}^{-1} = adj\ (B_{22}) / det\ (B_{22})} \tag{15}$$

Instead of using Eq no (15), the inverse of B22 can be computed by techniques explained in steps 2 and 3. The matrix B22 can be block diagonalized into 1x1 and 2x2 matrices and then the Eq no 15 can be used to find the inverse of 2x2-block matrix. By doing this one can save several floating-point multiplication operations at the cost of one extra division.

Step4: Reverse Transformation

This step finds $\boldsymbol{R^{-1}}$ from $\boldsymbol{B^{-1}}$.
From Eq no (10) i.e., $\boldsymbol{B = L\, R\, L^{T}}$.

$$\boldsymbol{R^{-1} = L^{T}\, B^{-1}\, L}$$

Step 5: Back Pivotation

From eq no (8) i.e., $\boldsymbol{R = P_k\, S\, P_k^{T}}$.

$$\boldsymbol{S^{-1} = P_k^{T}\, R^{-1}\, P_k}$$

The above steps of finding the inverse of $\boldsymbol{S}$ can be summarized by the equation given below

$$\boldsymbol{S^{-1} = P_k^{T}\, L^{T}\, B^{-1}\, L^{-1}\, P_k}$$

CONCLUSION

Comparison of computation of inverse of a 4x4 symmetrical matrix, based on general inverse routines and the algorithm described here is given below.

METHODS / FLOP TYPE	Existing Methods (Typical)		Proposed Methods (Implemented in the appendix)	
	Using LU decomposition.	Using LDL' only	Adj/det applied for 3x3 block	Adj/det applied for 2x2 block
Addition	38	40	20	27
Subtraction	24	-	6	1
Multiplication	84	52	42	37
Division	4	4	2	3
Total	150	96	70	68

The algorithm is implemented on Analog Devices ADSP2189M fixed point processor. The input data to the processor is in IEEE floating-point format. A chart indicating the number of processor cycles, total number of

program memory words and data memory words is given below.

Processor Cycles	Program Memory Words	Data Memory Words
23,782	407	125

APPENDIX

```
%=======================================
% IMPLEMENTATION OF THE ALGORITHM AS A
% MATLAB FUNCTION
%=======================================
function S = sym4inv ( S );
%  Step1: Diagonal Pivotation
max = S(1, 1);
for j = 2 : 4
        k = 1;
        temp = S(j, j);
        if temp > max
                max = temp;
                k = j;
        end
end

if max < 1.0e-9
        fprintf ('WARNING:Division by small number);
end
if k ~= 1
        for  i  =  2 : 4
                if  i  <  k
                        temp = S(1, i);
                        S(1, i) = S(i, k);
                        S(i, k) = temp;
                end
                if i > k
                        temp = S(1, i);
                        S(1, i)= S(i, k);
                        S(i, k) = temp;
                end
        end
        temp = S(1, 1);
        S(1, 1) = S(k, k);
        S(k, k) = temp;
end
%-------------------------------------------------------------------
% Step 2: Block Diagonalization by Gaussian Elimination
% Technique.
%-------------------------------------------------------------------
S (1, 1) = 1 / S(1, 1);
for  i = 2 : 4
        r( i ) = -S(1, i) * S(1, 1);
                for j = i : 4
                        S(i, j) = S(i, j) + r( i )* S(1, j);
                end
end
%-------------------------------------------------------------------
%  Step3:  3x3 Block Inversion
%-------------------------------------------------------------------
b(1, 1) = S(2, 2); b(2, 2) = S(3, 3); b(3, 3) = S(4, 4);
b(1, 2) = S(2, 3); b(1, 3) = S(2, 4); b(2, 3) = S(3, 4);
b = inv3x3(b);
S(2, 2) = b(1, 1); S(3, 3) = b(2, 2); S(4, 4) = b(3, 3);
S(2, 3) = b(1, 2); S(2, 4) = b(1, 3); S(3, 4) = b(2, 3);
%-------------------------------------------------------------------
% Step 4: Reverse Transformation:
%-------------------------------------------------------------------
for i = 2 : 4
        S(1, i)= 0;
        for j = 2 : 4
                if j >= i
                        S(1, i) = S(1, i) + r( j ) * S(i, j);
                else
                        S(1, i)= S(1, i) + r(j)* S(j, i);
                end
        end
        S(1, 1) = S(1, 1) + r(i) * S(1, i);
end
%-------------------------------------------------------------------
% Step 5: Back Pivotation :
%-------------------------------------------------------------------
if  k ~= 1
        for i = 2 : 4
                if i < k
                        temp = S(1, i);
                        S(1, i) = S(i, k);
                        S(i, k) = temp;
                end
                if i > k
                        temp = S(1, i);
                        S(1, i) = S(i, k);
                        S(i, k) = temp;
                end
        end
        temp = S(1, 1);
        S(1, 1)= S(k, k);
        S(k, k) = temp;
end
%-------------------------------------------------------------------
% Lower triangular portion
%-------------------------------------------------------------------
for i = 1 : 4
        for j = i : 4
                S(j, i) = S(i, j);
        end
end

%=======================================

function S = inv3x3(S);

% Using block diagonalization technique
%-------------------------------------------------------------------
% Step1: Diagonal Pivotation
```

```
%-------------------------------------------------------------------
max = S(1,1);
k = 1;
if S(2, 2) > max
        max = S(2, 2);
        k = 2;
end
if S(3, 3) > max
        k = 3;
end
if k = = 2
  temp = S(1, 1); S(1, 1) = S(2, 2); S(2, 2) = temp;
  temp = S(1, 3); S(1, 3) = S(2, 3); S(2, 3) = temp;
end
if k = = 3
  temp = S(1, 1); S(1, 1) = S(3, 3); S(3, 3) = temp;
  temp = S(1, 2); S(1, 2) = S(2, 3); S(2, 3) = temp;
end
%-------------------------------------------------------------------
% Step2: Block Diagonalization
%-------------------------------------------------------------------
S(1, 1) = 1 / S(1, 1);
for i = 2:3
        r(i) = -S(1, i) * S(1, 1);
        for j = i:3
                S(i, j) = S(i, j) + r(i)* S(1, j);
        end
end

%-------------------------------------------------------------------
% Step 3: 2x2 block inversion
%-------------------------------------------------------------------
% Determinant Compuation
det = S(2, 2)* S(3, 3) - S(2, 3)* S(2, 3);
if det < 1.0e-9
        fprintf ('WARNING: Matrix is ill-poised or
        badly scaled results might be erroneous');
end
idet = 1/det;
%Computation of inverse of 2x2 block
temp = S(2, 2);
S(2, 2) = S(3, 3) * idet;
S(3, 3) = temp * idet;
S(2, 3) = -S(2, 3)* idet;
%-------------------------------------------------------------------
% Step 4: reverse transformation:
%-------------------------------------------------------------------
for i = 2 : 3
        S(1, i)= 0;
        for j = 2 : 3
                if j >= i
                        S(1, i)= S(1, i) + r(j)* S(i, j);
                else
                        S(1, i)= S(1, i) + r(j)* S(j, i);
                end
        end
        S(1, 1) = S(1, 1) + r(i) * S(1, i);
end
%-------------------------------------------------------------------
% Step5: Back Pivotation
%-------------------------------------------------------------------
if k == 2
  temp = S(1, 1); S(1, 1) = S(2, 2); S(2, 2) = temp;
  temp = S(1, 3); S(1, 3) = S(2, 3); S(2, 3) = temp;
end
if k = = 3
        temp = S(1, 1); S(1, 1) = S(3, 3); S(3, 3) = temp;
        temp = S(1, 2); S(1, 2) = S(2, 3); S(2, 3) = temp;
        end
%-------------------------------------------------------------------

% An Alternate function used to compute the inverse of
%  3x3 symmetric block .
function S = inv3x3(S);
%-------------------------------------------------------------------
% Adjoint Computation
%-------------------------------------------------------------------
b(1, 1) = S(2, 2)* S(3, 3) - S(2, 3)* S(2, 3);
b(2, 2) = S(1, 1)* S(3, 3) - S(1, 3)* S(1, 3);
b(3, 3) = S(1, 1)* S(2, 2) - S(1, 2)* S(1, 2);
b(1, 2) = S(1, 3)* S(2, 3) - S(1, 2)* S(3, 3);
b(1, 3) = S(1, 2)* S(2, 3) - S(1, 3)* S(2, 2);
b(2, 3) = S(1, 2)* S(1 ,3) - S(1, 1)* S(2, 3);
%-------------------------------------------------------------------
% Determinant Computation
%-------------------------------------------------------------------
det = S(1, 1)* b(1, 1) + S(1, 2)* b(1, 2) + S(1, 3)* b(1, 3);
if det < 1.0e-9
        fprintf ('WARNING: Matrix is may be ill-poised
        or badly scaled. Results might be erroneous');
end
%-------------------------------------------------------------------
% Computation of inverse of 3x3 block
%-------------------------------------------------------------------
idet = 1/det;
for i = 1 : 3
        for j = i : 3
        S(i, j) = b(i, j) * idet;
        end
end
%=====================================
```

REFERENCES

Golub H.G and Charles F. Van Loan (1996),"Matrix Computations", *The Johns Hopkins University Press*, Baltimore, Maryland, pp 94-206

Elliott D. Kaplan (1996), "Understanding GPS: Principles and Applications", *Artech House, Inc*. 685,Canton Street, Norwood, MA 02062

Young C. Lee (1986), "Analysis of Range and Position Comparison Methods as a Means to Provide GPS Integrity in the User Receiver", *Global Positioning System Vol. 5*, Institute of Navigation, Alexandria VA, pp 5-19

Gilbert Strang and Kai Borre (1997), "Linear Algebra, Geodesy and GPS", *Wellesley-Cambridge Press*, Box 812060, Wellesley MA 02181 USA