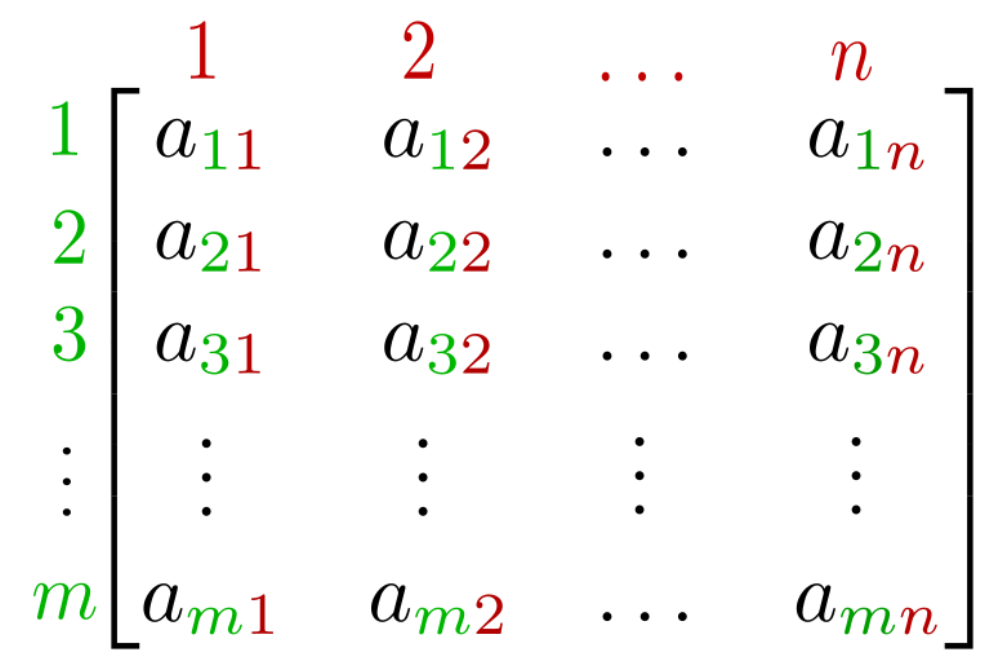
**BLOCKWISE INVERSION ALGORITHM**



D

C

B

A

Figure 1 n x n Matrix that needs to be divided into Blocks

* Figure 1 Shows the Matrix that we desire to Invert.
* First Task is to Divide the Matrix into Blocks A, B, C and D.
* After Inversion Our Task is to Find the Inverse of the Matrix using the formula illustrated in the Figure 2.

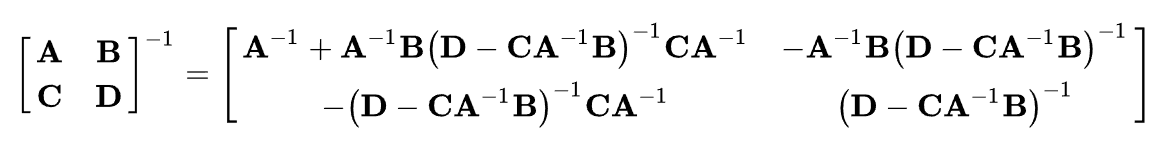


Figure 2 Blockwise Inversion Formula that gives the Inverse Matrix

Matrix that we Want to Invert =

* The Formula shown in figure 2 shows that each block’s calculation only requires the inverse of Matrix A and Schur Complement.
* The Expression is called the Schur Complement of the Matrix divided into blocks.

#Conditions for Inverse to exist

* **A** must be square, so that it can be inverted.
* Furthermore, **A** and **D** − **CA**−1**B** must be non-singular.
* We can recursively call the inverse function to calculate the Matrix inverse.
* This will considerably reduce the memory required for the whole operation
* But the Time required of multiple function calls will be more than other algorithms.

Flow of Algorithm

Figure 3 Flow of Blockwise Inverse Algorithm

#MATLAB Implementation of Block wise Inversion Algorithm

%Declare the Matrix

A = [1 2 3 6 3;2 5 4 1 3;1 3 5 1 5;3 2 1 4 2; 4 5 6 4 2];

%Calculate the Inverse of the Matrix Using MATLAB Built in Function

B = inv(A);

%Call the Blockwise Inversion function to calculate the inverse

Ainv = BlockwiseInv(A);

%Display the Inverse Matrix

disp(Ainv)

%%

%%Function Starts for Blockwise Inversion%%

function Ainv = BlockwiseInv(A)

[m,n]=size(A);

%Check if the matrix is sqaure matrix or not if not show error in inverting the matrix%

if m ~= n || rank(A) < n

fprintf("Inverse not posssible");

%If the Inverse Matrix is of Size 1 then Calculate inverse using below operation

elseif m == n && m == 1

Ainv = 1/A;

%If the matrix is 2x2 then invert it using the function made to invert the 2x2 matrix%

elseif m == 2 && n == 2

Ainv = TwoxTwo(A);

%Divide the Matrix into blocks and then Perform the inverse on it

elseif m == n && m>=2

%Divide the Matrix Block 1 into Matrix of n-1 x n-1 Called A

for i = 1:n-1

for j = 1:n-1

A1(i,j) = A(i,j);

end

end

%Divide the Matrix Block into Matrix B of size of n-1 x 1

for i=1:n-1

A2(i,1)=A(i,n);

end

%Divide the Matrix Block into Matrix C of size of 1 x n-1

for i=1:n-1

A3(1,i)=A(n,i);

end

%Final Element of n x n Matrix into D

A4 = A(n,n);

%Calculation of inverse of A

A1inv = BlockwiseInv(A1);

%Calculate the Schur Complement using A B C D blocks of the Matrix

SchurComp1 = SchurComp(A1,A2,A3,A4);

%Calculation of the Blockwise Inverse using the Formula shown in

%Above Explanation

B1 = A1inv+A1inv\*A2\*SchurComp1\*A3\*A1inv;

B2 = -1\*A1inv\*A2\*SchurComp1;

B3 = -1\*SchurComp1\*A3\*A1inv;

B4 = SchurComp1;

%Concatenate the Matrix to retrun the final Inverse Matrix

B12 = [B1 B2];

B34 = [B3 B4];

Ainv = [B12;B34];

end

end

%%

%Calculation of Schur Complement of the Matrix Based on Blocks ABCD

function Sch = SchurComp(A,B,C,D)

Sch1 = D - C\*BlockwiseInv(A)\*B;

Sch = BlockwiseInv(Sch1);

end

%%

%Inverse calculation of 2x2 Matrix

function a = TwoxTwo(A)

temp = 1/det(A);

if det(A) == 0

fprintf("Non Invertible matrix");

return

end

a(1,1) = A(2,2);

a(1,2) = -A(1,2);

a(2,1) = -A(2,1);

a(2,2) = A(1,1);

a = temp\*a;

end

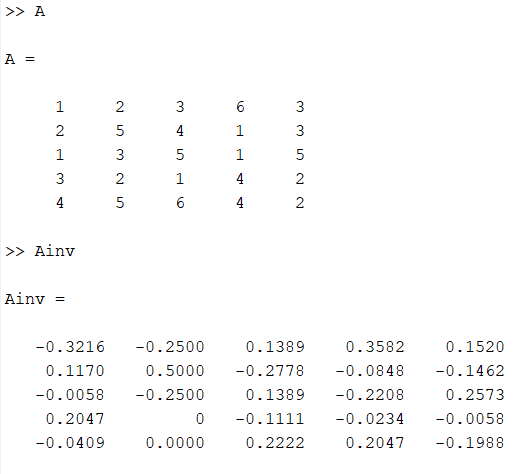


Figure 4 Output of the Above shown code

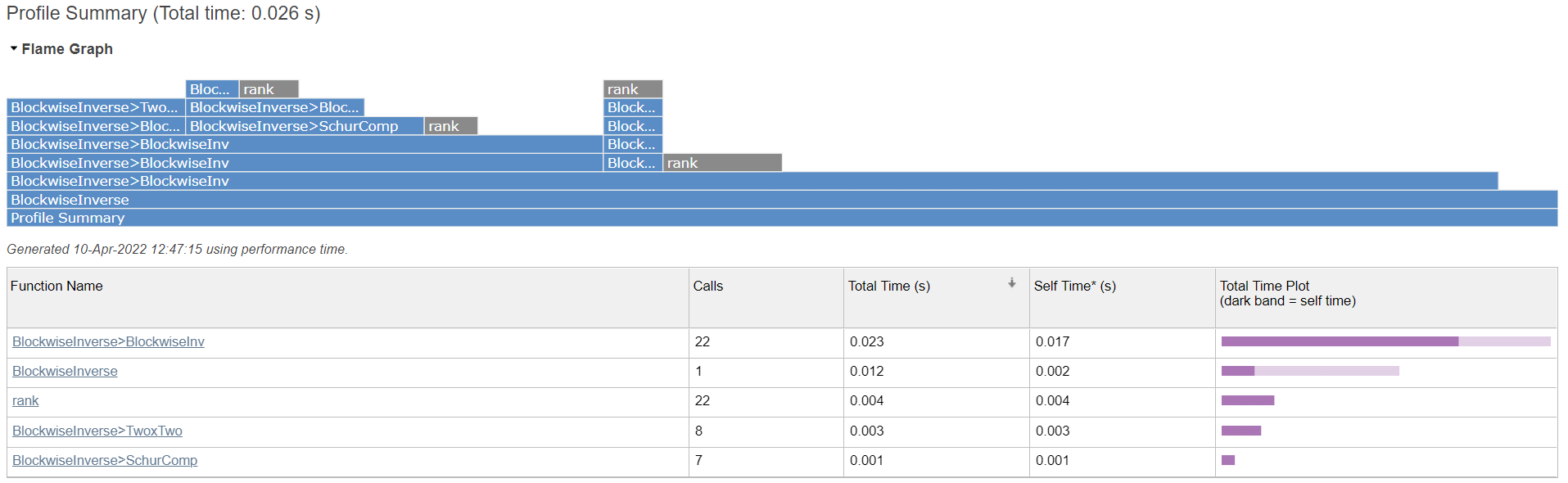


Figure 5 Profile Summary shows the total Time required to Finish the Inversion Operation is 0.026 seconds

**CHOLESKY DECOMPOSITION METHOD**

Cholesky inversion only applicable on positive definite matrices and using Cholesky decomposition we can factorize matrix in terms of lower triangular matrix and its transpose.

Our Aim will be decomposing the matrix into upper and lower triangular matrices. For this a standard formula will help us to calculate each entry of the matrices.

**A=[L][L\*]**

**A=[U][U\*]**

The Formula shown below is used to find the entries of the lower triangular matrix.



L denotes the lower triangular matrix with real and positive diagonal entries. Similar to that U is an upper triangular matrix.

A =

Now to calculate inverse of matrix, a standard algorithm is to find its Cholesky decomposition (decomposition into a lower-triangular and an upper-triangular matrix), use back substitution on the triangular pieces, and then

combine the results.

Our aim is to decompose the matrix into upper and lower triangle matrix and then calculate their inverses to calculate the overall inverse. This makes it overall computationally more efficient.

In below code this is implemented by directly assigning formula to l11 & U11 and a for loop to calculate remaining elements of first column (Lower) & first row (Upper).

Use the available elements and the formula to obtain the other elements of the matrix of the L, U.

**#MATLAB Code for Cholesky Decomposition**

%%

%Give the input as Matrix A of size n x n

%Obtain the length of A

%Declare the L matrix of size n x n filled with zeros

%Declare the U matrix of size n x n filled with zeros

%fill the first element of the A and L matrix equal to sqrt(A(1,1))

A = [1 2 3 6 3;2 5 4 1 3;1 3 5 1 5;3 2 1 4 2; 4 5 6 4 2];

N = length(A);

L = zeros(N,N);

U = zeros(N,N);

L(1,1) = sqrt(A(1,1));

U(1,1) = L(1,1);

%%

%Calculation of first row and first column of upper and lower matrices

for a=2:N

L(a,1) = A(a,1)/L(1,1);

U(1,a) = A(1,a)/L(1,1);

end

%%

%Use of the equation to calculate the other elements of L,U

for i=2:N

for j = i:N

if i == j

L(i,j) = sqrt(A(j,i)-L(j,1:i-1)\*U(1:i-1,i));

U(j,i) = L(j,i);

else

L(j,i)=(A(j,i)-L(j,1:i-1)\*U(1:i-1,i))/L(i,i);

end

end

for k = i+1:N

U(i,k) = (A(i,k)-L(i,1:i-1)\*U(1:i-1,k))/L(i,i);

end

end

%%

%calculating inverse

Linv = inv(L);

Uinv = inv(U);

Ainv = Uinv\*Linv

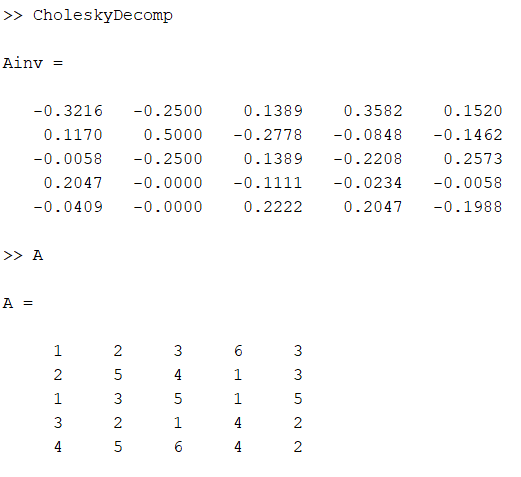


Figure 6 Output of the Above shown code

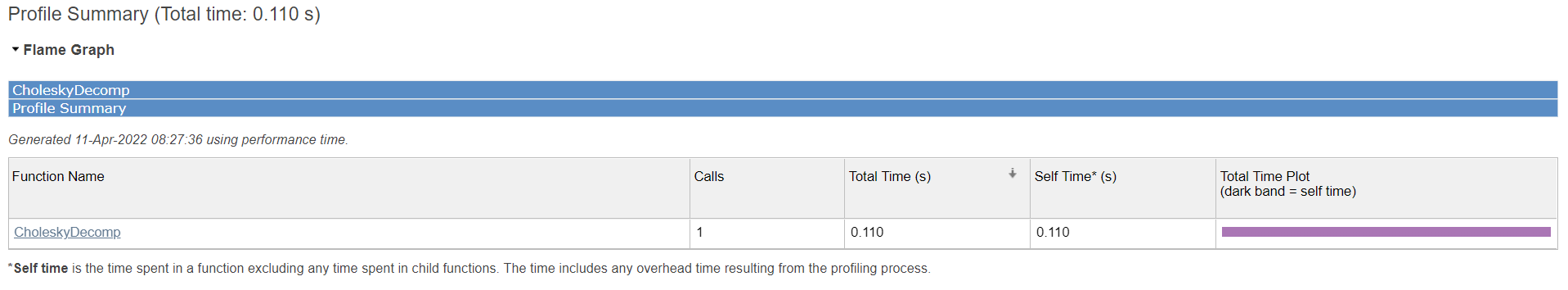
****

Figure 7 Profile Summary shows the total Time required to Finish the Inversion Operation is 0.11 seconds

**GAUSS JORDAN ELIMINATION METHOD**

* Gauss Jordan method is a variant method of Gaussian elimination.
* In this we perform the row reduction technique under some rules to obtain the inverse of the matrix.
* First step to obtain that will be to form the augmented matrix.
* An Augmented matrix is extended n x n matrix attached side by side to our original matrix.
* This augmented matrix will be an Identity Matrix that is of same size of our original matrix.
* Now one by one we perform the row reduction to obtain the echelon form.

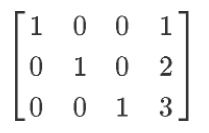


Figure 8 Example of Row reduced Echelon form

The following row operations are performed on augmented matrix when required:

• Interchange any two row

• Multiply each element of row by a non-zero integer

• Replace a row by the sum of itself and a constant multiple of another row of the matrix

Augmented Matrix = [A|I]

* We play around with rules as stated above to make the matrix A into identity matrix. And the augmented part will change itself to give the Inverse of the matrix.

**#PSUEDO CODE FOR THE GAUSS JORDAN METHOD**

1. Start

2. Read Order of Matrix (n).

3. Read Matrix (A):

For i = 1 to n

For j = 1 to n

Read Ai,j

Next j

Next i

4. Augment Identity Matrix of Order n to Matrix A:

For i = 1 to n

For j = 1 to n

If i = j

Ai,j+n = 1

Else

Ai,j+n = 0

End If

Next j

Next i

5. Apply Gauss Jordan Elimination on Augmented Matrix (A):

For i = 1 to n

If Ai,i = 0

Print "Mathematical Error!"

Stop

End If

For j = 1 to n

If i ≠ j

Ratio = Aj,i/Ai,i

For k = 1 to 2\*n

Aj,k = Aj,k - Ratio \* Ai,k

Next k

End If

Next j

Next i

6. Row Operation to Convert Principal Diagonal to 1.

For i = 1 to n

For j = n+1 to 2\*n

Ai,j = Ai,j/Ai,i

Next j

Next i

7. Display Inverse Matrix:

For i = 1 to n

For j = n+1 to 2\*n

Print Ai,j

Next j

Next i

8. Stop

**#MATLAB CODE FOR GAUSS JORDAN METHOD**

A = [1 2 3 6 3;2 5 4 1 3;1 3 5 1 5;3 2 1 4 2; 4 5 6 4 2];

n = 5;

B = A;

%Augment Identity Matrix of Order n to Matrix A

for i=1:n

for j=1:n

if i == j

A(i,j+n)=1;

else

A(i,j+n)=0;

end

end

end

%Apply Gauss Jordan Elimination on Augmented Matrix (A)

for i = 1:n

if A(i,i) == 0

fprintf("Mathematical Error");

break;

end

for j = 1:n

if i~=j

Ratio = A(j,i)/A(i,i);

for k=1:2\*n

A(j,k) = A(j,k) - Ratio\*A(i,k);

end

end

end

end

%Row Operation to Convert Principal Diagonal to 1

for i = 1:n

for j=n+1:2\*n

A(i,j)=A(i,j)/A(i,i);

end

end

% Writing the whole Display Inverse Matrix

for i = 1:n

for j = n+1:2\*n

C(i,j) = A(i,j);

end

end

Y = inv(B);

disp(C)

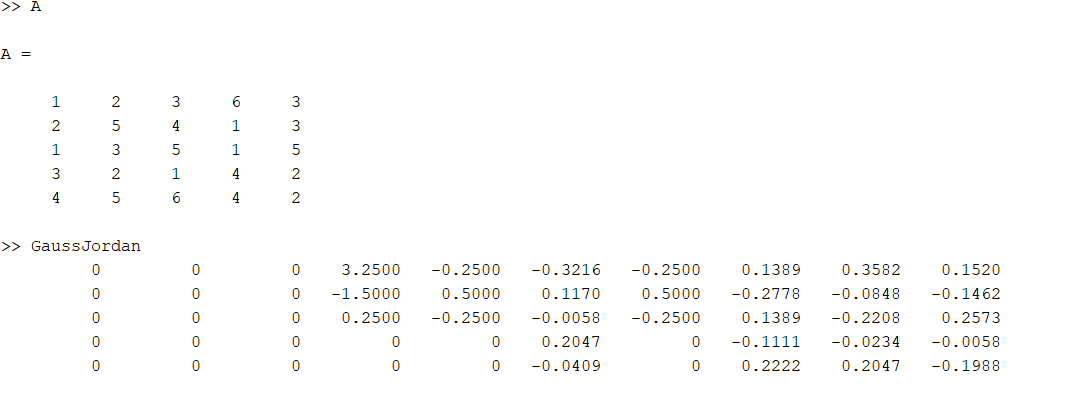


Figure 9 Output of the Above shown code

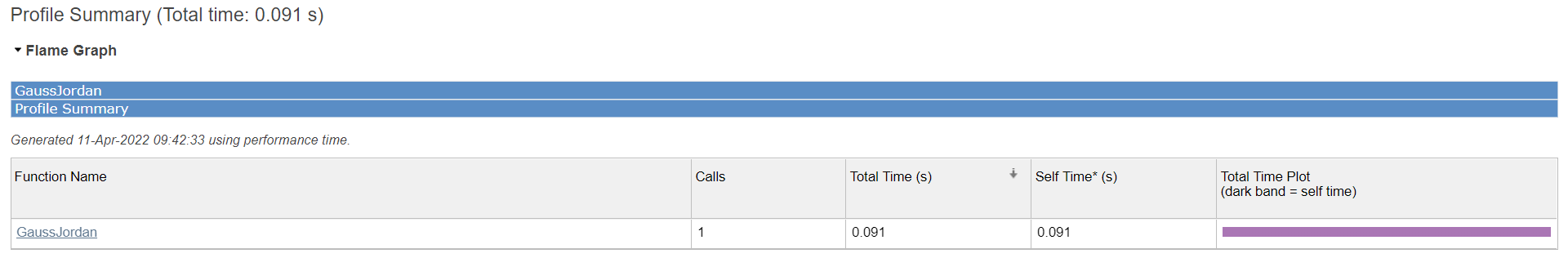


Figure 10 Profile Summary shows the total Time required to Finish the Inversion Operation is 0.091 seconds