DSP Lab

ASSIGNMENT 5 — Matrix Inverse Algorithms

Akhil Kumar Donka
EE22MTECH02003

April 10, 2022

Problem

To implement the below mentioned matrix inversion algorithms :

- Blockwise Inverse
- Cholesky decompositon Inverse
- Gauss Jordan Method

---

1. Blockwise Inverse

Blockwise inverse calculation is a recursive procedure, where we first split out given matrix (k x k) into 4 sub-block (order = b) matrices called frames using forward and backward recursive procedures. Forward procedure goes till we reach k-2 steps where resulting frames will be of size 2 x 2. Then in backward procedure, for all 4 frames generated, operations are carried out to generate single block.

Consider :

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

$$M^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \tag{2}$$

From the result of blockwise inversion, we get:

$$M^{-1} = \begin{bmatrix} (M/D)^{-1} & -(M/D)^{-1}BD^{-1} \\ -D^{-1}C(M/D)^{-1} & D^{-1} + D^{-1}C(M/D)^{-1}BD^{-1} \end{bmatrix} \tag{3}$$

where $(M/D)^{-1}$ is inverse of Schur complement of D in M

So, to get N_{11} of inverse matrix, we successively split matrix into its 4 frames and for each frame generated we apply Schur compliment to reduce them into single frame.

MATLAB function has been created which accepts three parameters, i.e. Matrix, size of sublevel frames for each iteration, check flag. Check flag is only for recursive calls of function which makes sure that matrix passed for computation is valid. For each iteration, We start by declaring an identity matrix of size similar to M. We get frame size for m_{11} and also we create other three frames as well. And then we make recursive function call on "H" frame of inverse matrix and apply mentioned equations on other frames i.e. E, F, G which uses the frame "H" in calculations. Then we create the final inverse matrix using above frames and return it from function.

MATLAB Code

```
%% Function calling

size = 6;
Matrix = randn(size);
blocksizes =  2 * ones(1, 3);
% calling inbuilt inverse operation for comparing
invM = inv(Matrix);
blockInvM = blockInverse(Matrix, blocksizes);
fprintf("Error : %.4f\n", norm(blockInvM-invM));

%% function for block inversion

function m_inverse = blockInverse(M, framesizes, pass)

    if nargin < 3 || ~pass
        % to check whether matrix passed to function valid or not
        assert(ismatrix(M));
        assert(sum(framesizes) == size(M, 1) && size(M, 1) == size(M, 2));
    end

    if length(framesizes) == 1
        % at last recursive call
        m_inverse = M \ speye(size(M));
    else
        % getting framesize for each iterations
        frameSize = framesizes(1);
        % Slicing matrix into four sublevel matrices
        m11 = M(1:frameSize, 1:frameSize);
        m12 = M(1:frameSize, frameSize+1:end);
        m21 = M(frameSize+1:end, 1:frameSize);
        m22 = M(frameSize+1:end, frameSize+1:end);

        %applying block inversion formula
        invA = m11 \ speye(size(m11));
        Ainv_B = m11 \ m12;
        C_Ainv = m21 / m11;
        % recursive call
        n22 = blockInverse(m22 - m21 * (m11 \ m12), framesizes(2:end), true);
        n21 = -n22 * C_Ainv;
```

```
        n12 = -Ainv_B * n22;
        n11 = invA - n12 * C_Ainv;

        % computed matrix inverse at each recursive calls and parent call
        m_inverse = [n11 n12; n21 n22];
    end

end
```

2. Cholesky decompositon Inverse

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.

$$M = LL^* \tag{4}$$

or equivalently in terms of upper triangular matrix ($L = U^*$) and its transpose.

$$M = U^*U \tag{5}$$

Or we can say :

$$M = LU \tag{6}$$

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 subsitution on the triangular pieces, and then combine the results finally. Consider the below decomposition :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \tag{7}$$

Now from right hand side calculations of above matrix, we can get values of first column of lower triangular matrix and first row of upper triangular matrix. In below code this is implemented by directly assigning formula to $Lower_{11}$ & $Upper_{11}$ and a for loop to calculate remaining elements of first column (Lower) & first row (Upper).

Similarly we get equations of other elements of both lower and upper triangular matrices in terms of known elements and apply them using another for loop.

Since calculating inverse for lower and upper triangular matrices is computationally efficient, we calculate their inverse and use below equation to compute inverse of given matrix.

$$M^{-1} = U^{-1}L^{-1} \tag{8}$$

MATLAB Code

```
%% Function calling
```

```
size = 6;
% creating a positive definite matrix
Matrix = complex(randn(size), randn(size)) ;
Matrix = Matrix'*Matrix;
% calling inbuilt inverse operation for comparing
invM = inv(Matrix);
choleskyInvM = choleskyInverse(Matrix);
fprintf("Error : %.4f\n", norm(choleskyInvM-invM));

%% Function defination
function invMat = choleskyInverse(Matrix)
    N = length(Matrix);
    % initialised lower and upper matrices
    Lower = zeros(N,N);
    Upper = zeros(N,N);

    % applying known equations on l_11 and u_11
    Lower(1,1) = sqrt(Matrix(1,1));
    Upper(1,1) = Lower(1,1);

    % getting other values of first column of lower and first row of upper
    for a=2:N
        Lower(a,1) = Matrix(a,1)/Lower(1,1);
        Upper(1,a) = Matrix(1,a)/Lower(1,1);
    end

    % getting remaining elements
    for i=2:N
        for j = i:N
            if i == j
                Lower(i,j) = sqrt(Matrix(j,i)-Lower(j,1:i-1)*Upper(1:i-1,i));
                Upper(j,i) = Lower(j,i);
            else
                Lower(j,i)=(Matrix(j,i)-Lower(j,1:i-1)*Upper(1:i-1,i))/Lower(i,i);
            end
        end
        for k = i+1:N
            Upper(i,k) = (Matrix(i,k)-Lower(i,1:i-1)*Upper(1:i-1,k))/Lower(i,i);
        end
    end
```

```
    %calculating inverse

    Linv = Lower \ speye(size(Lower));
    Uinv = Upper \ speye(size(Upper));
    invMat = Uinv*Linv;
end
```

3. Gauss Jordan Method

In this method, for below given equation we play around with the rows until we make Matrix A into the Identity Matrix I.

$$[A|I] : AugmentedMatrix \tag{9}$$

This is carried out by doing elementary operations on augmented matrix which include:

- swap rows
- multiply or divide each element in a a row by a constant
- replace a row by adding or subtracting a multiple of another row to it

Algorithm

In order or carry out this method following steps needs to be followed:

1. Form augmented matrix using an identity matrix of same size
2. Perform the row reduction operation on this augmented matrix to generate a row reduced echelon form of the matrix
3. 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

MATLAB Code

```
%% Function calling

size = 6;
Matrix = randn(size);
% calling inbuilt inverse operation for comparing
invM = inv(Matrix);
gaussInvM = gaussInverse(Matrix);
fprintf("Error : %.4f\n", norm(gaussInvM-invM));
```

```
%% Function defination

function invMatrix =  gaussInverse(M)
    % getting size of matrix
    [rows, ~]  = size(M);
    % creating identity matrix for augumentation
    invMatrix = eye(rows);
    for j = 1 : rows
        for i = j : rows
            if M(i,j) ~= 0
                for k = 1 : rows
                    s = M(j,k);
                    M(j,k) = M(i,k);
                    M(i,k) = s;
                    s = invMatrix(j,k);
                    invMatrix(j,k) = invMatrix(i,k);
                    invMatrix(i,k) = s;
                end
                t = 1/M(j,j);
                for k = 1 : rows
                    M(j,k) = t * M(j,k);
                    invMatrix(j,k) = t * invMatrix(j,k);
                end
                for L = 1 : rows
                    if L ~= j
                        t = -M(L,j);
                        for k = 1 : rows
                            M(L,k) = M(L,k) + t * M(j,k);
                            invMatrix(L,k) = invMatrix(L,k) + t * invMatrix(j,k);
                        end
                    end
                end
            end
            break
        end
        if M(i,j) == 0
            disp('Warning: Singular Matrix')
            invMatrix = Inf(rows);
            return
        end
```

```
    end
end
```

Algorithm Complexity

	Time Complexity
Blockwise Inverse	$O(n^{\epsilon})$; $\epsilon \geq 2$
Cholesky Decomposition Inverse	$O(n^3)$
Gauss Jordan Inverse	$O(n^3)$