1 Matrix inversion

- *Algorithm:* Cholesky Algorithm (algo. 1), Levinson-Durbin Algorithm (algo. 2)
- *Input:* A square matrix A
- *Complexity:* $\mathcal{O}(n^3)$
- *Data structure compatibility:* N/A
- *Common applications:* Finding inverse, solving linear systems

Problem. Matrix inversion
Finding the inverse of a matrix

Description

Two matrices are named inverse to each other if their product is the identity matrix of their size. That is to say

$$AA^{-1} = I_n$$

Finding the inverse of a matrix is very important in linear algebra and real life practices, so a good inversion method is necessary. The most intuitive method must be euqation solving, which means building a linear system with regard to vectors and solve it using elimination and substitution. But this can be time-consuming.

However, based on some special features of matrix, the time of calculating the inverse could be reduced.

Cholesky algorithm is a inverse finding algorithm for Hermittian matrices. A $n \times n$ matrix A is said to be Hermittian if it is conjugate symmetric. In another word

$$a_{ij} = \bar{a_{ji}} \text{ for } 1 \leq i, j \leq n$$

And if the matrix is a real matrix, being Hermittian is equivalent to being symmetric.

Cholesky algorithm is based on the Cholesky decomposition of a Hermittian. Cholesky decomposition is decomposing a matrix into the product of an upper triangular matrix with its conjugate transpose. In a word, it is like

$$A = R^*R$$

where R is upper triangular.

After decomposing, the original equation can be divided into two equation systems .

$$R^*RA^{-1} = I_n \Rightarrow \begin{cases} R^*y & = I_n \\ RA^{-1} & = y \end{cases}$$

Solving this equation system is simpler since the coefficient matrices are now tiangular matrices and consequently methods like forward/backward substitution can be directly applied. What's more, based on the

Hermittian property the amount of operations can be reduced by nearly half. The decomposition part requires $\frac{n^3}{6}$ operations and the back substitution is a $\mathcal{O}(n^2)$ process, so the complexity for Cholesky algorithm is $\mathcal{O}(n^3)$

Another algorithm is Levinson-Durbin algorithm. This algorithm is designed for a special type of matrices called symmetric Toeplitz matrices. Toeplitz matrices are suqare matrices who have same entries on every diagonals(not only main diagonal), which is to say

$$A_{i,j} = a_{i-j} \text{ for } 1 \leq i, j \leq n$$

where $\{a_k\}$ is a constant sequence whose index ranges from $-(n-1)$ to $(n-1)$

Levinson-Durbin algorithm uses the idea of recursion. Using recursive call, it first finds the inverse of the $(n-1) \times (n-1)$ matrix on the left-top of A(denoted as A_{n-1}), denoted as A_{n-1}^{-1}. Then it adds a row of 0 at the the bottom of A_{n-1}^{-1}.

It is clear that if you multiply the result of such operation with A, the result will only have differences with the first $(n-1)$ columns in I_n on the last row.

The next step is to find the backward vector of A, which is a vector b_n such that (here e_n stands for the n_{th} column in the identity matrix I_n)

$$Ab_n = e_n$$

After finding this backward vector, weight it and add it to columns in A_{n-1}^{-1} to eliminate the errors in the last row and the append it to the right of A_{n-1}^{-1}, you will get A^{-1}.

The process of finding b_n is also recursive. And here another term called forward vector needs to be introduced, which is a vector f_n which satisfies

$$Af_n = e_1$$

Because the matrices studied are symmetric Toeplitz matrices, f_n is just the reverse of b_n, which means

$$f_n[i] = b_n[n-i+1] \text{ for } 1 \leq i \leq n$$

Here is the process of finding b_n. First use recursive call to find b_{n-1}, and generate f_{n-1} by reversing b_{n-1}. Append a 0 at the end of f_{n-1} and the beginning of b_{n-1}, then the product of A and the new two vectors(denoted as $f_n^{'}, b_n^{'}$) will only have difference with e_1 and e_n on the last/first row respectively.

Denote the deviations as δ_n, δ_1, it can be seen that using linear combinations of these two vectors can eliminate the deviations and generate b_n

$$b_n = \frac{1}{1-\delta_1\delta_n} b_n^{'} - \frac{\delta_1}{1-\delta_1\delta_n} f_n^{'}$$

Levinson recursion originally serves to solve linear systems, and in that case it is of $\mathcal{O}(n^2)$ complexity. Because here the constant terms are vectors, the complexity will be $\mathcal{O}(n^3)$

Algorithm 1: Cholesky Algorithm

Input : A Hermittian square matrix A

Output: The inverse of A

```
Function Cholesky(A):
    n ← size of A;
    R ← size-n square matrix filled with zeros;
    for i ← 1 to n do
        for j← i to n do
            for k← 1 to i-1 do
                R[i,j] ← R[i,j] + R[k,i]*R[k,j];
            end for
            if i≠j then
                R[i,j] ← (A[i,j]−R[i,j]) / R[i,i] ;
            else
                R[i,j] ← √(A[i,j] − R[i,j]) ;
            end if
        end for
    end for
    X ← size-n square matrix filled with zeros;
    for i ← n to 1 do
        for j← i to n do
            X[i,j] ← Σ_{k=i+1}^{n} R[i,k]X[k,j];
            if i=j then
                X[i,j] ← (1/R[i,j] − X[i,j]) / R[i,j];
            else
                X[i,j] ← (0−X[i,j]) / R[i,j];
                X[j,i] ← X[i,j]*;
            end if
        end for
    end for
    return X
end
return Cholesky(A)
```

Algorithm 2: Levinson-Durbin Algorithm

Input : A Symmetric Toeplitz matrix A

Output: The inverse of A

Function `LevinsonDurbin(`A`)`**:**
    n $\leftarrow$ size of A;
    if $n=1$ **then**
        return $[\frac{1}{A[1,1]}]$
    B $\leftarrow$ `LevinsonDurbin(`$A[1:n-1, 1:n-1]$`)`;
    Append a row of zeros to the bottom of B;
    Append `BackwardVector(`A`)` to the right of B;
    for $i \leftarrow 1$ **to** $n\text{-}1$ **do**
        d $\leftarrow \sum_{k=1}^{n} A[n,k]B[k,i]$;
        for $j \leftarrow 1$ **to** n **do**
            $B[j,i] \leftarrow B[j,i] - d \times B[j,n]$;
        end for
    end for
    return B
end

Function `BackwardVector(`A`)`**:**
    n $\leftarrow$ size of A;
    if $n=1$ **then**
        return $[\frac{1}{A[1,1]}]$
    b $\leftarrow$ `BackwardVector(`$A[1:n-1, 1:n-1]$`)`;
    Append a zero at the head of b;
    f $\leftarrow$ b reversed;
    $dn \leftarrow \sum_{k=1}^{n} A[n,k]f[k]$;
    $d1 \leftarrow \sum_{k=1}^{n} A[1,k]b[k]$;
    $b \leftarrow \frac{1}{1-d1\times dn}b - \frac{d1}{1-d1\times dn}f$;
    return b
end

return `LevinsonDurbin(`A`)`

References

- Aravindh Krishnamoorthy, Deepak Menon (2013), "Matrix Inversion Using Cholesky Decomposition"
- Trench, W. F. (1964). "An algorithm for the inversion of finite Toeplitz matrices." J. Soc. Indust. Appl. Math., v. 12, pp. 515522.
- Musicus, B. R. (1988). "Levinson and Fast Choleski Algorithms for Toeplitz and Almost Toeplitz Matrices." RLE TR No. 538, MIT