Matrices with Structured Non Overlapping diagonal Blocks

Debajoy Mukherjee

July 9, 2024

1 Problem Description

Consider a matrix A of size $nd \times nd$, where each non-overlapping $d \times d$ block of the matrix, D_{ij}, is a diagonal matrix. So the matrix consists of n^2 such blocks. An example of such a matrix is shown below:

$$\begin{bmatrix} D_{11} & D_{12} & D_{13} & \cdots & D_{1n} \\ D_{21} & D_{22} & D_{23} & \cdots & D_{2n} \\ D_{31} & D_{32} & D_{33} & \cdots & D_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{n1} & D_{n2} & D_{n3} & \cdots & D_{nn} \end{bmatrix}$$

Construct an efficient data structure to represent such matrices and devise algorithms to perform matrix operations, such as matrix multiplications and matrix inverse, on the data structure you designed. Provide a technical write-up of your solution along with associated code implementing your solution.

2 Diagonal Matrix

Diagonal Matrices are matrices such that off-diagonal elements are zero. The Properties of a Diagonal matrix are closed under addition and scalar multiplication. We claim that every diagonal matrix can be represented as a vector containing the diagonal elements.

Multiplication Multiplication of 2 diagonal matrices will be a new diagonal matrix such that the new diagonal is a product of the 2 diagonals. This can be achieved by an element-wise product of the 2 vectors.

Addition Addition of 2 diagonal Matrices is a diagonal matrix with the new diagonal being the sum of the 2 diagonals. This can be achieved by element-wise addition of the two vectors.

Determinant The determinant is a product of the diagonals. The matrix is invertible if the determinant is non-zero. This is just the product of the elements of the vector.

These properties let us mainly treat the diagonal matrices as vectors and the vectors as individual elements of a matrix.

3 Matrices with diagonal matrices as the elements

This is the step in which we treat every element of the 2d matrix of size nxn as a diagonal matrix. A diagonal matrix is represented as a data structure that contains the diagonal elements as the entries of a vector. Multiplication between these 2 vectors is closed and is just element-wise multiplication and so is the addition and subtraction operation. With these operations defined, we can apply simple matrix rules to multiply 2 matrices.

4 Matrix Operations

So now we have a data structure of n*n matrix where every element is a d-size vector. The d-size vector represents a diagonal matrix. Now the question remains how to define basic operations like matrix multiplication and matrix inversion using this new representation.

4.1 Matrix Multiplication

Matrix multiplication can be thought of the simple operation:

$$A_{jk} = \sum_k a_{jk} b_{kj}$$

Now we know that the individual elements of this matrix are vectors of d length. This implies that matrix multiplication is well-defined if we can define the multiplication and addition operations. Now that we have already defined those operations. This operation is possible.

4.2 Order of Complexity Analysis

Generally to multiply a n*d x n*d matrix, the order of complexity is $O(n^3 d^3)$ but now as the matrix is just n^2 we can just think of the operation :

$$\sum_k a_{jk} b_{kj}$$

The addition operation has an order of complexity $O(d)$ and so is the multiplication operation since it is element-wise. Now, given this to compute just one element A_{jk} we have to do n times addition operation of a multiplication operation. So a multiplication operation is $O(d)$ and an addition operation is $O(d)$ so this results in $O(nd)$. So computing n^2 such elements results in $O(n^3 d)$. This is a significant boost in performance.

4.3 Matrix inverse

To find the matrix inverse we perform the Gauss-Jordon method of finding inverse. This method is also in order of $O(n^3 d^3)$ but with our new efficient data structure which is a naive way to utilize the structure of the matrix, we can perform the same operations using diagonal matrices as individual elements. If we think about it the method mainly 2 main operations. 1. Dividing an entire row with a scalar to make the A_{ii} element of the ith row 1. This operation can be performed in $0(n)$ steps for a simple nxn matrix. For our case, this operation is $0(n * d)$. Then we use this row to make all elements of ith column zero. This is done in $0(n * n)$. This can be done in $O(n^2 d)$. We have n such operations. So resulting complexity is $0(n^3 d)$.

One more important caveat to this problem is that is the row exchange operation required, or in what cases would the inverse operation not be valid? This happens essentially for matrix operations when the pivot element is 0. For us, this is the case for the determinant of the pivot element is 0. This is the time when we can allow row exchanges.

5 Conclusion

In conclusion, we have developed a paradigm in which it is easier to extend this approach to solve problems of similar type with structured matrix structure. The entire code is developed in python but to make it faster, future extensions could be done to compute the inverse in C++.