EE4013 Assignment-1

Shaik Mastan Vali - EE18BTECH11039

Download all the codes from

https://github.com/Mastan1301/EE4013/tree/main/assignment−1/codes

and latex-tikz codes from

https://github.com/Mastan1301/EE4013/tree/main/assignment−1

1 Problem

Consider the following matrix:

$$X = \begin{bmatrix} 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \\ 1 & 5 & 25 & 125 \end{bmatrix}$$

Calculate the absolute value of the product of the Eigen values of X.

2 Solution-1

We use the following property -

$$\prod_{i=0}^{k-1} \lambda_i = det(X)$$

where λ_i are the Eigen values and $det(.)$ is the determinant operator.

2.1 Algorithm

To find the determinant, we use the following recurrence relation.

$$det(X, n) = \sum_{j=0}^{n-1} (-1)^j \times X[0][j] \times det(cof(X, 0, j), n-1)$$

where $cof(i, j)$ is the cofactor matrix of the position (i, j) in the matrix R.

The recursion, for our example, is given in the flow chart.

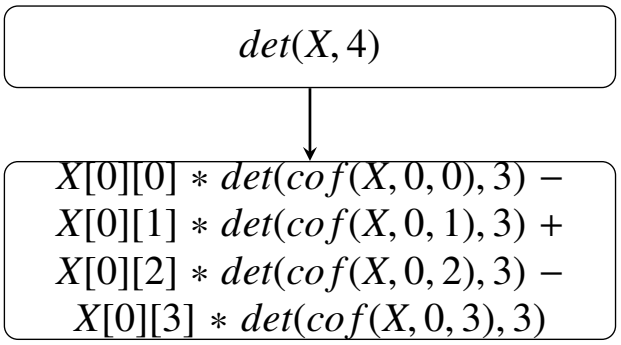

Fig. 0: The flow of recursion

To understand the process, consider $C_1 = cof(X, 0, 0)$.

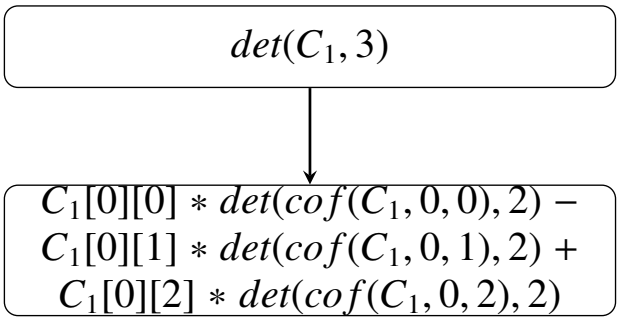

Fig. 0: A 3×3 sub-problem

Now, consider $C_2 = cof(C_1, 0, 0)$.

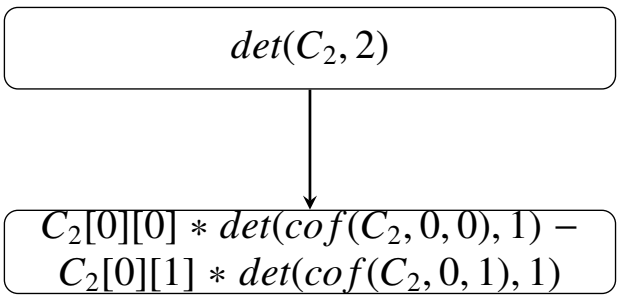

Fig. 0: A 2×2 sub-problem

The determinant of a 1×1 matrix is the value of

the element itself. We use this property as the base case.
The absolute value of the determinant of the given matrix is 12. Hence, the answer is 12.
The algorithm is in the following file -

https://github.com/Mastan1301/EE4013/tree/main/assignment-1/codes/det/main.c

2.2 Complexity of the algorithm

At each stage of recursion of depth k, there are $O(n-k)$ recursive calls. Hence, the time complexity is

$$O(n \times (n-1) \times .... \times 1) = O(n!)$$

3 Solution-2

In this solution, we use the QR-algorithm to compute the Eigen values. The algorithm is as follows-

- Apply the QR decomposition so that

  $$X = Q_0 R_0$$

  where Q_0 is an orthogonal matrix and R_0 is an upper-triangular matrix.
- Compute

  $$E_0 = Q_0^{-1} X Q_0 = Q_0^T X Q_0$$

  Note that the Eigen values of X and E_0 are the same.
- Next, we decompose the E_0 and repeat this procedure till we obtain a triangular matrix. The Eigen values of a triangular matrix are given by the diagonal entries of the matrix.

We compute the QR decomposition using the Householder transformation.

3.1 Householder Transformation

The reflection hyperplane can be defined by a unit vector u that is orthogonal to the hyperplane, which is its normal vector. The reflection of a vector x about this hyperplane is the linear transformation:

$$\begin{aligned} y &= x - 2(u^T x)u \\ &= x - 2u(u^T x) \\ &= x - 2(uu^T)x \\ &= (I - 2uu^T)x \end{aligned}$$

The matrix constructed from this transformation can be expressed in terms of an outer product as:

$$H = H = I - 2uu^T$$

is called as the **Householder matrix**.

3.2 Algorithm

We compute the QR decomposition using the Householder transformation. Multiplying a given vector x, for example the first column of matrix X, with the Householder matrix H, which is given as

$$H = I - \frac{2}{u^T u} uu^T$$

reflects x about a plane given by its normal vector u. When the normal vector of the plane u is given as

$$u = x - \|x\|_2 e_1$$

then the transformation reflects x onto the first standard basis vector

$$e_1 = \begin{bmatrix} 1 & 0 & 0 & \dots \end{bmatrix}^T$$

which means that all entries but the first become zero.
In general, if we take $u = x - s\|x\|e_1$ where $s = \pm 1$ and $v = u/\|u\|$ then

$$Hx = \left(I - 2\frac{uu^T}{u^T u}\right) x = s\|x\|e_1.$$

Let us first verify that this works:

$$\begin{aligned} u^T x &= (x - s\|x\|e_1)^T x \\ &= \|x\|^2 - s x_1 \|x\| \\ u^T u &= (x - s\|x\|e_1)^T (x - s\|x\|e_1) \\ &= \|x\|^2 - 2s x_1 \|x\| + \|x\|^2 \|e_1\|^2 \\ &= 2\left(\|x\|^2 - s x_1 \|x\|\right) = 2u^T x \end{aligned}$$

$$Hx = x - 2u\frac{u^T x}{u^T u} = x - u = s\|x\|e_1.$$

As a byproduct of this calculation, note that we have

$$u^T u = -2s\|x\|u_1 = -2x_1 + 2s\|x\|$$

Hence, to ensure $u^T u > 0$ always, we take $s = -sign(x_1)$.

$$\implies Hx = -sign(x_1)\|x\|e_1$$

First, we multiply X with the Householder matrix H_1 we obtain when we choose the first matrix column for X. This results in a matrix H_1X with zeros in the left column (except for the first row).

$$H_1X = \begin{bmatrix} -sign(X_{11})\alpha_1 & * & \cdots & * \\ 0 & & & \\ \vdots & & X' & \\ 0 & & & \end{bmatrix}$$

where α_1 is the 2-norm of the first column of X. This can be repeated for X' (obtained from H_1X by deleting the first row and first column), resulting in a Householder matrix H'_2. Note that H'_2 is smaller than H_1. Since we want it really to operate on H_1X instead of X' we need to expand it to the upper left, filling in a 1 , or in general:

$$H_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & H'_k \end{bmatrix}$$

This is how we can, column by column, remove all sub-diagonal elements of A and thus transform it into R.

$$H_n \dots H_3H_2H_1X = R$$

The product of all the Householder matrices H, for every column, in reverse order, will then yield the orthogonal matrix Q.

$$H_1H_2H_3 \dots H_n = Q$$

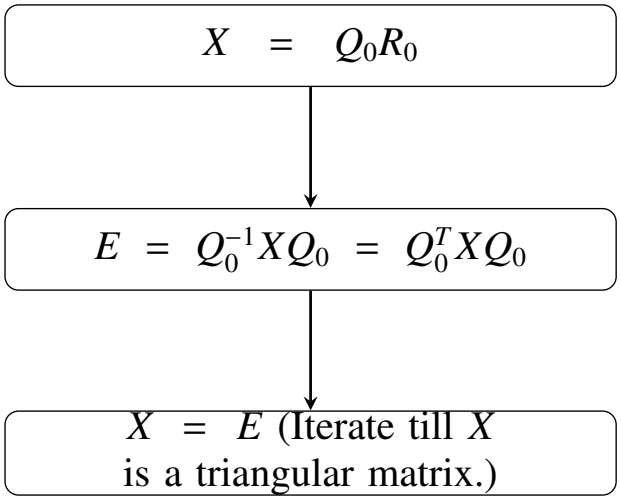

Fig. 0: The QR algorithm

Using this algorithm, the obtained Eigen values are 139.410, 5.003. 0.555, 0.0311. The absolute value of the product is 12.
The algorithm is in the following file-

https://github.com/Mastan1301/EE4013/tree/main/assignment−1/codes/QR_arrays/main.c

The utility functions are in the following header file-

https://github.com/Mastan1301/EE4013/tree/main/assignment−1/codes/QR_arrays/utils.h

Compile the code using-

```
gcc main.c −o main −lm
```

Run the code using-

```
./main
```

The C++ program (using vectors) is in the following file-

https://github.com/Mastan1301/EE4013/tree/main/assignment−1/codes/QR_vectors/main.cpp

Compile the code using-

```
g++ main.c −o main −lm
```

Run the code using-

```
./main
```

3.3 Complexity of the QR algorithm using arrays/vectors

The QR decomposition step takes $O(n^4)$ computations, because the matrix multiplication at each iteration takes $O(n^3)$ time, and there are $O(n)$ iterations.
The matrix convergence is achieved in $O(\max_{i,j}|\lambda_i/\lambda_j|^n)$ steps. Hence the total time complexity is $O((\max_{i,j}|\lambda_i/\lambda_j|^n) \times n^4)$.

3.4 QR algorithm using linked lists

In this algorithm, we use linked lists to store the matrices. The list is defined by a *head* node. The function *at(head, i, j)* returns the pointer to the node at i^{th} row and j^{th} column.
The code is in the following file-

https://github.com/Mastan1301/EE4013/tree/main/assignment−1/codes/QR_linked_list/main.c

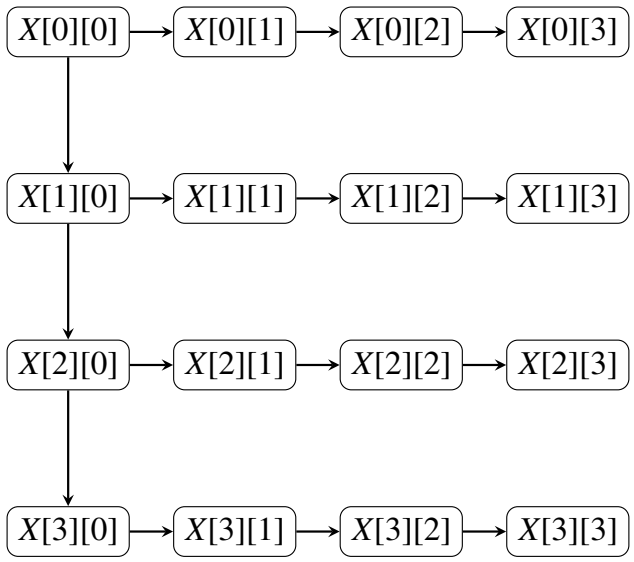

Fig. 0: A 4×4 matrix implementation using linked list

3.5 Complexity of the QR algorithm using linked lists

To access the element in the i^{th} row and j^{th} column, we need $O(i + j)$ iterations. Hence, the time complexity of accessing an element is $O(n)$. This additional factor in the algorithm gives us $O((\max_{i,j} |\lambda_i/\lambda_j|^n) \times n^5)$ complexity. However, using lists gives us better memory management due to non-continuguous memory allocation.

3.6 Comparison of runtimes of QR algorithm using different data structures

Data Structure	**Time (in ms.)**
Arrays	0.472
Vectors (in C++)	0.886
Linked lists	0.925